**蝴蝶开发者指南**

# 介绍

蝴蝶系统面向第三方应用提供通知推送服务。它具备如下特点：

* 负责将第三方应用服务器的通知（Notification）推送给特定的客户端。
* 提供消息的递送状态报告
* 通知（Notification）为一小段二进制数据（不大于2K）。对于大数据， 第三方客户端可以在接到通知后使用后续的操作直接与第三方服务器交互。
* 蝴蝶系统只负责通知的路由和推送， 不会对通知内容进行任何操作。
* 蝴蝶系统提供了消息的路由、暂存和推送功能。第三方客户端无须实现长时间保持在线或者轮询消息的机制。
* 蝴蝶客户端组件的默认实现为长连接IP Push。设备上的多个应用会共用同一个长连接，以节省系统资源。

# 整体结构

蝴蝶的整体结构示意图如下：

整个系统中参与的角色有：

|  |  |
| --- | --- |
| **角色** | |
| **蝴蝶服务器** | 负责接受第三方应用服务器的消息，并将消息推送到终端上的服务器。 |
| **第三方应用服务器** | 第三方应用的服务器端程序。负责发送通知。 |
| **第三方应用客户端应用** | 运行于终端中的程序。一般为第三方的客户端。接受来自蝴蝶系统推送的通知。 |
| **蝴蝶 客户端应用** | 以独立进程或者嵌入式的方式运行与终端或者第三方客户端进程中的程序。负责和蝴蝶服务器进行通讯。并将蝴蝶推送过来的通知最终发送给目标应用。 |
| **Notification(通知)** | 由第三方服务器发送，最终将会到达第三方客户端应用的数据。数据的格式不限（二进制）。体积较小。 |
| **Client ID** | 第三方应用实例标识。由蝴蝶服务生成。用于标识某个接收通知的端点。蝴蝶系统会跟据【Client ID】和【Application ID】来确定最终接收通知的应用实例。  第三方应用需要按保存【Client ID】。 |
| **Application ID** | 第三方应用的编号。蝴蝶会根据这个ID来确认第三方服务器的身份和权限。第三方应用服务在与蝴蝶服务进行交互时，需要提供这个ID。  【Application ID】由蝴蝶签发。通过硬编码或者配置文件固化与第三方应用服务器中。 |

# 一般推送流程

整个流程分为如下的场景：

1. 注册Push服务
2. 通知推送
3. 通知接收

## 绑定Push服务

终端上的程序需要首先向蝴蝶进行绑定后才能接到推送的通知。

第三方客户端需要提交第三方应用的【Application ID】给蝴蝶客户端来请求Push服务。蝴蝶客户端会生成并维护【Client ID】与第三方客户端的对应关系。并将【Client ID】发送给第三方客户端。

第三方客户端需要将新的【Client ID】存储并且发送给第三方应用服务器。

绑定成功后，蝴蝶系统会一直维护【Application ID】、【Client ID】与当前设备【Device ID】的对应关系。直到被第三方客户端显式的解除绑定。

如果不是第一次绑定，第三方客户端需要将以前的【Client ID】和【Application ID】发送给蝴蝶客户端。验证通过后，蝴蝶客户端会更新绑定关系并且下发新的【Client ID】给第三方客户端。

注意：如果第三方应用绑定了太多的【Client ID】后可能会引起402错误。

建议应用在每次应用启动、打开主界面时都进行一次绑定。

## 通知推送

通知的推送由第三方的应用服务器发起。整个流程如下：

1. 第三方应用服务器发送通知到蝴蝶服务器。同时附带接收通知的【Client ID】、第三方应用的【Application ID】以及【Application Key】。
2. 蝴蝶服务器进行必要的校验、鉴权。
3. 当第三方客户端中蝴蝶客户端在线的时候， 蝴蝶服务会通过【Client ID】直接将通知推送给客户端。否则， 消息会被暂时存储。

## 通知接收

蝴蝶客户端收到通知后， 会根据【Application ID】和【Client ID】找到本地对应的应用。最终将通知的内容以函数回调、进程间通讯、文件共享等方式传递给第三方客户端程序。传递方式取决于各个平台的实现。

# 约束

* 由于蝴蝶的特性， 一般通知应该是比较小的数据。目前限定为不大于2KB的二进制数据。较大数据可以由第三方客户端收到通知后， 再通过独立的协议与第三方服务器进行交互获取。

# 如何使用蝴蝶推送通知

第三方服务器可以通过HTTP协议， 向蝴蝶提交要推送的数据。第三方服务器需要遵循如下规则来提交数据：

* 使用POST发送数据
* 通知内容作为Request的Body
* Content-Type头的值为“application/octet-stream”
* 通知长度不超过2K
* 系统采用HTTP-BASIC的身份验证方式，客户端需要将【Application ID】作为用户名，【Application Key】作为密码。
* 第三方服务器可以通过名为“exptime”的Header来控制消息的有效期。Value是消息的有效期，以秒为单位。消息的最大有效期为24小时。
* 点对点推送， 需要将数据POST至：http://push-api-0.hudee.com/service/client/【Client ID】/message。
* URL中的【Client ID】需要按“UTF-8”进行URL编码
* 如果处理成功（Http Response的状态码为200），Http Response的Body内容为消息的唯一标识
* 如果处理失败， Http Response的Body为具体的错误信息

蝴蝶对请求处理后，会返回相应的HTTP状态码来标识处理的结果。回应的body为具体的纯文本信息。可能的回应如下：

|  |  |  |
| --- | --- | --- |
| Http状态码 | 说明 | 回复文本 |
| 200 | 处理成功 | 消息的唯一标识 |
| 404 | 【Client ID】不存在 | ERROR\_CLIENT\_NOT\_FOUND |
| 401 | 【Application ID】无效。可能是【Application ID】停用，Key无效等原因 | ERROR\_INVALID\_ID\_AND\_KEY |
| 400 | 推送消息为【inbox】类型或【X-Target】为“IOS”时，【Content-type】必须为“application/json”类型 | CONTENT\_MUST\_BE\_JSON |
| 413 | 普通推送通知内容超过了2K或IOS推送超过了APNS的最大长度。 | ERROR\_MESSAGE\_LENGTＨ＿EXCEED |
| 500 | 服务器内部错误 | ERROR\_INTERNAL |

# 如何使用查询通知的递送状态

蝴蝶服务器提供了对消息递送状态的查询服务：

* 第三方应用通过Http Get查询指定【Message ID】的递送状态
* 该服务位于：<http://push-api-0.hudee.com/service/client/all/message/【Message>ID】
* 服务器正常返回的Content-Type为“application/json”, 结果中包含一个属性为“status”。可能的取值有：
  + DELIVERING， 系统正在尝试对通知进行投递
  + ACKED， 客户端已经明确报告消息递送成功
  + EXPIRED，消息在有效期内没有被成功投递。系统放弃继续对这个消息进行投递
  + EXPIRED\_ACKED，消息过期后，蝴蝶服务器收到了客户端的递送成功报告

蝴蝶对请求处理后，会返回相应的HTTP状态码来标识处理的结果。如果出错，回应的body为具体的纯文本信息。可能的回应如下：

|  |  |  |
| --- | --- | --- |
| Http状态码 | 说明 | 回复文本 |
| 200 | 处理成功 | 消息的唯一标识 |
| 404 | 【Message ID】不存在 | ERROR\_MESSAGE\_NOT\_FOUND |
| 401 | 【Application ID】无效。可能是【Application ID】停用，Key无效等原因 | ERROR\_INVALID\_ID\_AND\_KEY |
| 500 | 服务器内部错误 | ERROR\_INTERNAL |

# 如何使用蝴蝶向IOS设备推送通知

目前蝴蝶的REST接口可以向IOS设备推送基本的通知。蝴蝶对于这些通知不做暂存、重发以及消息递送状态报告。第三方服务器通过推送的Http接口像蝴蝶提交数据：

* 使用POST发送数据
* 通知内容作为Request的Body
* Content-Type头的值为“application/json”
* 通知长度不超过256B
* 内容为Apple Push Notification的标准JSON格式。具体内容请参照《Local and Push Notification Programming Guide》中的“[The Notification Payload](http://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/ApplePushService/ApplePushService.html#//apple_ref/doc/uid/TP40008194-CH100-SW1)”
* 系统采用HTTP-BASIC的身份验证方式，客户端需要将【Application ID】作为用户名，【Application Key】作为密码。
* 需要将数据POST至：<http://push-api-0.hudee.com/service/client/【Device> Token】/message。

**使用IOS服务前， 请向蝴蝶的客服确认已经开通了此项服务。**

# 如何创建使用蝴蝶服务的第三方客户端应用程序

## Android

创建使用蝴蝶服务的客户端应用程序，开发者主要有以下几项工作：

注：下面以Android平台为例

* 创建包含应用使用蝴蝶服务许可的配置文件
* 编写应用注册和注销的服务
* 处理接收到的数据

### 创建配置文件

所有应用根目录下都有一个AndroidManifest.xml文件，要想使用蝴蝶，清单文件需要包含下列信息:

* com.hudee.pns.permission.RECEIVE：应用有注册和接收蝴蝶消息的权限
* android.permission.INTERNET：应用需要访问网络的权限.
* 申明 applicationPackage + ".permission.HUDEE\_MESSAGE 权限，用来防止别的应用窃听蝴蝶消息.
* 接收 com.hudee.pns.intent.RECEIVE 和 com.hudee.pns.intent.REGISTRATION 这两个Intent的Broadcast Receiver, 且Intent的category 需要设置为应用的包名. 这个Broadcast receiver 应该要求com.hudee.pns.permission.SEND 权限许可,从而保证不会收到其他恶意应用发送的假消息.

下边是一个AndroidManifest.xml的样例：

<manifest package="com.example.myapp" ...>

<!-- Only this application can receive the messages and registration result -->

<permissionandroid:name="com.example.myapp.**permission.HUDEE\_MESSAGE**" android:protectionLevel="signature" />

<uses-permission android:name="com.example.myapp.**permission.HUDEE\_MESSAGE**" />

 <!-- This app has permission to register and receive message -->

<uses-permission android:name="com.hudee.pns**.permission.RECEIVE**" />

 <!-- Send the registration id to the server -->

<uses-permission android:name="android.permission.INTERNET" />

<!-- Only 蝴蝶 servers can send messages for the app. If permission is not set - any other app can generate it -->

<receiverandroid:name=".HudeeReceiver" android:permission="com.hudee.pns.permission.SEND">

    <!-- Receive the actual message -->

    <intent-filter>

        <actionandroid:name="com.hudee.pns.intent.MESSAGE" />

        <categoryandroid:name="com.example.myapp" />

    </intent-filter>

    <!-- Receive the registration id -->

    <intent-filter>

        <actionandroid:name="com.hudee.pns.intent.REGISTRATION" />

        <categoryandroid:name="com.example.myapp" />

    </intent-filter>

</receiver>

### 应用向蝴蝶系统注册服务

应用需要发送Intent(com.hudee.pns.intent.REGISTER) 注册服务.必须带有参数“appId”。”appId”是由已经获得蝴蝶服务授权的第三方应用标识。

如果不是第一次绑定，应用还需发送“clientId”参数。将之前申请的【Client Id】提交给蝴蝶服务。

建议应用在每次启动、打开主界面时都进行一次绑定。

示例：

Intent registrationIntent = new Intent("com.hudee.pns.intent.REGISTER");

registrationIntent.putExtra("app", PendingIntent.getBroadcast(this, 0, new Intent(), 0)); //

registrationIntent.putExtra("appId",”a457883283ef6”);

registrationIntent.putExtra(“clientId",”user001”);

startService(registrationIntent);

### 应用向 蝴蝶系统注销服务

建议接入方客户端在用户注销登录或在该应用中停止push时进行一下注销服务操作。

**示例:**

Intent unregIntent = new Intent("com.hudee.pns.intent.UNREGISTER");

unregIntent.putExtra("app", PendingIntent.getBroadcast(this, 0, new Intent(), 0));

unregIntent.putExtra("appId",”a457883283ef6”);

unregIntent.putExtra("appId",”user001”);

startService(unregIntent);

### 应用处理注册结果

在创建清单文件中描述的，清单中定义了Intent(com.hudee.pns.intent.REGISTRATION)的广播接收者。也定义了Intent(com.hudee.pns.intent.MESSAGE)的广播接收者。

注册操作和消息都实现为Intent。

如果注册不能正常完成，REGISTRATION Intent会产生一个error参数。

REGISTRATION Intent可能的错误码：

|  |  |
| --- | --- |
| 错误码 | 说明 |
| 200 | 处理成功 |
| 401 | 【Application ID】无效 |
| 402 | 【Application ID】绑定了太多了的【Client ID】 |
| 403 | 【Application Key】或者【Device Key】无效 |
| 404 | 【Client ID】与【Application ID】不存在绑定关系 |
| 500/503 | 服务器内部错误， 暂时无法提供服务 |

应用注册成功，向第三方服务器提交【Client Id】.应用收到REGISTRATION Intent.

例如:

public void onReceive(Context context, Intent intent) {

    if (intent.getAction().equals("com.hudee.pns.intent.REGISTRATION")) {

        handleRegistration(context, intent);

    }

 }

private void handleRegistration(Context context, Intent intent) {

    if (intent.getStringExtra("error") != null) {

        // Registration failed, should try again later.

    } else if (intent.getStringExtra("unregistered") != null) {

        // unregistration done, new messages from the authorized sender will be rejected

    } else if (registration != null) {

       // Send the client ID to the 3rd party site that is sending the messages.

       // This should be done in a separate thread.

    }

}

### 应用处理接收的数据

当蝴蝶客户端收到服务器发送的消息时，它会提取消息的有效负载并将数据传给应用的Intent(com.hudee.pns.intent.MESSAGE).应用根据自身需要解析数据。

protected void onReceive(Context context, Intent intent) {

    String appId = intent.getExtras().getString(Config.HUDEE\_APP\_ID\_EXTRA);

    byte[] message = intent.getByteArrayExtras(Config.HUDEE\_MESSAGE\_EXTRA);

if (message != null) {

//handle received message...

       if (Log.isLoggable(TAG, Log.DEBUG)) {

           Log.d(TAG, "Messaging received: " + new String(message));

       }

    }

}

## BlackBerry

### 检查是否已经安装中间件

在编码过程中，请将HUDEE\_BB\_CORE以及HUDEE\_BB\_CORE.cod保存在src或者您的代码根目录下。

示例代码如下:

**privateboolean**isInstallCore() {

**int** handle = CodeModuleManager.*getModuleHandle*("HuDee\_Core");

**if** ( handle != 0 ){

**returntrue**;

}

**returnfalse**;

}

若检测到平台上未安装中间件则将HUDEE\_BB\_CORE以及HUDEE\_BB\_CORE.cod读取到设备内存中，并运行

**if** (isInstallCore()) {

**int** result = Dialog.*ask*(Dialog.*D\_YES\_NO*,"插件已经安装,否是覆盖");

**if** (result == Dialog.*YES*){

installCod();

}

**} else {**

**Dialog.inform("确认后会为您安装插件");**

**installCod();**

**}**

**privatevoidinstallCod() {**

**Class classs;**

**try {**

**classs = Class.forName(this.getClass().getName());**

**InputStreamcod\_is = classs.getResourceAsStream("/PNS\_BB\_Core.cod");**

**InputStreamjad\_is = classs.getResourceAsStream("/PNS\_BB\_Core");**

**booleancopyCore = false;**

**copyCore = writeCoreFile(cod\_is, "PNS\_BB\_Core.cod");**

**copyCore = writeCoreFile(jad\_is, "PNS\_BB\_Core.jad");**

**if (copyCore){**

**BrowserSessionbrowserSession = Browser.getDefaultSession();**

**browserSession.displayPage("file:///store/home/user/PNS\_BB\_Core.jad");**

**}**

**} catch (ClassNotFoundException e) {**

**e.printStackTrace();**

**}**

**}**

**privatebooleanwriteCoreFile(InputStreaminputStream, String fileName){**

**try {**

**FileConnection conn = (FileConnection) Connector.open("file:///store/home/user/"+ fileName, Connector.READ\_WRITE);**

**if (conn.exists()){**

**conn.delete();**

**}**

**conn.create();**

**byte[] bytes = IOUtilities.streamToBytes(inputStream);**

**OutputStream output = conn.openOutputStream();**

**output.write(bytes);**

**output.flush();**

**returntrue;**

**} catch (IOException e) {**

**returnfalse;**

**}**

**}**

至此中间件检测，安装完毕。

### 中间件常量定义

请定义以下常量：

//广播ID

**publicfinalstaticlong***REQUEST\_PUSH\_SERVICE* = 0xa418a364c40d919eL;

//第三方绑定时用的GUID

**publicfinalstaticint***CMD\_BIND* = 200;

**publicfinalstaticint***CMD\_CANCEL\_BIND* = 201;

**publicfinalstaticint***CMD\_SEND\_MESSAGE* = 202;

//Response的取值对应的INT值

**publicfinalstaticint***INT\_RESPONSE\_SUCCESS* = 200;

**publicfinalstaticint***INT\_RESPONSE\_ERROR* = 500;

**publicfinalstaticint***INT\_RESPONSE\_ERROR\_INVALID\_CREDENTIAL* = 501;

**publicfinalstaticint***INT\_RESPONSE\_ERROR\_INVALID\_APPLICATION\_ID* = 502;

**publicfinalstaticint***INT\_RESPONSE\_ERROR\_TOO\_MANY\_CLIENT\_ID* = 503;

### 绑定PUSH服务

客户端POST广播至REQUEST\_PUSH\_SERVICE，消息格式为

long Guid为REQUEST\_PUSH\_SERVICE

int data0 为 CMD\_BIND

int data1 为 0

Object object0 为使用vector，内容按顺序包含【Application ID】以及【Client ID】的String值.如果没有【Client ID】则不用包含

Object object1 为第三方程序自定义的guid long值的String格式。

若成功绑定，中间件会向object1所定义的guid发送广播消息，返回内容格式如下

Data0为CMD\_BIND

Data1 为服务器所返回的状态，具体请看Response的取值对应的INT值。

Object0 若服务器成功绑定则返回【Client ID】。应用可以将此【Client ID】传递给第三方应用服务器，以便第三方应用服务器推送消息。

### 解除PUSH绑定

同绑定服务，除了data0需要设定为CMD\_CANCEL\_BIND

建议接入方客户端在用户注销登录或在该应用中停止push时进行一下注销服务操作。

### 接收PUSH消息

若中间件收到新的PUSH消息，则会向第三方广播。

其中data0为CMD\_SEND\_MESSAGE

Object0 为经过base64编码后的消息。第三方客户端根据具体解析方式进行解析

## Symbian S60平台

### 为SDK安装加密库补丁

下载地址：<http://wiki.forum.nokia.com/index.php/File:SymbianCryptographyLibraries.zip>

S60 3Rd和5th都可使用本文件

注：因为安装默认安装目录错误的原因，安装后无法正常运行，应手工将X:\Symbian\epoc32复制到X:\Symbian\9.1\S60\_3rd\_MR\（X:\Symbian\5th\S60\_5th\_Edition\_SDK\_v1.0）下，与原有目录合并。

### 导入静态库

在工程mmp文件中导入静态库HDTunnel.lib

静态库的API接口

**Class**CHDTunnel:**public** CBase

{

**public**:

  // 第一次使用时,aClientId不用传递

**static**CHDTunnel\***NewL**(MHDNotificationObserver&aObserver,**const**TDesC&aApplicationId, **const**TDesC&aClientId=**\_L**("")) = 0;

  //注册Push服务

**voidRegisterL**();

  // 开始监听消息

**voidListenL**();

  // 发送消息，最大为1K

**voidSendL**(**const**TDesC&aMessage);

  // 设置唤醒参数

**voidSetWakeupParamsL**(**const**TDesC&aProcessParameters = **\_L**(""));

};

### 实现MHDNotificationObserver接口

**Class**MHDNotificationObServer

{

**public**:

  // 发生错误

**virtualvoidHandle\_HDError**(TIntaError) = 0;

  // 收到Push信息

**virtualvoidHandle\_HDNotification**(**const**TDesC8&aMessage) = 0;

  // 注册Push服务失败

**virtualvoidHandle\_HDRegisterFailed**(TIntaError) = 0;

  // 注册Push服务成功

**virtualvoidHandle\_HDRegisterSuccessed**(**const**TDesC&aClientId) = 0;

  // 取消注册结束

**virtualvoidHandle\_HDUnregisterCompleted**(TIntaError) = 0;

  // 消息已发送

**virtualvoidHandle\_HDMessageSent**(TIntaError) = 0;

};

### 注册Hudee服务

首次使用时，通过调用静态库HDTunnel.lib中的RegisterL()进行注册，当注册成功时，在Handle\_HDRegisterSuccessed(constTDesC&aClientId)中会收到一个【Client ID】，第三方应用需要保存此【Client ID】，下次启动服务时直接使用此【Client ID】，不用再次注册。应用可以将此【Client ID】传递给第三方应用服务器，以便第三方应用服务器推送消息。

示例：

RFsfs= iCoeEnv->**FsSession**();

RFile file;

\_**LIT**(KFileName,"c:\\ClientId.txt");

TInt err = file.**Open**(fs,KFileName,EFileRead);

**if**(err == KErrNotFound)

err = file.**Create**(fs,KFileName,EFileWrite);

TIntfileLen = 0;

file.Size(fileLen);

CHDTunnel\* iHDTunnel;

if(fileLen<=0)

{

//首次使用，通过注册获取ClientID；

iHDTunnel= CHDTunnel::NewL(\*this,**\_L**("Your App Id"),**\_L**(""));

iHDTunnel->RegisterL();

}

else

{

HBufC8\* clientId8 = HBufC8::New(fileLen);

TPtr8ptr8(clientId8->**Des**());

file.**Read**(ptr8);

HBufC16\* clientId16 = HBufC16::New(2\*fileLen);

clientId16->**Des**().Copy(clientId8->**Des**());

//拥有ClientID后，就不用再注册了，可以直接使用；

iHDTunnel = CHDTunnel::NewL(\*this,**\_L**("Your App Id"),clientId16->**Des**());

**delete** clientId8;

**delete** clientId16;

}

file.**Close**();

### 处理Push消息

通过调用静态库HDTunnel.lib中的ListenL()接口就可以监听PushL来的消息，当Hudee收到Push消息后，会通过Handle\_HDNotification(constTDesC8&aMessage)函数将Push消息传送给第三方应用，第三方应用程序只需要实现此接口对消息进行处理即可。

### 发送消息

通过调用静态库HDTunnel.lib中的SendL(constTDesC&aMessage)接口发送消息，最大发送为1K。当发送消息成功时会调用Handle\_HDMessageSent(TIntaError)，发送消息失败时会调用Handle\_HDError(TIntaError)。

### 设置唤醒参数

通过调用静态库HDTunnel.lib中的SetWakeupParamsL(constTDesC&aProcessParameters = \_L(""))接口在Hudee中设置唤醒参数。当有Push消息但第三方应用程序没有启动时，Hudee可以通过此参数来启动第三方应用程序。

### 中间件发布

将蝶蝴中间件安装包（hudeeserver.sisx）做为应用安装包的一个文件，保存至手机数据目录，例如在shangmail.pkg中填加

"..\hudee\hudeeserver\_shangmail.sisx" -"!:\data\ hudeeserver\_ shangmail.sisx "

应用第一次启动时，调用API安装蝴蝶中间件

CHDPublic::InstallHudeeL(\_L("\\data\\hudeeserver\_shangmail.sisx"));

### 中间件与应用集成发布

以shangmail为例：

1. 联系蝴蝶商务，得到定制的hudee中间件相关文件，包括README\_shangmail.TXT、hudeeserver\_shangmail.exe，并复制到与sis目录平级的hudee目录中
2. 设置蝴蝶中间件可以开机自启动，添加自启动项目到YOURUID.rss文件中，如：

文件 20027CF4.rss：

RESOURCE STARTUP\_ITEM\_INFO StartHudeeServerShangmail

{

executable\_name = "!:\\sys\\bin\\hudeeserver\_shangmail.exe";

recovery = EStartupItemExPolicyNone;

}

1. 在打包文件中添加蝴蝶中间件支持文件，如：

文件shangmail.pkg中添加：

"..\hudee\README\_shangmail.TXT" -"!:\data\hudee\README\_shangmail.TXT"

"$(EPOCROOT)Epoc32\Data\z\resource\apps\20027CF4.rsc" -"c:\private\101f875a\import\[20027CF4].rsc"

..\hudee\hudeeserver\_shangmail.exe" -"!:\sys\bin\hudeeserver\_shangmail.exe",FR,RI