信鸽服务端 Java SDK V1.1.8 用户手册

1 整体介绍

本 SDK 提供信鸽服务端接口的 Java 封装，与信鸽后台通信。使用时引用 XingeApp 包即可，高级接口可参考 demo.java 来学习使用方法。

2 快捷方式

本章节描述如何使用**一行代码**完成推送操作，可参考 demo.java。
注：快捷方式只支持推送通知，不支持透传消息。快捷方式不支持定时推送。

基本概念：

名称	类型	描述
accessId	long	推送目标应用的 ID
secretKey	String	目标应用的密钥
title	String	消息标题，**Android 专属**
content	String	消息内容
token	String	接收消息的设备 Token
account	String	接收消息的账号
tag	String	接收消息的设备标签
environment	int	可选值为 XingeApp.IOSENV_PROD 或者 XingeApp.IOSENV_DEV，**IOS 专属**

注：secretKey 是后台接口鉴权的密钥，accessKey 为客户端鉴权密钥。调用后台 sdk 时需要使用 secretKey。

2.1 Android 平台推送消息给单个设备

```
public static JSONObject pushTokenAndroid(long accessId,String secretKey,String title,String
content,String token)
```

示例：

```
XingeApp.pushTokenAndroid(000, "myKey", "标题", "大家好!", "3dc4gcd98sdc");
```

返回值：

```
{"ret_code":0}                                                    //成功
{"ret_code":-1, "err_msg":"error description"}                    //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

2.2 Android 平台推送消息给单个账号

```
public static JSONObject pushAccountAndroid(long accessId,String secretKey,String title,String content,String account)
```

示例：

```
XingeApp.pushAccountAndroid(000, "myKey", "标题", "大家好!", "nickName");
```

返回值：

```
{"ret_code":0}                                                    //成功
{"ret_code":-1, "err_msg":"error description"}                    //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

2.3 Android 平台推送消息给所有设备

```
public static JSONObject pushAllAndroid(long accessId,String secretKey,String title,String content)
```

示例：

```
XingeApp.pushAllAndroid(000, "myKey", "标题", "大家好!");
```

返回值：

```
//返回 push_id，即推送的任务 ID
{"ret_code":0, "result":{"push_id":"11121"}}                      //成功
{"ret_code":-1, "err_msg":"error description"}                    //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

2.4 Android 平台推送消息给标签选中设备

```
public static JSONObject pushTagAndroid(long accessId,String secretKey,String title,String content,String tag)
```

示例：

```
XingeApp.pushTagAndroid(000, "myKey", "标题", "大家好!", "beijing");
```

返回值：

```
//返回 push_id，即推送的任务 ID
{"ret_code":0, "result":{"push_id":"11121"}}                      //成功
{"ret_code":-1, "err_msg":"error description"}                    //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

2.5 IOS 平台推送消息给单个设备

```
public static JSONObject pushTokenIos(long accessId,String secretKey,String content,String
```

token,int environment)

示例：

```
XingeApp.pushTokenIos(000, "myKey", "你好!", "3dc4gcd98sdc", XingeApp.IOSENV_PROD);
```

返回值：

```
{"ret_code":0}                                                    //成功
{"ret_code":-1, "err_msg":"error description"}                    //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

2.6 IOS 平台推送消息给单个账号

```
public static JSONObject PushAccountIos(long accessId,String secretKey,String content,String
account,int environment)
```

示例：

```
XingeApp.pushAccountIos(000, "myKey", "你好", "nickName", XingeApp.IOSENV_PROD);
```

返回值：

```
{"ret_code":0}                                                    //成功
{"ret_code":-1, "err_msg":"error description"}                    //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

2.7 IOS 平台推送消息给所有设备

```
public static JSONObject pushAllIos(long accessId,String secretKey,String content,int
environment)
```

示例：

```
XingeApp.pushAllIos(000, "myKey", "大家好!", XingeApp.IOSENV_PROD);
```

返回值：

```
//返回 push_id，即推送的任务 ID
{"ret_code":0, "result":{"push_id":"11121"}}                      //成功
{"ret_code":-1, "err_msg":"error description"}                    //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

2.8 IOS 平台推送消息给标签选中设备

```
public static JSONObject pushTagIos(long accessId,String secretKey,String content,String tag,int
environment)
```

示例：

```
XingeApp.pushTagIos(000, "myKey", "大家好!", "beijing", XingeApp.IOSENV_PROD);
```

返回值：

```
//返回 push_id，即推送的任务 ID
```

```
{"ret_code":0, "result":{"push_id":"11121"}}                    //成功
{"ret_code":-1, "err_msg":"error description"}                  //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3 高级接口

本章节描述如何使用带有全部参数的推送接口。

3.1 数据结构介绍

3.1.1 TimeInterval

表示一个允许推送的时间闭区间，从 startHour : startMin 到 endHour : endMin

字段名	类型	说明
startHour	int	起始小时，取值范围 [0,23]
startMin	int	起始分钟，取值范围 [0,59]
endHour	int	截止小时，取值范围 [0,23]
endMin	int	截止分钟，取值范围 [0,59]

Example:

```
TimeInterval acceptTime = new TimeInterval(0, 0, 23, 59);
```

3.1.2 ClickAction

通知消息被点击时触发的事件

字段名	类型	说明
actionType	int	TYPE_ACTIVITY 打开 activity 或 app 本身 TYPE_URL 打开指定的 url TYPE_INTENT 打开指定 Intent 默认为 TYPE_ACTIVITY 使用 setActionType(int actionType)设置
url	String	要打开的 url， actionType 为 TYPE_URL 时必填。 使用 setUrl(String url)设置
confirmOnUrl	int	打开 url 时是否需要用户确认，1 需要，0 不需要，默认为 0。 actionType 为 TYPE_URL 时生效 使用 setComfirmOnUrl(int confirmOnUrl)设置
activity	String	打开指定的 activity，不填则运行 app。 actionType 为 TYPE_ACTIVITY 时生效 使用 setActivity(String activity)设置
atyAttrIntentFlag	int	activity 属性，只针对 actionType=TYPE_ACTIVITY 的情况。

		创建通知时，intent 的属性，如： intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_RESET_TASK_IF_NEEDED) 使用 setAtyAttrIntentFlag(int atyAttrIntentFlag)设置
atyAttrPendingIntentFlag	int	activity 属性，只针对 actionType=TYPE_ACTIVITY 的情况。 PendingIntent 的属性，如： PendingIntent.FLAG_UPDATE_CURRENT 使用 setAtyAttrPendingIntentFlag(int atyAttrPendingIntentFlag)设置
intent	String	打开指定 intent。 actionType 为 TYPE_INTENT 时生效。 请在 Android 上使用 intent.toUri(Intent.URI_INTENT_SCHEME)方法来得到序列化后的 intent 字符串填在此处，自定义 intent 参数也会包含在其中 使用 setIntent(String intent)设置

Example:

```
ClickAction action = new ClickAction();
action.setActionType(ClickAction.TYPE_URL);
action.setUrl("http://xg.qq.com");
```

3.1.3 Style

定义通知消息如何展现

字段名	类型	说明
builderId	int	本地通知样式，必填。含义参见终端 SDK 文档 构造函数中指定
ring	int	是否响铃，0 否，1 是。选填，默认 0 构造函数中指定
vibrate	int	是否振动，0 否，1 是。选填，默认 0 构造函数中指定
clearable	int	通知栏是否可清除，0 否，1 是。选填，默认 1 构造函数中指定
nId	int	若大于 0，则会覆盖先前弹出的相同 id 通知；若为 0，展示本条通知且不影响其他通知；若为-1，将清除先前弹出的所有通知，仅展示本条通知。选填，默认为 0 构造函数中指定
lights	int	是否呼吸灯，0 否，1 是，选填，默认 1 构造函数中指定
iconType	int	指定通知栏图标是使用应用内图标还是使用自己上传的图标。 0 是应用内图标，1 是上传图标，选填。默认 0 构造函数中指定

iconRes	String	应用内图标文件名（xg.png）或者下载图标的 url 地址，选填 使用 `setIconRes(String iconRes)`设置
ringRaw	String	指定应用内的声音（ring.mp3），选填 使用 `setRingRaw(String ringRaw)`设置
styleId	int	Web 端设置是否覆盖编号的通知样式，0 否，1 是，选填。默认 1 构造函数中指定
smallIcon	String	"xg"指定状态栏的小图片(xg.png),选填 使用 `setSmallIcon(String smallIcon)`设置

Example:

```
    //依次为(int builderId, int ring, int vibrate, int clearable, int nId, int lights, int iconType,
int styleId)
    Style style = new Style(0,0,0,1,0,1,0,1);
    style.setIconRes("xg.png");
```

3.1.4 Message

定义推送消息(Android 平台)

字段名	类型	说明
title	String	标题，标题+内容不得超过 800 英文字符或 400 非英文字符 使用 `setTitle(String title)`设置
content	String	内容，标题+内容不得超过 800 英文字符或 400 非英文字符 使用 `setContent(String content)`设置
expireTime	int	消息离线存储多久，单位为秒，最长存储时间 3 天。选填，默认为 3 天，设置为 0 等同于使用默认值 使用 `setExpireTime(int expireTime)`设置
sendTime	String	消息定时推送的时间，格式为 year-mon-day hour:min:sec，若小于服务器当前时间则立即推送。选填，默认为空字符串，代表立即推送 使用 `setSendTime(String sendTime)`设置
acceptTime	List	元素为 TimeInterval 类型，表示允许推送的时间段，选填 使用 `addAcceptTime(TimeInterval acceptTime)`来添加元素
type	int	消息类型。 `TYPE_NOTIFICATION`:通知； `TYPE_MESSAGE`:透传消息。必填 注意：`TYPE_MESSAGE` 类型消息 默认在终端是不展示的，不会弹出通知 使用 `setType(int type)`设置
multiPkg	int	0 表示按注册时提供的包名分发消息；1 表示按 access id 分发消息，所有以该 access id 成功注册推送的 app 均可收到消息。选填，默认为 0 使用 `setMultiPkg(int multiPkg)`设置
style	Style	通知样式，透传消息可不填 使用 `setStyle(Style style)`设置
action	ClickAction	通知被点击的动作，默认为打开 app。透传消息可不填 使用 `setAction(ClickAction action)`设置

custom	Map	选填。用户自定义的 key-value，key 必须为 string 注意：用于用户透传消息，此项内容将记入消息字符数。 使用 `setCustom(Map<String, Object> custom)`设置
loopInterval	int	选填。循环任务的执行间隔，以天为单位，取值[1, 14]。 但是循环任务第一次执行与最后一次执行的间隔不超过 14 天。 如：loopInterval=3　loopTimes=6 即为非法请求 此参数仅对 pushAllDevice 和 pushTags 接口有效。 使用 `setLoopInterval(int loopInterval)`设置
loopTimes	int	选填。循环任务的执行次数，取值[1, 15]。 但是循环任务第一次执行与最后一次执行的间隔不超过 14 天。 如：loopInterval=3　loopTimes=6 即为非法请求 此参数仅对 pushAllDevice 和 pushTags 接口有效。 使用 `setLoopTimes(int loopTimes)`设置

Example:

```
Message mess = new Message();
mess.setType(Message.TYPE_NOTIFICATION);
mess.setTitle("title");
mess.setContent("中午");
mess.setExpireTime(86400);

#含义：样式编号0，响铃，震动，不可从通知栏清除，不影响先前通知
Mess.setStyle(new Style(0,1,1,0,0));

ClickAction action = new ClickAction();
action.setActionType(ClickAction.TYPE_URL);
action.setUrl("http://xg.qq.com");
action.setComfirmOnUrl(1);
mess.setAction(action);

Map<String,Object> custom = new HashMap<String,Object>;
custom.put("key","value")
mess.setCustom(custom);

TimeInterval acceptTime1 = new TimeInterval(12, 0, 13, 59);
TimeInterval acceptTime2 = new TimeInterval(19, 0, 20, 59);
mess.addAcceptTime(acceptTime1);
mess.addAcceptTime(acceptTime2);
```

3.1.5 MessageIOS

定义 iOS 平台推送消息

字段名	类型	说明

expireTime	int	消息离线存储多久，单位为秒，最长存储时间 3 天。选填，默认为 3 天，设置为 0 等同于使用默认值 使用 `setExpireTime(int expireTime)`设置
sendTime	String	消息定时推送时间，格式为 year-mon-day hour:min:sec，若小于服务器当前时间则立即推送。选填，默认为空字符串，代表立即推送 使用 `setSendTime(String sendTime)`设置
acceptTime	List	元素为 TimeInterval，表示允许推送的时间段，选填 使用 `addAcceptTime(TimeInterval acceptTime)`添加元素
alert	String	定义详见 APNS payload 中的 alert 字段 使用 `setAlert(String alert)`设置
badge	int	设置角标数值。定义详见 APNS payload 中的 badge 字段 使用 `setBadge(int badge)`设置
sound	String	设置通知声音。定义详见 APNS payload 中的 sound 字段 使用 `setSound(String sound)`设置
custom	Map	选填。用户自定义的 key-value。注意每一个 key 和 value 的字符都计入消息字符数 使用 `setCustom(Map<String, Object> custom)`设置
category	String	iOS 8新增。定义详见 APNS payload 中的 category 字段 使用 `setCategory(String category)`设置
raw	String	选填。用户自定义 APNS 通知内容，格式为 json 字典对象转换的字符串。定义详见 APNS payload 使用 `setRaw(String raw)`设置 设置该项后，只有 expireTime、sendTime 设置项生效，其余设置项均失效
loopInterval	int	选填。循环任务的执行间隔，以天为单位，取值[1, 14]。 但是循环任务第一次执行与最后一次执行的间隔不超过 14 天。 如：loopInterval=3 loopTimes=6 即为非法请求 此参数仅对 pushAllDevice 和 pushTags 接口有效。 使用 `setLoopInterval(int loopInterval)`设置
loopTimes	int	选填。循环任务的执行次数，取值[1, 15]。 但是循环任务第一次执行与最后一次执行的间隔不超过 14 天。 如：loopInterval=3 loopTimes=6 即为非法请求 此参数仅对 pushAllDevice 和 pushTags 接口有效。 使用 `setLoopTimes(int loopTimes)`设置

Example:

```
MessageIOS mess = new MessageIOS();
mess.setExpireTime(86400);
mess.setAlert("ios test");
mess.setBadge(1);
mess.setSound("beep.wav");
Map<String,Object> custom = new HashMap<String,Object>;
custom.put("key","value")
mess.setCustom(custom);
TimeInterval acceptTime = new TimeInterval(0, 0, 23, 59);
mess.addAcceptTime(acceptTime);

JSONObject obj = new JSONObject();
JSONObject aps = new JSONObject();
aps.put("sound", "beep.wav");
aps.put("alert", "ios test");
aps.put("badge", 1);
aps.put("content-available", 1);
obj.put("aps", aps);
mess.setRaw(obj.toString());
```

3.1.6 TagTokenPair

定义一个标签-token 对

字段名	类型	说明
tag	String	标签，长度最多 50 字节，不可包含空格
token	String	设备 token。注意长度，Android 是 40 或 64 字节，iOS 是 64 字节

Example:

```
TagTokenPair pair = new TagTokenPair("tag1", "token00000000000000000000000000000001");
```

3.1.7 XingeApp

该类提供与信鸽后台交互的接口。构造函数有两个参数，均为必选。

参数名	类型	必需	默认值	参数描述
accessId	int	是	无	推送目标应用 id
secretKey	String	是	无	推送密钥

注：secretKey 是后台接口鉴权的密钥，accessKey 为客户端鉴权密钥。调用后台 sdk 时需要使用 secretKey。

Example:

```
XingeApp push = new XingeApp(000, "myKey");
```

3.2 XingeApp 高级接口

3.2.1 pushSingleDevice 推送消息给单个设备

```
public JSONObject pushSingleDevice(String deviceToken,Message message) //android 使用
public JSONObject pushSingleDevice(String deviceToken,MessageIOS message,int environment) //IOS 使用
```

参数说明：

参数名	类型	必需	默认值	参数描述
deviceToken	string	是	无	推送目标设备 token
message	Message	是	无	消息体，IOS 推送使用 MessageIOS 类型
environment	int	**iOS 专用**	0	向 iOS 设备推送时必填，IOSENV_PROD 表示推送生产环境；IOSENV_DEV 表示推送开发环境。 推送 Android 平台不填或填 0

Example：

```
XingeApp push = new XingeApp(000, 'myKey');
Message mess = new Message(); //$mess = new MessageIOS();
//完善 Message 消息
...
JSONObject ret = push.pushSingleDevice('token', mess);
```

Return value：

```
{"ret_code":0}                                          //成功
{"ret_code":-1, "err_msg":"error description"}          //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.2 pushSingleAccount 推送消息给单个账号

```
public JSONObject pushSingleAccount(int deviceType, String account, Message message) //android 使用
public JSONObject pushSingleAccount(int deviceType, String account, MessageIOS message, int environment) //IOS 使用
```

备注：设备的账号由终端 SDK 在调用推送注册接口时设置，详情参考终端 SDK 文档。

参数说明：

参数名	类型	必需	默认值	参数描述
deviceType	int	否	0	请填 0

account	string	是	无	推送目标账号
message	Message	是	无	消息体，IOS 推送使用 MessageIOS 类型
environment	int	**iOS 专用**	0	向 iOS 设备推送时必填，IOSENV_PROD 表示推送生产环境；IOSENV_DEV 表示推送开发环境。 推送 Android 平台不填或填 0

Example:

```
XingeApp push = new XingeApp(000, 'myKey');
Message mess = new Message(); //$mess = new MessageIOS();
//完善 Message 消息
...
JSONObject ret = push. pushSingleAccount (0, 'nickName', mess);
```

Return value:

```
{"ret_code":0}                                          //成功
{"ret_code":-1, "err_msg":"error description"}          //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.3 pushAccountList 推送消息给多个账号

此接口不支持定时推送。
如果推送量不大，推荐使用 pushSingleAccount 接口。
如果推送目标帐号数量很大（比如≥10000），推荐使用 pushAccountListMultiple 接口，用户请自行比较异同。

```
public JSONObject pushAccountList(int deviceType, List<String> accountList, Message message)

//android 使用

public JSONObject pushAccountList(int deviceType, List<String> accountList, MessageIOS message,

int environment) //ios 使用
```

备注：设备的账号由终端 SDK 在调用推送注册接口时设置，详情参考终端 SDK 文档。

参数说明：

参数名	类型	必需	默认值	参数描述
deviceType	int	否	0	请填 0
accountList	List	是	无	元素为推送的目标账号，string 类型，数量有限制，目前为 100 个
message	Message	是	无	消息体，IOS 推送使用 MessageIOS 类型 批量发送 message 不使用 sendTime
environment	int	**iOS 专用**	0	向 iOS 设备推送时必填，IOSENV_PROD 表示推送生产环境；IOSENV_DEV 表示推送开发环境。 推送 Android 平台不填或填 0

Example:

```
XingeApp push = new XingeApp(000, 'myKey');
```

```
Message mess = new Message(); //$mess = new MessageIOS();
//完善 Message 消息
...
List<String> accountList = new ArrayList<String>();
accountList.add('nickName');
JSONObject ret = push.pushAccountList (0, accountList, mess);
```

Return value:

```
//result 为一个整数数组，按顺序记录每一个 account 的推送结果
{"ret_code":0, "result":[0,2,0]}                    //成功
{"ret_code":-1, "err_msg":"error description"}      //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.4 pushAllDevice 推送消息给单个 app 的所有设备

```
public JSONObject pushAllDevice(int deviceType,Message message) //android 使用
public JSONObject pushAllDevice(int deviceType,MessageIOS message,int environment) //IOS 使用
```

参数说明：

参数名	类型	必需	默认值	参数描述
deviceType	int	否	0	请填 0
message	Message	是	无	消息体，IOS 推送使用 MessageIOS 类型
environment	int	**iOS 专用**	0	向 iOS 设备推送时必填，IOSENV_PROD 表示推送生产环境；IOSENV_DEV 表示推送开发环境。 推送 Android 平台不填或填 0

Example:

```
XingeApp push = new XingeApp(000, 'myKey');
Message mess = new Message(); //$mess = new MessageIOS();
//完善 Message 消息
...
JSONObject ret = push.pushAllDevice (0, mess);
```

Return value:

```
{"ret_code":0, "result":{"push_id":"11121"}}        //成功
{"ret_code":-1, "err_msg":"error description"}      //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.5 pushTags 推送消息给 tags 指定的设备

```
public JSONObject pushTags(int deviceType,List<String> tagList,String tagsOp,Message message)
```

//android 使用

```
public JSONObject pushTags(int deviceType,List<String> tagList,String tagsOp,MessageIOS message,
int environment) //IOS 使用
```

参数说明：

参数名	类型	必需	默认值	参数描述
deviceType	int	否	0	请填 0
tagList	List	是	无	指定推送目标的 tag 列表，每个 tag 是一个 string
tagsOp	String	是	无	多个 tag 的运算关系，取值为 AND 或 OR
message	Message	是	无	消息体，IOS 推送使用 MessageIOS 类型
environment	int	**iOS 专用**	0	向 iOS 设备推送时必填，IOSENV_PROD 表示推送生产环境；IOSENV_DEV 表示推送开发环境。 推送 Android 平台不填或填 0

Example：

```
XingeApp push = new XingeApp(000, 'myKey');
Message mess = new Message(); //$mess = new MessageIOS();
//完善 Message 消息
...
List<String> tagList = new ArrayList<String>();
tagList.add('tag');
JSONObject ret = push.pushTags(0, tagList, 'OR', mess);
```

Return value：

```
{"ret_code":0, "result":{"push_id":"11121"}}                    //成功
{"ret_code":-1, "err_msg":"error description"}                  //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.6 createMultipush 创建大批量推送消息

此接口创建的任务不支持定时推送

```
public JSONObject createMultipush(Message message)//android 使用
```

```
public JSONObject createMultipush(MessageIOS message, int environment)//ios 使用
```

备注：设备的账号由终端 SDK 在调用推送注册接口时设置，详情参考终端 SDK 文档。

参数说明：

参数名	类型	必需	默认值	参数描述
message	Message	是	无	消息体，IOS 推送使用 MessageIOS 类型 批量发送 message 不使用 sendTime
environment	int	**iOS 专用**	0	向 iOS 设备推送时必填，IOSENV_PROD 表示推送生产环境；IOSENV_DEV 表示推送开发环境。

				推送 Android 平台不填或填 0

Example:

```
XingeApp push = new XingeApp(000, 'myKey');
Message mess = new Message(); //$mess = new MessageIOS();
//完善 Message 消息
...
JSONObject ret = push.createMultipush (mess);
```

Return value:

```
{"ret_code":0, "result":{"push_id":"11121"}}                    //成功
{"ret_code":-1, "err_msg":"error description"}                  //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.7 pushAccountListMultiple 推送消息给大批量账号(可多次)

循环任务、定时任务量大或其他特殊需求，建议用户采用此接口自行控制发送时间

```
public JSONObject pushAccountListMultiple(int pushId, List<String> accountList)
```

备注：设备的账号由终端 SDK 在调用推送注册接口时设置，详情参考终端 SDK 文档。

参数说明：

参数名	类型	必需	默认值	参数描述
pushId	int	是	无	createMultipush 接口返回的 push_id
accountList	List	是	无	元素为推送的目标账号，string 类型，数量有限制，目前为 1000 个

Example:

```
XingeApp push = new XingeApp(000, 'myKey');
int pushId = 11121;
List<String> accountList1 = new ArrayList<String>();
accountList1.add('nickName1');
JSONObject ret1 = push.pushAccountListMultiple (pushId, accountList1);
List<String> accountList2 = new ArrayList<String>();
accountList2.add('nickName2');
JSONObject ret2 = push.pushAccountListMultiple (pushId, accountList2);
```

Return value:

```
{"ret_code":0}                                                  //成功
{"ret_code":-1, "err_msg":"error description"}                  //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.8 pushDeviceListMultiple 推送消息给大批量设备(可多次)

循环任务、定时任务量大或其他特殊需求，建议用户采用此接口自行控制发送时间

```
public JSONObject pushDeviceListMultiple(int pushId, List<String> deviceList)
```

参数说明：

参数名	类型	必需	默认值	参数描述
pushId	int	是	无	createMultipush 接口返回的 push_id
deviceList	List	是	无	元素为推送的目标 token，string 类型，数量有限制，目前为 1000 个

Example：

```
XingeApp push = new XingeApp(000, 'myKey');
int pushId = 11121;
List<String> deviceList1 = new ArrayList<String>();
deviceList1.add('token1');
JSONObject ret1 = push.pushDeviceListMultiple (pushId, deviceList1);
List<String> deviceList2 = new ArrayList<String>();
deviceList2.add('token2');
JSONObject ret2 = push.pushDeviceListMultiple (pushId, deviceList2);
```

Return value：

```
{"ret_code":0}                                          //成功
{"ret_code":-1, "err_msg":"error description"}          //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.9 queryPushStatus 查询群发消息发送状态

```
public JSONObject queryPushStatus(List<String> pushIdList)
```

参数说明：

参数名	类型	必需	默认值	参数描述
pushIdList	List	是	无	推送任务 id 列表，每个 id 为一个 string

Example：

```
XingeApp push = new XingeApp(000, 'myKey');
List<String> pushIdList = new ArrayList<String>();
pushIdList.add('11211');
JSONObject ret = push->queryPushStatus(pushIdList);
```

Return value：

```
//返回由
```

```
//{
//   "push_id":"11121",  //任务 ID
//   "status":1,         //状态 0->待推送; 1、6->推送中; 2->推送完成; 3->推送失败
//   "start_time":"2015-02-02 12:12:12",
//}
//对象组成的数组
{"ret_code":0, "result": {"list": [{"push_id":"11121", "status":1, "start_time":"2015-02-02 12:12:12"}] } }                    //成功
{"ret_code":-1, "err_msg":"error description"}                                                                                  //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.10 queryDeviceCount 查询应用覆盖的设备数

```
public JSONObject queryDeviceCount()
```

Example:

```
XingeApp push = new XingeApp(000, 'myKey');
JSONObject ret = push.queryDeviceCount();
```

Return value:

```
{"ret_code":0, "result":{"device_num":10000000}}                    //成功
{"ret_code":-1, "err_msg":"error description"}                      //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.11 queryTags 查询应用的 tags

```
public JSONObject queryTags(int start,int limit)
```

参数说明：

参数名	类型	必需	默认值	参数描述
start	int	否	0	开始位置
limit	int	否	100	限制结果数量

Example:

```
XingeApp push = new XingeApp(000, 'myKey');
JSONObject ret = push.queryTags(0, 100);
```

Return value:

```
//其中 total 为标签总数，tags 为要查询区间的所有 tag
{"ret_code":0, "result":{"total":3, "tags":["tag1","tag2","tag3"]}}  //成功
{"ret_code":-1, "err_msg":"error description"}                       //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.12 cancelTimingPush 取消尚未推送的定时消息

```
public JSONObject cancelTimingPush(int pushId)
```

参数说明：

参数名	类型	必需	默认值	参数描述
pushId	string	是	无	推送任务 id

Example：

```
XingeApp push = new XingeApp(000, 'myKey');
JSONObject ret = push.cancelTimingPush('1234');
```

Return value：

```
{"ret_code":0}                                          //成功
{"ret_code":-1, "err_msg":"error description"}          //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.13 BatchSetTag 批量为 token 设置标签

```
public JSONObject BatchSetTag(List<TagTokenPair> tagTokenPairs)
```

参数说明：

参数名	类型	必需	默认值	参数描述
tagTokenPairs	List	是	无	元素必须是 TagTokenPair。后台将对每个 pair 里的 token 设置相应的标签，每次调用最多允许输入 20 个 pair。

Example：

```
XingeApp push = new XingeApp(000, 'myKey');
List<TagTokenPair> pairs = new ArrayList<TagTokenPair>();
pairs.add(new TagTokenPair("tag1","token0000000000000000000000000001"));
pairs.add(new TagTokenPair("tag2","token0000000000000000000000000002"));
JSONObject ret = push.BatchSetTag(pairs);
```

Return value：

```
{"ret_code":0}                                          //成功
{"ret_code":-1, "err_msg":"error description"}          //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.14 BatchDelTag 批量为 token 删除标签

```
public JSONObject BatchDelTag(List<TagTokenPair> tagTokenPairs)
```

参数说明：

参数名	类型	必需	默认值	参数描述
tagTokenPairs	array	是	无	元素必须是 TagTokenPair。后台将对每个 pair 里的 token 删除相应的标签，每次调用最多允许输入 20 个 pair。

Example:

```
XingeApp push = new XingeApp(000, 'myKey');
List<TagTokenPair> pairs = new ArrayList<TagTokenPair>();
pairs.add(new TagTokenPair("tag1","token0000000000000000000000000001"));
pairs.add(new TagTokenPair("tag2","token0000000000000000000000000002"));
JSONObject ret = push.BatchDelTag($pairs);
```

Return value:

```
{"ret_code":0}                                                    //成功
{"ret_code":-1, "err_msg":"error description"}                    //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.15 queryTokenTags 查询 token 的 tags

```
public JSONObject queryTokenTags(String deviceToken)
```

参数说明：

参数名	类型	必需	默认值	参数描述
deviceToken	String	是		

Example:

```
XingeApp push = new XingeApp(000, 'myKey');
JSONObject ret = push.queryTokenTags("token0000000000000000000000000002");
```

Return value:

```
{"ret_code":0, "result":{"tags":["tag1","tag2","tag3"]}}          //成功
{"ret_code":-1, "err_msg":"error description"}                    //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.16 queryTagTokenNum 查询 tag 下 token 的数目

```
public JSONObject queryTagTokenNum(String tag)
```

参数说明：

参数名	类型	必需	默认值	参数描述
tag	string	是		

Example:

```
XingeApp push = new XingeApp(000, 'myKey');
JSONObject ret = push.queryTagTokenNum("beijing");
```

Return value:

```
{"ret_code":0, "result":{"device_num":100000}}                    //成功
{"ret_code":-1, "err_msg":"error description"}                    //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.17 queryInfoOfToken 查询 token 的相关信息

```
public JSONObject queryInfoOfToken(String deviceToken)
```

参数说明：

参数名	类型	必需	默认值	参数描述
deviceToken	String	是		

Example：

```
XingeApp push = new XingeApp(000, 'myKey');
JSONObject ret = push.queryInfoOfToken("token0000000000000000000000000002");
```

Return value:

```
//isReg:1(已注册), 0(未注册)
//connTimestamp:最近一次活跃的时间戳
//msgsNum:该 token 未获取的离线消息数
{"ret_code":0, "result":{"isReg":1,"connTimestamp":1426493097,"msgsNum":2}}
                                                                  //成功
{"ret_code":-1, "err_msg":"error description"}                    //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.18 queryTokensOfAccount 查询 account 绑定的 token

```
public JSONObject queryTokensOfAccount(String account)
```

参数说明：

参数名	类型	必需	默认值	参数描述
account	String	是		

Example：

```
XingeApp push = new XingeApp(000, 'myKey');
JSONObject ret = push.queryTokensOfAccount("nickName");
```

Return value:

```
{"ret_code":0, "result":{"tokens":["token1","token2","token3"]}}  //成功
{"ret_code":-1, "err_msg":"error description"}                    //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.19 deleteTokenOfAccount 删除 account 绑定的 token

```
public JSONObject deleteTokenOfAccount (String account, String token)
```

参数说明：

参数名	类型	必需	默认值	参数描述
account	String	是	无	要删除 token 的 account
token	String	是	无	要删除的 token

Example：

```
XingeApp push = new XingeApp(000, 'myKey');
JSONObject ret = push.deleteTokenOfAccount ("nickName", "token3");
```

Return value：

```
{"ret_code":0, "result":{"tokens":["token1","token2"]}}          //成功
{"ret_code":-1, "err_msg":"error description"}                   //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

3.2.20 deleteAllTokensOfAccount 删除 account 绑定的所有 token

```
public JSONObject delteAllTokensOfAccount(String account)
```

参数说明：

参数名	类型	必需	默认值	参数描述
account	String	是	无	要删除所有 token 的 account

Example：

```
XingeApp push = new XingeApp(000, 'myKey');
JSONObject ret = push.deleteAllTokensOfAccount("nickName");
```

Return value：

```
{"ret_code":0}                                                   //成功
{"ret_code":-1, "err_msg":"error description"}                   //失败
```

注：ret_code 为 0 表示成功，其他为失败，具体请查看附录。

4 附录

4.1 通用返回码描述

ret_code 含义如下

值	含义
0	调用成功
-1	参数错误，请对照文档检查请求参数
-2	请求时间戳不在有效期内，检查请求的时间戳设置是否正确，机器时间是否正确
-3	sign 校验无效，检查 access id 和 secret key（注意不是 access key）
2	参数错误，请对照文档检查请求参数
7	别名/账号绑定的终端数满了（10 个），请解绑部分终端
14	收到非法 token，例如 ios 终端没能拿到正确的 token，请检查 token 是否正确
15	信鸽逻辑服务器繁忙，请稍后再试
20	鉴权错误，请检查 access id 和 access key 是否正确
40	推送的 token 没有在信鸽中注册，或者推送的帐号没有绑定 token，请检查注册逻辑
48	推送的账号没有在信鸽中注册，请检查注册逻辑
63	标签系统忙，请稍后再试
71	APNS 服务器繁忙，请稍后再试
73	消息字符数超限，请缩小消息内容
76	请求过于频繁，请稍后再试
78	循环任务参数错误，请对照文档检查请求参数
100	APNS 证书错误。请重新提交正确的证书
其他	内部错误