魅族开放平台PUSH系统JAVA版本SDK

JavaPushSdk发布说明(请使用最新版本)

- 中央仓库获取 MVN Repository 或者 Central Repository
- 直接下载获取 Java Server SDK

更新日志

[2021-11-11]V1.2.9.2021111_release

- 增加通知栏图片，展开大图推送支持

[2019-01-14]V1.2.8.20190114_release

- update log config

[2018-11-06]V1.2.7.20181228_release

- 多包名功能代码

[2018-11-06]V1.2.7.20181106_release

- fastjson升级版本1.2.51，fix远程代码执行高危安全漏洞

[2018-03-07]V1.2.7.20180307_release

- 增加sdk日志配置文件

[2017-11-23]V1.2.6.20171123_release

- 增加pushId开关关闭状态返回

[2017-11-20]V1.2.5.20171120_release

- 通知栏消息聚合功能

[2017-08-15]V1.2.4.20170815_release

- 推送&订阅接口域名解耦

[2017-07-18]V1.2.3.20170718_release

- 通知栏pushId、alias增加消息回执功能

[2017-06-09]V1.2.2.20170609_release

- 开放统计接口：获取应用推送统计（最长跨度30天）

[2017-04-28]V1.2.1.20170428_release

- 开放别名、标签订阅服务接口
- 开放通知栏、透传消息开关接口

[2017-03-21]V1.1.1.20170321_release

- fixed #12

[2017-02-16]V1.1.0.20170216_release

- 推送结果增加msgId
- 通过msgId以及推送目标在推送平台查询具体的推送日志明细

[2016-12-18]V1.0.0.20161218_release

- 1.0.0标准版

说明

- 为优化flyme系统整体功耗，推送平台已经限制透传消息推送的使用
- 受影响的接口及功能：

1. pushId透传消息推送(pushMessage)
2. 别名透传消息推送(pushMessageByAlias)
3. 获取taskId的透传推送(getTaskId)
4. 全网透传推送(pushToApp)
5. 标签透传推送(pushToTag)
6. 在平台上进行的透传推送

目录

类型定义

推送服务(IFlymePush)

> 调用该类实例的方法来推送消息， 构造函数说明如下：

参数名称	类型	必填	默认	描述
appSecret	String	是	null	注册应用appSecret
useSSL	Boolean	否	false	是否使用https接口调用：true 使用https连接，false使用http连接

订阅服务(IFlymePushSub)

> 调用该类实例的方法做别名、标签、推送开关订阅服务， 构造函数说明如下：

参数名称	类型	必填	默认	描述
appId	Long	是	null	注册应用appId
appSecret	String	是	null	注册应用appSecret
useSSL	Boolean	否	false	是否使用https接口调用：true 使用https连接，false使用http连接

通知栏消息体(Message)

> 推送消息实体（抽象类）

子类	说明
VarnishedMessage	通知栏消息体
UnVarnishedMessage	透传消息体

通知栏消息(VarnishedMessage)

参数名称	类型	必填	默认	描述
appId	Long	是	null	注册应用appId
restrictedPackageNames	String[]	否	null	多包名配置【最长50】
title	String	是	null	推送标题, 【字数限制1~32】
content	String	是	null	推送内容, 【字数限制1~100】
noticeBarType	int	否	0	通知栏样式(0, "标准"),(1, "图片"),(2, "安卓原生")【非必填，默认值为0】
noticeBarImgUrl	String	否	null	通知栏图片, 【noticeBarType为图片时，必填】
noticeExpandType	int	否	0	展开方式 (0, "标准"),(1, "文本"),(2, "大图")【非必填，默认值为0】
noticeExpandContent	String	否	null	展开内容, 【noticeExpandType为文本时，必填】
noticeExpandImgUrl	String	否	null	展开大图url, 【noticeExpandType为大图时，必填】
clickType	int	否	0	点击动作 (0,"打开应用"),(1,"打开应用页面"),(2,"打开URI页面"),(3, "应用客户端自定义")【非必填，默认值为0】
url	String	否	null	URI页面地址, 【clickType为打开URI页面时，必填】
parameters	JSONObject	否	null	透传参数 【JSON格式，非必填】
activity	String	否	null	应用页面地址, 【clickType为打开应用页面时，格式 pkg.activity eg: com.meizu.upspushdemo.TestActivity 必填】
customAttribute	String	否	null	应用客户端自定义内容, 【clickType为应用客户端自定义时，必填】
isOffLine	Boolean	否	true	是否进离线消息, (false 否 true 是) 【非必填，默认值为true】
validTime	int	否	24	有效时长 (1~72小时内的正整数), 【isOffLine值为true时，必填，值的范围1~72】
pushTimeType	int	否	0	定时推送 (0, "即时"),(1, "定时"), 【只对全部用户推送生效】
startTime	Date	否	null	任务定时开始时间,【非必填，pushTimeType为True必填】只对全部用户推送生效
isFixSpeed	Boolean	否	false	是否定速推送, 【非必填，默认值为False】
fixSpeedRate	Long	否	0	定速速率,【isFixSpeed为true时，必填】
isSuspend	Boolean	否	true	是否通知栏悬浮窗显示 (true显示，false不显示)【非必填，默认True】
isClearNoticeBar	Boolean	否	true	是否可清除通知栏 (true可以，false不可以) ，【非必填，默认true】
isFixDisplay	Boolean	否	false	是否定时展示 【非必填,默认false】
fixDisplayTime	(Date,Date)	否	(null,null)	定时展示开始,结束时间 【fixDisplay为true时，必填，并且开始时间要晚于结束时间】
vibrate	Boolean	否	true	震动 (false关闭 true 开启) , 【非必填，默认true】
lights	Boolean	否	true	闪光 (false关闭 true 开启) , 【非必填，默认true】

参数名称	类型	必填	默认	描述
sound	Boolean	否	true	声音 (false关闭 true 开启)，【非必填，默认true】
notifyKey	String	否	null	分组合并推送的key,凡是带有此key的通知栏消息只会显示最后到达的一条。由数字([0-9]), 大小写字母([a-zA-Z]), 下划线(_)和中划线(-)组成, 长度不大于8个字符
extra	Map<String, String>	否	null	回执参数（ExtraParam）

透传消息(UnVarnishedMessage)

参数名称	类型	必填	默认	描述
appId	Long	是	null	注册应用appId
title	String	否	null	推送标题,任务推送建议填写，方便数据查询,【字数限制1~32】
content	String	是	null	推送内容,【必填，字数限制3800byte以内】
isOffLine	Boolean	否	true	是否进离线消息,【非必填，默认为true】
validTime	int	否	24	有效时长 (1~72小时内的正整数),【isOffLine值为true时，必填，值的范围1--72】
pushTimeType	int	否	0	定时推送 (0, "即时"),(1, "定时"), 【只对全部用户推送生效】
startTime	Date	否	null	任务定时开始时间, 【pushTimeType为1必填】只对全部用户推送生效
isFixSpeed	Boolean	否	false	是否定速推送,【非必填，默认值为false】
fixSpeedRate	Long	否	0	定速速率 【isFixSpeed为true时，必填】

接口返回值(ResultPack)

方法名称	类型	描述
code()	String	接口响应码
Comment()	String	接口响应描述
value()	T	接口响应内容
errorCode()	Enum	接口响应异常枚举 详见ErrorCode

消息推送结果(PushResult)

方法名称	类型	描述
getMsgId()	String	推送消息ID，用于推送流程明细排查
getRespTarget()	Map	推送目标结果状态(KEY：推送响应码 VALUE：响应码对应的目标用户)

接口响应码定义(ErrorCode)

名称	Code	Commen
UNKNOWN_ERROR	-1	未知错误
SUCCESS	200	成功
SYSTEM_ERROR	1001	系统错误
SYSTEM_BUSY	1003	服务器忙
PARAMETER_ERROR	1005	参数错误，请参考API文档
INVALID_SIGN	1006	签名认证失败
INVALID_APPLICATION_ID	110000	appId不合法
INVALID_APPLICATION_KEY	110001	appKey不合法
UNSUBSCRIBE_PUSHID	110002	pushId未注册
INVALID_PUSHID	110003	pushId非法
PARAM_BLANK	110004	参数不能为空
APP_IN_BLACK_LIST	110009	应用被加入黑名单
APP_REQUEST_EXCEED_LIMIT	110010	应用请求频率过快
APP_PUSH_TIME_EXCEED_LIMIT	110051	超过该应用的次数限制
APP_REQUEST_PUSH_LIMIT	110019	超过该应用每天推送次数限制
INVALID_APPLICATION_PACKAGENAME	110031	packageName不合法
INVALID_TASK_ID	110032	非法的taskId
INVALID_APPLICATION_SECRET	110033	非法的appSecret
DIRECT_OUT_LIMIT	110053	透传超过限制

推送响应码定义（PushResponseCode）

名称	Code	Commen
RSP_NO_AUT	201	没有权限，服务器主动拒绝
RSP_DB_ERROR	501	推送消息失败（db_error）
RSP_INTERNAL_ERROR	513	推送消息失败
RSP_SPEED_LIMIT	518	推送超过配置的速率
RSP_OVERFLOW	519	推送消息失败服务过载
RSP_REPEATED	520	消息折叠（1分钟内同一设备同一应用消息收到多次，默认5次）
RSP_UNSUBSCRIBE_PUSHID	110002	pushId失效(pushId未订阅)
RSP_INVALID_PUSHID	110003	pushId非法
RSP_UNSUBSCRIBE_ALIAS	110005	别名未订阅(包括推送开关关闭的设备)
RSP_OFF_PUSHID	110010	pushId失效(消息开关关闭)

推送类型（PushType）

枚举	类型	描述
STATUSBAR	Enum	通知栏消息类型
DIRECT	Enum	透传消息类型

推送标签集合类型（ScopeType）

枚举	类型	描述
UNION	Enum	并集
INTERSECTION	Enum	交集

任务推送统计（TaskStatistics）

名称	类型	描述
targetNo	Long	目标数
validNo	Long	有效数
pushedNo	Long	推送数
acceptNo	Long	接受数
displayNo	Long	展示数
clickNo	Long	点击数

任务推送统计（天）（DailyPushStatics）

名称	类型	描述
date	Date	日期
targetNo	Long	目标数
validNo	Long	有效数
pushedNo	Long	推送数
acceptNo	Long	接受数
displayNo	Long	展示数
clickNo	Long	点击数

订阅别名信息 (AliasInfo)

名称	类型	描述
pushId	String	订阅pushId
alias	String	订阅别名

订阅标签信息 （ TagInfo ）

名称	类型	描述
pushId	String	订阅pushId
tags	List	订阅标签

订阅开关状态 (SwitchStatusInfo)

名称	类型	描述
pushId	String	订阅pushId
statusbarSwitch	boolean	通知栏开关状态
directSwitch	boolean	透传开关状态

回执类型（ CallBackType ）

枚举	类型	描述
RECEIVE	Enum	送达回执
CLICK	Enum	点击回执
RECEIVE_CLICK	Enum	送达与点击回执

回执参数（ ExtraParam ）

枚举	类型	描述
CALLBACK	Enum	回执接口（第三方接收回执的Http接口, 最大长度128字节）
CALLBACK_PARAM	Enum	回执参数（第三方自定义回执参数, 最大长度64字节）
CALLBACK_TYPE	Enum	回执类型（(1-送达回执, 2-点击回执, 3-送达与点击回执), 默认3）

接口说明

图片上传

描述

> 通知栏图片、展开大图使用的图片链接需要由此接口上传生成

注：上传的图片最多保留三天，请在需要时才上传

uploadImage上传接口

- 接口说明

接口	说明
public ResultPack uploadImage(long appId, int imgType, String imgUrl)	上传图片

- 参数说明

参数名称	类型	必需	默认	描述
appId	long	是	null	应用id
imgType	int	是	null	图片使用类型
imgUrl	String	是	null	图片来源url

- 返回值

```
ImageInfo
imgType：图片类型
imgUrl：推送时使用的url
```

- 示例

```
/**
 * 图片上传
 */
@Test
public void testPicUpload()throws Exception{
    String imgUrl = "";
    IFlymePush push = new IFlymePush(APP_SECRET_KEY);
    ResultPack<ImageInfo> ret =
push.uploadImage(appId,ImageInfo.NOTICE_BAR_IMG,imgUrl);
    System.out.println(ret.toString());
}
```

非任务推送

描述

> 向指定的pushId推送消息

注：推送平台使用pushId来标识每个独立的用户，每一台终端上每一个app拥有一个独立的pushId

应用场景

> - 场景1：查找手机业务需要远程定位位置，可发送消息指令到对应的设备
> - 场景2：社区用户回帖消息提醒，用户对发表的帖子有最新回复时，消息提醒发帖者

pushId通知栏消息推送（pushMessage）

- 接口说明

接口	说明
`ResultPack<PushResult> pushMessage(VarnishedMessage message, List<String> pushIds)`	推送通知栏消息
`ResultPack<PushResult> pushMessage(VarnishedMessage message, List<String> pushIds, int retries)`	推送通知栏消息

- 参数说明

参数名称	类型	必需	默认	描述
message	VarnishedMessage	是	null	推送消息
pushIds	List	是	null	推送目标，一批最多不能超过1000个
retries	int	否	0	超时or异常重试次数

- 返回值

```
PushResult

msgId;  推送消息ID，用于推送流程明细排查
respTarget;  推送目标结果状态(key：推送响应码  value：响应码对应的目标用户 )

注：只返回不合法、超速以及推送失败的目标用户，业务一般对超速的pushId处理。
```

- 示例

```
    /**
     * 通知栏消息推送（pushMessage）
     * @throws Exception
     */
    @Test
    public void testVarnishedMessagePush() throws Exception {
        //推送对象
        IFlymePush push = new IFlymePush(APP_SECRET_KEY);

        //组装消息
        VarnishedMessage message = new VarnishedMessage.Builder().appId(appId)
                .restrictedPackageNames(new String[]{"com.xxx.abc"})//多包名推送时
才需配置，不填表示所有
                .title("Java SDK 推送标题").content("Java SDK 推送内容")
                .noticeExpandType(1)
                .noticeExpandContent("展开文本内容")

.clickType(2).url("http://www.baidu.com").parameters(JSON.parseObject("
{\"k1\":\"value1\",\"k2\":0,\"k3\":\"value3\"}"))
                .offLine(true).validTime(12)

.suspend(true).clearNoticeBar(true).vibrate(true).lights(true).sound(true)
                .build();

        //目标用户
        List<String> pushIds = new ArrayList<String>();
        pushIds.add("pushId_1");
        pushIds.add("pushId_2");
        // 1 调用推送服务
        ResultPack<PushResult> result = push.pushMessage(message, pushIds);
        if (result.isSucceed()) {
            // 2 调用推送服务成功 （其中map为设备的具体推送结果，一般业务针对超速的code类型
做处理）
            PushResult pushResult = result.value();
            String msgId = pushResult.getMsgId();//推送消息ID，用于推送流程明细排查
            Map<Integer, List<String>> targetResultMap =
pushResult.getRespTarget();//推送结果，全部推送成功，则map为empty
            System.out.println("push msgId:" + msgId);
            System.out.println("push targetResultMap:" + targetResultMap);
            if (targetResultMap != null && !targetResultMap.isEmpty()) {
                // 3 判断是否有获取超速的target
                if
(targetResultMap.containsKey(PushResponseCode.RSP_SPEED_LIMIT.getValue())) {
                    // 4 获取超速的target
                    List<String> rateLimitTarget =
targetResultMap.get(PushResponseCode.RSP_SPEED_LIMIT.getValue());
```

```
                    System.out.println("rateLimitTarget is :" +
rateLimitTarget);
                    //TODO 5 业务处理，重推......
                }
            }
        } else {
            // 调用推送接口服务异常 eg: appId、appKey非法、推送消息非法.....
            // result.code(); //服务异常码
            // result.comment();//服务异常描述

           //全部超速
            if
(String.valueOf(ErrorCode.APP_REQUEST_EXCEED_LIMIT.getValue()).equals(result.cod
e())) {
                //TODO 5 业务处理，重推......
            }
            System.out.println(String.format("pushMessage error code:%s
comment:%s", result.code(), result.comment()));
        }
    }
```

pushId透传消息推送（pushMessage）

- 接口说明

接口	说明
`ResultPack<PushResult> pushMessage(UnVarnishedMessage message, List<String> pushIds)`	推送透传消息
`ResultPack<PushResult> pushMessage(UnVarnishedMessage message, List<String> pushIds, int retries)`	推送透传消息

- 参数说明

参数名称	类型	必需	默认	描述
message	UnVarnishedMessage	是	null	推送消息
pushIds	List	是	null	推送目标，一批最多不能超过1000个
retries	int	否	0	超时or异常重试次数

- 返回值

```
PushResult

msgId;  推送消息ID，用于推送流程明细排查
respTarget;  推送目标结果状态(key：推送响应码  value：响应码对应的目标用户 )
注：只返回不合法、超速以及推送失败的目标用户，业务一般对超速的pushId处理。
```

- 示例

```
    /**
     * 透传消息推送（pushMessage）
```

```
     * @throws Exception
     */
    @Test
    public void testUnVarnishedMessagePush() throws Exception {
        //推送对象
        IFlymePush push = new IFlymePush(APP_SECRET_KEY);
        //组装透传消息
        UnVarnishedMessage message = new UnVarnishedMessage.Builder()
                .appId(appId)
                .title("Java SDK 透传推送标题")
                .content("Java Sdk透传推送内容")
                .isOffLine(true)
                .validTime(10)
                .build();

        //目标用户
        List<String> pushIds = new ArrayList<String>();
        pushIds.add("pushId_1");
        pushIds.add("pushId_2");

        // 1 调用推送服务
         ResultPack<PushResult> result = push.pushMessage(message, pushIds);
         if (result.isSucceed()) {
             // 2 调用推送服务成功 （其中map为设备的具体推送结果，一般业务针对超速的code类型做处理）
             PushResult pushResult = result.value();
             String msgId = pushResult.getMsgId();//推送消息ID，用于推送流程明细排查
             Map<Integer, List<String>> targetResultMap = pushResult.getRespTarget();//推送结果，全部推送成功，则map为empty
             System.out.println("push msgId:" + msgId);
             System.out.println("push targetResultMap:" + targetResultMap);
             if (targetResultMap != null && !targetResultMap.isEmpty()) {
                 // 3 判断是否有获取超速的target
                 if (targetResultMap.containsKey(PushResponseCode.RSP_SPEED_LIMIT.getValue())) {
                     // 4 获取超速的target
                     List<String> rateLimitTarget = targetResultMap.get(PushResponseCode.RSP_SPEED_LIMIT.getValue());
                     System.out.println("rateLimitTarget is :" + rateLimitTarget);
                     //TODO 5 业务处理，重推......
                 }
             }
         } else {
             // 调用推送接口服务异常 eg: appId、appKey非法、推送消息非法.....
             // result.code(); //服务异常码
             // result.comment();//服务异常描述

             //全部超速
             if (String.valueOf(ErrorCode.APP_REQUEST_EXCEED_LIMIT.getValue()).equals(result.code())) {
                 //TODO 5 业务处理，重推......
             }
             System.out.println(String.format("pushMessage error code:%s comment:%s", result.code(), result.comment()));
         }
    }
```

别名通知栏消息推送（pushMessageByAlias）

- 接口说明

接口	说明
`ResultPack<PushResult> pushMessageByAlias(VarnishedMessage message, List<String> alias)`	推送通知栏消息
`ResultPack<PushResult> pushMessageByAlias(VarnishedMessage message, List<String> alias, int retries)`	推送通知栏消息

- 参数说明

参数名称	类型	必需	默认	描述
message	VarnishedMessage	是	null	推送消息
alias	List	是	null	推送目标，一批最多不能超过1000个
retries	int	否	0	超时or异常重试次数

- 返回值

```
PushResult

msgId;  推送消息ID，用于推送流程明细排查
respTarget;  推送目标结果状态(key：推送响应码  value：响应码对应的目标用户 )
注：只返回不合法、超速以及推送失败的目标用户
```

- 示例

```
/**
 * 别名通知栏消息推送（pushMessage）
 *
 * @throws Exception
 */
@Test
public void testVarnishedMessagePushByAlias() throws Exception {
    //推送对象
    IFlymePush push = new IFlymePush(APP_SECRET_KEY);

    //组装消息
    VarnishedMessage message = new VarnishedMessage.Builder().appId(appId)
            .title("Java SDK 推送标题").content("Java SDK 推送内容")
            .noticeExpandType(1)
            .noticeExpandContent("展开文本内容")

.clickType(2).url("http://push.meizu.com").parameters(JSON.parseObject("
{\"k1\":\"value1\",\"k2\":0,\"k3\":\"value3\"}"))
            .offLine(true).validTime(12)
            .isFixDisplay(true).fixDisplayTime(str2Date("2017-10-01 12:00:00"),
str2Date("2017-10-01 12:30:00"))
```

```
.suspend(true).clearNoticeBar(true).vibrate(true).lights(true).sound(true)
            .build();

        //目标用户
        List<String> alias = new ArrayList<String>();
        alias.add("Android");
        alias.add("alias2");
        // 1 调用推送服务
        ResultPack<PushResult> result = push.pushMessageByAlias(message, alias);
        if (result.isSucceed()) {
            // 2 调用推送服务成功 （其中map为设备的具体推送结果，一般业务针对超速的code类型做处理）
            PushResult pushResult = result.value();
            String msgId = pushResult.getMsgId();//推送消息ID，用于推送流程明细排查
            Map<Integer, List<String>> targetResultMap = pushResult.getRespTarget();//推送结果，全部推送成功，则map为empty
            System.out.println("push msgId:" + msgId);
            System.out.println("push targetResultMap:" + targetResultMap);
            if (targetResultMap != null && !targetResultMap.isEmpty()) {
                // 3 判断是否有获取超速的target
                if (targetResultMap.containsKey(PushResponseCode.RSP_SPEED_LIMIT.getValue())) {
                    // 4 获取超速的target
                    List<String> rateLimitTarget = targetResultMap.get(PushResponseCode.RSP_SPEED_LIMIT.getValue());
                    System.out.println("rateLimitTarget is :" + rateLimitTarget);
                    //TODO 5 业务处理，重推......
                }
            }
        } else {
            // 调用推送接口服务异常 eg: appId、appKey非法、推送消息非法.....
            // result.code(); //服务异常码
            // result.comment();//服务异常描述
             //全部超速
             if (String.valueOf(ErrorCode.APP_REQUEST_EXCEED_LIMIT.getValue()).equals(result.code())) {
                 //TODO 5 业务处理，重推......
             }
            System.out.println(String.format("pushMessage error code:%s comment:%s", result.code(), result.comment()));
        }
}
```

别名透传消息推送（pushMessageByAlias）

- 接口说明

接口	说明
`ResultPack<PushResult> pushMessageByAlias(UnVarnishedMessage message, List<String> alias)`	推送透传消息
`ResultPack<PushResult> pushMessageByAlias(UnVarnishedMessage message, List<String> alias, int retries)`	推送透传消息

- 参数说明

参数名称	类型	必需	默认	描述
message	UnVarnishedMessage	是	null	推送消息
alias	List	是	null	推送目标，一批最多不能超过1000个
retries	int	否	0	超时or异常重试次数

- 返回值

```
PushResult

msgId;  推送消息ID，用于推送流程明细排查
respTarget;  推送目标结果状态(key：推送响应码  value：响应码对应的目标用户 )
注：只返回不合法、超速以及推送失败的目标用户
```

- 示例

```
/**
 * 别名透传推送
 *
 * @throws Exception
 */
@Test
public void testUnVarnishedMessagePushByALias() throws Exception {
    //推送对象
    IFlymePush push = new IFlymePush(APP_SECRET_KEY);
    //组装透传消息
    UnVarnishedMessage message = new UnVarnishedMessage.Builder()
            .appId(appId)
            .title("Java SDK 透传推送标题")
            .content("Java Sdk透传推送内容")
            .build();

    //目标用户
    List<String> alias = new ArrayList<String>();
    alias.add("alias");
    alias.add("alias2");

        // 1 调用推送服务
        ResultPack<PushResult> result = push.pushMessageByAlias(message, alias);
        if (result.isSucceed()) {
            // 2 调用推送服务成功 （其中map为设备的具体推送结果，一般业务针对超速的code类型做处理）
            PushResult pushResult = result.value();
```

```
            String msgId = pushResult.getMsgId();//推送消息ID，用于推送流程明细排查
            Map<Integer, List<String>> targetResultMap =
pushResult.getRespTarget();//推送结果，全部推送成功，则map为empty
            System.out.println("push msgId:" + msgId);
            System.out.println("push targetResultMap:" + targetResultMap);
            if (targetResultMap != null && !targetResultMap.isEmpty()) {
                // 3 判断是否有获取超速的target
                if
(targetResultMap.containsKey(PushResponseCode.RSP_SPEED_LIMIT.getValue())) {
                    // 4 获取超速的target
                    List<String> rateLimitTarget =
targetResultMap.get(PushResponseCode.RSP_SPEED_LIMIT.getValue());
                    System.out.println("rateLimitTarget is :" +
rateLimitTarget);
                    //TODO 5 业务处理，重推......
                }
            }
        } else {
            // 调用推送接口服务异常 eg: appId、appKey非法、推送消息非法.....
            // result.code(); //服务异常码
            // result.comment();//服务异常描述

             //全部超速
             if
(String.valueOf(ErrorCode.APP_REQUEST_EXCEED_LIMIT.getValue()).equals(result.cod
e())) {
                 //TODO 5 业务处理，重推......
             }
            System.out.println(String.format("pushMessage error code:%s
comment:%s", result.code(), result.comment()));
        }
}
```

任务推送

描述

> 首先获取推送的taskId,然后通过taskId向指定的pushId推送消息。

应用场景

> 浏览器对指定的某一大批量pushId用户推送活动或者新闻消息，通过先获取taskId，然后通过taskId批量推送，推送过程中可以根据taskId时时获取推送统计结果。

获取推送taskId(getTaskId)

- 接口说明

接口	说明
ResultPack getTaskId(PushType pushType, Message message)	获取推送taskId

- 参数说明

参数名称	类型	必需	默认	描述
pushType	PushType	是	null	消息类型
message	Message	是	null	消息体

- 返回值

```
Long  任务ID
```

- 示例

```
    /**
     * 获取通知栏推送taskId(getTaskId)
     * @throws Exception
     */
    @Test
    public void testGetVarnishedMessageTaskId() throws Exception {
        //推送对象
        IFlymePush push = new IFlymePush(APP_SECRET_KEY);

        //组装消息
        VarnishedMessage message = new VarnishedMessage.Builder().appId(appId)
                .title("java Sdk推送标题").content("java Sdk 推送内容")
                .noticeExpandType(1)
                .noticeExpandContent("展开文本内容")

.clickType(2).url("http://www.baidu.com").parameters(JSON.parseObject("
{\"k1\":\"value1\",\"k2\":0,\"k3\":\"value3\"}"))
                .offLine(true).validTime(12)

.suspend(true).clearNoticeBar(true).vibrate(false).lights(false).sound(false)
                .fixSpeed(true).fixSpeedRate(20)
                .build();

        ResultPack<Long> result = push.getTaskId(PushType.STATUSBAR, message);
        System.out.println(result);
    }
```

```
    /**
     * 获取透传推送taskId(getTaskId)
     * @throws Exception
     */
    @Test
    public void testGetUnVarnishedMessageTaskId() throws Exception {
        //推送对象
        IFlymePush push = new IFlymePush(APP_SECRET_KEY);

        //组装消息
        UnVarnishedMessage message = new UnVarnishedMessage.Builder()
                .appId(appId)
                .title("java sdk 推送标题")
                .content("java sdk 推送内容")
                .build();
```

```
        ResultPack<Long> result = push.getTaskId(PushType.DIRECT, message);
        System.out.println(result);
    }
```

pushId消息推送（pushMessageByTaskId）

- 接口说明

接口	说明
`ResultPack<PushResult> pushMessageByTaskId(PushType pushType, long appId, long taskId, List<String> pushIds)`	任务消息推送
`ResultPack<PushResult> pushMessageByTaskId(PushType pushType, long appId, long taskId, List<String> pushIds, int retries)`	任务消息推送
- 参数说明	

参数名称	类型	必需	默认	描述
pushType	PushType	是	null	消息类型
appId	Long	是	null	推送应用ID
taskId	Long	是	null	推送任务ID
pushIds	List	是	null	推送目标，一批最多不能超过1000个
retries	int	否	0	超时or异常重试次数

- 返回值

```
PushResult

msgId;  推送消息ID，用于推送流程明细排查
respTarget;  推送目标结果状态(key：推送响应码  value：响应码对应的目标用户 )
注：只返回不合法、超速以及推送失败的目标用户，业务一般对超速的pushId对处理
```

- 示例

```
    /**
     * 任务消息推送（pushMessageByTaskId）
     * @throws IOException
     */
    @Test
    public void testPushPyTaskId() throws IOException {
        //推送对象
        IFlymePush push = new IFlymePush(APP_SECRET_KEY);

        //目标用户
        List<String> pushIds = new ArrayList<String>();
        pushIds.add("pushId_1");
        pushIds.add("pushId_2");
```

```
        //通知栏任务消息推送
        Long taskId = 123l;
        // 1 调用推送服务
        ResultPack<PushResult> result =
push.pushMessageByTaskId(PushType.STATUSBAR, appId, taskId, pushIds, 0);
        if (result.isSucceed()) {
            // 2 调用推送服务成功 （其中map为设备的具体推送结果，一般业务针对超速的code类型
做处理）
            PushResult pushResult = result.value();
            String msgId = pushResult.getMsgId();//推送消息ID，用于推送流程明细排查
            Map<Integer, List<String>> targetResultMap =
pushResult.getRespTarget();//推送结果，全部推送成功，则map为empty
            System.out.println("push msgId:" + msgId);
            System.out.println("push targetResultMap:" + targetResultMap);
            if (targetResultMap != null && !targetResultMap.isEmpty()) {
                // 3 判断是否有获取超速的target
                if
(targetResultMap.containsKey(PushResponseCode.RSP_SPEED_LIMIT.getValue())) {
                    // 4 获取超速的target
                    List<String> rateLimitTarget =
targetResultMap.get(PushResponseCode.RSP_SPEED_LIMIT.getValue());
                    System.out.println("rateLimitTarget is :" +
rateLimitTarget);
                    //TODO 5 业务处理，重推......
                }
            }
        } else {
            // 调用推送接口服务异常 eg: appId、appKey非法、推送消息非法.....
            // result.code(); //服务异常码
            // result.comment();//服务异常描述
            System.out.println(String.format("pushMessage error code:%s
comment:%s", result.code(), result.comment()));
        }

        //透传消息任务推送
        taskId = 123l;
        // 1 调用推送服务
        result = push.pushMessageByTaskId(PushType.DIRECT, appId, taskId,
pushIds, 0);
        if (result.isSucceed()) {
            // 2 调用推送服务成功 （其中map为设备的具体推送结果，一般业务针对超速的code类型
做处理）
            PushResult pushResult = result.value();
            String msgId = pushResult.getMsgId();//推送消息ID，用于推送流程明细排查
            Map<Integer, List<String>> targetResultMap =
pushResult.getRespTarget();//推送结果，全部推送成功，则map为empty
            System.out.println("push msgId:" + msgId);
            System.out.println("push targetResultMap:" + targetResultMap);
            if (targetResultMap != null && !targetResultMap.isEmpty()) {
                // 3 判断是否有获取超速的target
                if
(targetResultMap.containsKey(PushResponseCode.RSP_SPEED_LIMIT.getValue())) {
                    // 4 获取超速的target
                    List<String> rateLimitTarget =
targetResultMap.get(PushResponseCode.RSP_SPEED_LIMIT.getValue());
                    System.out.println("rateLimitTarget is :" +
rateLimitTarget);
                    //TODO 5 业务处理，重推......
```

```
                }
            }
        } else {
            // 调用推送接口服务异常 eg: appId、appKey非法、推送消息非法.....
            // result.code(); //服务异常码
            // result.comment();//服务异常描述

             //全部超速
             if
(String.valueOf(ErrorCode.APP_REQUEST_EXCEED_LIMIT.getValue()).equals(result.cod
e())) {
                 //TODO 5 业务处理，重推......
             }
            System.out.println(String.format("pushMessage error code:%s
comment:%s", result.code(), result.comment()));
        }
    }
```

别名消息推送（pushAliasMessageByTaskId）

- 接口说明

接口	说明
`ResultPack<PushResult> pushAliasMessageByTaskId(PushType pushType, long appId, long taskId, List<String> alias)`	任务消息推送
`ResultPack<PushResult> pushAliasMessageByTaskId(PushType pushType, long appId, long taskId, List<String> alias, int retries)`	任务消息推送
- 参数说明	

参数名称	类型	必需	默认	描述
pushType	PushType	是	null	消息类型
appId	Long	是	null	推送应用ID
taskId	Long	是	null	推送任务ID
alias	List	是	null	推送目标别名，一批最多不能超过1000个
retries	int	否	0	超时or异常重试次数

- 返回值

```
PushResult

msgId;  推送消息ID，用于推送流程明细排查
respTarget;  推送目标结果状态(key：推送响应码  value：响应码对应的目标用户 )
注：只返回不合法、超速以及推送失败的目标用户，业务一般对超速的pushId对处理
```

- 示例

```
     **
     * 别名任务消息推送
     *
     * @throws IOException
     */
    @Test
    public void testPushAliasPyTaskId() throws IOException {
        //推送对象
        IFlymePush push = new IFlymePush(APP_SECRET_KEY);

        //目标用户
        List<String> alias = new ArrayList<String>();
        alias.add("alias123");
        alias.add("Android654");

        //通知栏任务消息推送
        Long taskId = 45361L;
        // 1 调用推送服务
        ResultPack<PushResult> result =
    push.pushAliasMessageByTaskId(PushType.STATUSBAR, appId, taskId, alias);
            if (result.isSucceed()) {
                // 2 调用推送服务成功 （其中map为设备的具体推送结果，一般业务针对超速的code类型
    做处理）
                PushResult pushResult = result.value();
                String msgId = pushResult.getMsgId();//推送消息ID，用于推送流程明细排查
                Map<Integer, List<String>> targetResultMap =
    pushResult.getRespTarget();//推送结果，全部推送成功，则map为empty
                System.out.println("push msgId:" + msgId);
                System.out.println("push targetResultMap:" + targetResultMap);
                if (targetResultMap != null && !targetResultMap.isEmpty()) {
                    // 3 判断是否有获取超速的target
                    if
    (targetResultMap.containsKey(PushResponseCode.RSP_SPEED_LIMIT.getValue())) {
                        // 4 获取超速的target
                        List<String> rateLimitTarget =
    targetResultMap.get(PushResponseCode.RSP_SPEED_LIMIT.getValue());
                        System.out.println("rateLimitTarget is :" +
    rateLimitTarget);
                        //TODO 5 业务处理，重推......
                    }
                }
            } else {
                // 调用推送接口服务异常 eg: appId、appKey非法、推送消息非法.....
                // result.code(); //服务异常码
                // result.comment();//服务异常描述

                 //全部超速
                 if
    (String.valueOf(ErrorCode.APP_REQUEST_EXCEED_LIMIT.getValue()).equals(result.cod
    e())) {
                     //TODO 5 业务处理，重推......
                 }

                System.out.println(String.format("pushMessage error code:%s
    comment:%s", result.code(), result.comment()));
            }

        //透传消息任务推送
```

```
    taskId = 45407L;
    // 1 调用推送服务
    result = push.pushAliasMessageByTaskId(PushType.DIRECT, appId, taskId,
alias);
        if (result.isSucceed()) {
            // 2 调用推送服务成功 （其中map为设备的具体推送结果，一般业务针对超速的code类型
做处理）
            PushResult pushResult = result.value();
            String msgId = pushResult.getMsgId();//推送消息ID，用于推送流程明细排查
            Map<Integer, List<String>> targetResultMap =
pushResult.getRespTarget();//推送结果，全部推送成功，则map为empty
            System.out.println("push msgId:" + msgId);
            System.out.println("push targetResultMap:" + targetResultMap);
            if (targetResultMap != null && !targetResultMap.isEmpty()) {
                // 3 判断是否有获取超速的target
                if
(targetResultMap.containsKey(PushResponseCode.RSP_SPEED_LIMIT.getValue())) {
                    // 4 获取超速的target
                    List<String> rateLimitTarget =
targetResultMap.get(PushResponseCode.RSP_SPEED_LIMIT.getValue());
                    System.out.println("rateLimitTarget is :" +
rateLimitTarget);
                    //TODO 5 业务处理，重推......
                }
            }
        } else {
            // 调用推送接口服务异常 eg: appId、appKey非法、推送消息非法.....
            // result.code(); //服务异常码
            // result.comment();//服务异常描述

             //全部超速
             if
(String.valueOf(ErrorCode.APP_REQUEST_EXCEED_LIMIT.getValue()).equals(result.cod
e())) {
                 //TODO 5 业务处理，重推......
             }
            System.out.println(String.format("pushMessage error code:%s
comment:%s", result.code(), result.comment()));
        }
}
```

应用全部推送(pushToApp)

- 接口说明

接口	说明
`ResultPack<Long> pushToApp(PushType pushType, Message message)`	应用全部推送

- 参数说明

参数名称	类型	必需	默认	描述
pushType	PushType	是	null	消息类型
message	Message	是	null	消息体

- 返回值

```
Long  任务ID
```

- 示例

```
    /**
     * 应用全部推送(pushToApp)
     * @throws IOException
     */
    @Test
    public void testPushToApp() throws IOException {
        //推送对象
        IFlymePush push = new IFlymePush(APP_SECRET_KEY);

        //通知栏全部消息推送
        VarnishedMessage message = new VarnishedMessage.Builder().appId(appId)
                .title("java Sdk 全部推送标题").content("java Sdk 全部推送内容")
                .noticeExpandType(1)
                .noticeExpandContent("展开文本内容")

.clickType(2).url("http://www.baidu.com").parameters(JSON.parseObject("
{\"k1\":\"value1\",\"k2\":0,\"k3\":\"value3\"}"))
                .offLine(true).validTime(12)

.suspend(true).clearNoticeBar(true).vibrate(false).lights(false).sound(false)
                .fixSpeed(true).fixSpeedRate(30)
                .pushTimeType(1)
                .startTime(new Date())
                .build();
        ResultPack<Long> result = push.pushToApp(PushType.STATUSBAR, message);
        System.out.println(result);

        //透传全部推送
        UnVarnishedMessage message2 = new UnVarnishedMessage.Builder()
                .appId(appId)
                .title("Java SDK 全部推送标题")
                .content("Java Sdk全部推送内容")
                .isOffLine(true)
                .validTime(10)
                .pushTimeType(1)
                .startTime(new Date())
                .build();
        result = push.pushToApp(PushType.DIRECT, message2);
        System.out.println(result);
    }
```

应用标签推送(pushToTag)

- 接口说明

接口	说明
`ResultPack<Long> pushToTag(PushType pushType, Message message, List<String> tagName, ScopeType scopeType)`	应用标签推送

- 参数说明

参数名称	类型	必需	默认	描述
pushType	PushType	是	null	消息类型
message	Message	是	null	消息体
tagName	List	是	null	推送标签
scopeType	ScopeType	是	null	标签集合类型

- 返回值

```
Long  任务ID
```

- 示例

```
/**
 * 标签推送(pushToTag)
 *
 * @throws IOException
 */
@Test
public void testPushToTag() throws IOException {
    //推送对象
    IFlymePush push = new IFlymePush(APP_SECRET_KEY);

    //推送标签
    List<String> tagName = new ArrayList<String>();
    tagName.add("news");
    tagName.add("tech");

    //通知栏标签推送
    VarnishedMessage varnishedMessage = new
VarnishedMessage.Builder().appId(appId)
            .title("java Sdk 标签推送标题").content("java Sdk 标签推送内容")
            .noticeExpandType(1)
            .noticeExpandContent("展开文本内容")
            .offLine(true).validTime(12)

.suspend(true).clearNoticeBar(true).vibrate(false).lights(false).sound(false)
            .fixSpeed(true).fixSpeedRate(30)
            .pushTimeType(1)
            .startTime(new Date())
            .build();
    ResultPack<Long> result = push.pushToTag(PushType.STATUSBAR,
varnishedMessage, tagName, ScopeType.INTERSECTION);
    System.out.println(result);

    //透传标签推送
    UnVarnishedMessage unVarnishedMessage = new UnVarnishedMessage.Builder()
            .appId(appId)
            .title("Java SDK 标签推送标题")
            .content("Java Sdk标签推送内容")
            .isOffLine(true)
```

```
            .validTime(10)
            .pushTimeType(1)
            .startTime(new Date())
            .build();
    result = push.pushToTag(PushType.DIRECT, unVarnishedMessage, tagName,
ScopeType.UNION);
    System.out.println(result);
}
```

取消推送任务(cancelTaskPush)

- 接口说明

接口	说明
`ResultPack<Boolean> cancelTaskPush(PushType pushType, long appId, long taskId)`	只针对全部用户推送以及标签推送且推送状态为待推送或者推送中的任务取消

- 参数说明

参数名称	类型	必需	默认	描述
pushType	PushType	是	null	消息类型
appId	Long	是	null	应用ID
taskId	Long	是	null	任务ID

- 返回值

```
Boolean  true:成功  false：失败
```

- 示例

```
    /**
     * 取消推送任务(cancelTaskPush) 只针对全网推送生效
     * @throws IOException
     */
    @Test
    public void testCancelTaskPush() throws IOException {
        //推送对象
        IFlymePush push = new IFlymePush(APP_SECRET_KEY);
        long taskId = 1231;
        ResultPack resultPack = push.cancelTaskPush(PushType.STATUSBAR, appId,
taskId);
        System.out.println(resultPack);
    }
```

推送统计

获取任务推送统计(getTaskStatistics)

- 接口说明

接口	说明
`public ResultPack<TaskStatistics> getTaskStatistics(long appId, long taskId)`	获取推送统计

- 参数说明

参数名称	类型	必需	默认	描述
appId	Long	是	null	应用ID
taskId	Long	是	null	任务ID

- 返回值

```
TaskStatistics
```

- 示例

```
    /**
     * 获取任务统计结果
     *
     * @throws IOException
     */
    @Test
    public void testGetTaskStatistics() throws IOException {
        //推送对象
        IFlymePush push = new IFlymePush(APP_SECRET_KEY);
        long taskId = 44760L;
        ResultPack<TaskStatistics> resultPack = push.getTaskStatistics(appId,
taskId);
        System.out.println(resultPack);
    }
```

获取应用推送统计(dailyPushStatics)

- 接口说明

接口	说明
`public ResultPack<List<DailyPushStatics>> dailyPushStatics(long appId, Date startTime, Date endTime)`	获取应用推送统计

- 参数说明

参数名称	类型	必需	默认	描述
appId	Long	是	null	应用ID
startTime	Date	是	null	开始日期
endTime	Date	是	null	结束日期

- 返回值

```
List<DailyPushStatics>
```

- 示例

```
    /**
     * 获取任务统计结果
     *
     * @throws IOException
     */
     @Test
     public void testDailyPushStatics() throws IOException {
         //推送对象
         IFlymePush push = new IFlymePush(APP_SECRET_KEY);
         Date startTime = DateUtils.str2Date("2017-06-03", "yyyy-MM-dd");
         Date endTime = DateUtils.str2Date("2017-06-10", "yyyy-MM-dd");
         ResultPack<List<DailyPushStatics>> resultPack =
push.dailyPushStatics(appId, startTime, endTime);
         System.out.println(resultPack);
     }
```

高级功能

消息送达与回执

- 支持回执接口

  pushId通知栏消息推送(pushMessage)

  别名通知栏消息推送(pushMessageByAlias)
- 开发者通过设置通知栏消息extra来指定消息的送达和点击回执规则
- 回执地址请登录推送平台【配置管理】->【回执管理】注册回执地址

key	value含义
callback	回执接口（第三方接收回执的Http接口, 最大长度128字节）
callback.param	回执参数（第三方自定义回执参数, 最大长度64字节）
callback.type	回执类型（(1-送达回执, 2-点击回执, 3-送达与点击回执), 默认3）

魅族推送服务器每隔1s将已送达或已点击的消息ID和对应设备的pushId或alias通过调用开发者http接口传给开发者(每次调用后，魅族推送服务器会清空这些数据,下次传给业务方将是新一拨数据)

注:

消息的送达回执只支持向pushId或alias发送的消息

单个应用注册不同回执地址累计上限不能超过100个

- 回执响应内容

key	value
cb	回执明细内容 如下所述（Json数据）
access_token	回执接口访问令牌（推送平台设置回执地址令牌）

回执明细格式说明: 外层key代表相应的消息id和回执类型（msgId-type），value是一个JSONObject，包含了下面的参数值

param： 业务上传的自定义参数值
type： callback类型
targets： 一批alias或者pushId集合

```
{
    "msgId2-1": {
        "param": "param2",
        "type": 1,
        "targets": [
            "pushId3",
            "pushId2",
            "pushId1"
        ]
    },
    "msgId1-2": {
        "param": "param1",
        "type": 2,
        "targets": [
            "alias2",
            "alias",
            "alias1"
        ]
    }
}
```

- 抓包示例

```
Hypertext Transfer Protocol

    POST /callbackUrl/callback HTTP/1.1\r\n
        [Expert Info (Chat/Sequence): POST /callbackUrl/callback HTTP/1.1\r\n]
        Request Method: POST
        Request URI: /callbackUrl/callback
        Request Version: HTTP/1.1
```

```
    Content-Length: 32\r\n
        [Content length: 32]

    Content-Type: application/x-www-form-urlencoded; charset=UTF-8\r\n
    Host: xxx.xxx.xxx\r\n
    Connection: Keep-Alive\r\n
    User-Agent: Apache-HttpClient/4.3.4 (java 1.5)\r\n
    Accept-Encoding: gzip,deflate\r\n
    \r\n
    [Full request URI: http://xxx.xxx.xxx/callbackUrl/callback]
    [HTTP request 1/1]
    [Response in frame: 7253]
    File Data: 32 bytes

HTML Form URL Encoded: application/x-www-form-urlencoded
    Form item: "access_token" = "token"
        Key: access_token
        Value: token
    Form item: "cb" = "json value"
        Key: cb
        Value: json value
```

- 示例

```
    @Test
    public void testPushIDCallback() throws Exception {
        //推送对象
        IFlymePush push = new IFlymePush(APP_SECRET_KEY);

        //组装消息
        VarnishedMessage message = new VarnishedMessage.Builder().appId(appId)
                .title("Java SDK 推送标题").content("Java SDK 推送内容")
                .extra(ExtraParam.CALLBACK.getKey(), callbackURL)
                .extra(ExtraParam.CALLBACK_PARAM.getKey(), "param")
                .extra(ExtraParam.CALLBACK_TYPE.getKey(),
CallBackType.RECEIVE.getKey())
                .build();

        //目标用户
        List<String> pushIds = new ArrayList<String>();
        pushIds.add("pushId_1");
        pushIds.add("pushId_2");

        // 1 调用推送服务
        ResultPack<PushResult> result = push.pushMessage(message, pushIds);
        handleResult(result);
    }

    @Test
    public void testAliasCallback() throws Exception {
        //推送对象
        IFlymePush push = new IFlymePush(APP_SECRET_KEY);

        //组装消息
        VarnishedMessage message = new VarnishedMessage.Builder().appId(appId)
                .title("Java SDK 推送标题").content("Java SDK 推送内容")
                .extra(ExtraParam.CALLBACK.getKey(), callbackURL)
```

```
                .extra(ExtraParam.CALLBACK_PARAM.getKey(), "param")
                .extra(ExtraParam.CALLBACK_TYPE.getKey(),
CallBackType.RECEIVE.getKey())
                .build();

        //目标用户
        List<String> alias = new ArrayList<String>();
        alias.add("alias_1");
        alias.add("alias_2");

        // 1 调用推送服务
        ResultPack<PushResult> result = push.pushMessageByAlias(message, alias);
        handleResult(result);
    }
```

订阅服务

修改通知栏订阅开关状态 (updateStatusbarSwitch)

- 接口说明

接口	说明
`ResultPack<SwitchStatusInfo> updateStatusbarSwitch(String pushId, Boolean subSwitch)`	修改通知栏订阅开关状态

- 参数说明

参数名称	类型	必需	默认	描述
pushId	String	是	null	订阅pushId
subSwitch	Boolean	是	null	开关状态 true:打开 false:关闭

- 返回值

```
SwitchStatusInfo
```

- 示例

```
   @Test
    public void updateStatusbarSwitch() throws Exception {
        ResultPack<SwitchStatusInfo> resultPack =
sub.updateStatusbarSwitch(pushId, true);
        System.out.println("updateStatusbarSwitch:" + resultPack);
    }
```

修改透传订阅开关状态 (updateDirectSwitch)

- 接口说明

接口	说明
`ResultPack<SwitchStatusInfo> updateDirectSwitch(String pushId, Boolean subSwitch)`	修改透传订阅开关状态

- 参数说明

参数名称	类型	必需	默认	描述
pushId	String	是	null	订阅pushId
subSwitch	Boolean	是	null	开关状态 true:打开 false:关闭

- 返回值

```
SwitchStatusInfo
```

- 示例

```
    @Test
    public void updateDirectSwitch() throws Exception {
        ResultPack<SwitchStatusInfo> resultPack = sub.updateDirectSwitch(pushId,
false);
        System.out.println("updateDirectSwitch:" + resultPack);
    }
```

同步修改所有开关状态 (updateAllSwitch)

- 接口说明

接口	说明
`ResultPack<SwitchStatusInfo> updateAllSwitch(String pushId, Boolean subSwitch)`	同步修改所有开关状态

- 参数说明

参数名称	类型	必需	默认	描述
pushId	String	是	null	订阅pushId
subSwitch	Boolean	是	null	开关状态 true:打开 false:关闭

- 返回值

```
SwitchStatusInfo
```

- 示例

```
    @Test
    public void updateAllSwitch() throws Exception {
        ResultPack<SwitchStatusInfo> resultPack = sub.updateAllSwitch(pushId,
true);
        System.out.println("updateAllSwitch:" + resultPack);
    }
```

获取订阅开关状态 (updateAllSwitch)

- 接口说明

接口	说明
`ResultPack<SwitchStatusInfo> getRegisterSwitch(String pushId)`	获取订阅开关状态

- 参数说明

参数名称	类型	必需	默认	描述
pushId	String	是	null	订阅pushId

- 返回值

```
SwitchStatusInfo
```

- 示例

```
    @Test
    public void getRegisterSwitch() throws Exception {
        ResultPack<SwitchStatusInfo> resultPack = sub.getRegisterSwitch(pushId);
        System.out.println("getRegisterSwitch:" + resultPack);
    }
```

别名订阅 (subscribeAlias)

- 接口说明

接口	说明
`ResultPack<AliasInfo> subscribeAlias(String pushId, String alias)`	别名订阅

- 参数说明

参数名称	类型	必需	默认	描述
pushId	String	是	null	订阅pushId
alias	String	是	null	订阅别名 60字符限制

- 返回值

```
AliasInfo
```

- 示例

```
    @Test
    public void subscribeAlias() throws Exception {
        ResultPack<AliasInfo> resultPack = sub.subscribeAlias(pushId, "flyme");
        System.out.println("subscribeAlias:" + resultPack);
    }
```

取消别名订阅 (unSubscribeAlias)

- 接口说明

接口	说明
`ResultPack<AliasInfo> unSubscribeAlias(String pushId)`	取消别名订阅

- 参数说明

参数名称	类型	必需	默认	描述
pushId	String	是	null	订阅pushId

- 返回值

```
AliasInfo
```

- 示例

```
    @Test
    public void unSubscribeAlias() throws Exception {
        ResultPack<AliasInfo> resultPack = sub.unSubscribeAlias(pushId);
        System.out.println("unSubscribeAlias:" + resultPack);
    }
```

获取订阅别名 (getSubAlias)

- 接口说明

接口	说明
`ResultPack<AliasInfo> getSubAlias(String pushId)`	获取订阅别名

- 参数说明

参数名称	类型	必需	默认	描述
pushId	String	是	null	订阅pushId

- 返回值

```
AliasInfo
```

- 示例

```
    @Test
    public void getSubAlias() throws Exception {
        ResultPack<AliasInfo> resultPack = sub.getSubAlias(pushId);
        System.out.println("getSubAlias:" + resultPack);
    }
```

标签订阅 (subscribeTags)

- 接口说明

接口	说明
`ResultPack<TagInfo> subscribeTags(String pushId, List<String> tags)`	标签订阅

- 参数说明

参数名称	类型	必需	默认	描述
pushId	String	是	null	订阅pushId
tags	List	是	null	订阅标签（单个标签20个字符限制，100个标签数量限制）

- 返回值

```
TagInfo
```

- 示例

```
    @Test
    public void subscribeTags() throws Exception {
        List<String> tags = new ArrayList<String>(2);
        tags.add("tag1");
        tags.add("tag2");
        ResultPack<TagInfo> resultPack = sub.subscribeTags(pushId, tags);
        System.out.println("subscribeTags:" + resultPack);
    }
```

取消标签订阅 (unSubscribeTags)

- 接口说明

接口	说明
`ResultPack<TagInfo> unSubscribeTags(String pushId, List<String> tags)`	取消标签订阅

- 参数说明

参数名称	类型	必需	默认	描述
pushId	String	是	null	订阅pushId
tags	List	是	null	订阅标签（单个标签20个字符限制，100个标签数量限制）

- 返回值

```
TagInfo
```

- 示例

```
    @Test
    public void unSubscribeTags() throws Exception {
        List<String> tags = new ArrayList<String>(2);
        tags.add("tag1");
        tags.add("tag2");
        ResultPack<TagInfo> resultPack = sub.unSubscribeTags(pushId, tags);
        System.out.println("unSubscribeTags:" + resultPack);
    }
```

获取订阅标签 (getSubTags)

- 接口说明

接口	说明
`ResultPack<TagInfo> getSubTags(String pushId)`	获取订阅标签

- 参数说明

参数名称	类型	必需	默认	描述
pushId	String	是	null	订阅pushId

- 返回值

```
TagInfo
```

- 示例

```
    @Test
    public void getSubTags() throws Exception {
        ResultPack<TagInfo> resultPack = sub.getSubTags(pushId);
        System.out.println("getSubTags:" + resultPack);
    }
```

取消订阅所有标签 (unSubAllTags)

- 接口说明

接口	说明
`ResultPack<Boolean> unSubAllTags(String pushId)`	取消订阅所有标签

- 参数说明

参数名称	类型	必需	默认	描述
pushId	String	是	null	订阅pushId

- 返回值

```
Boolean
```

- 示例

```
@Test
 public void unSubAllTags() throws Exception {
     ResultPack<Boolean> resultPack = sub.unSubAllTags(pushId);
     System.out.println("unSubAllTags:" + resultPack);
 }
```