信鸽&IM调研

| 日期 | 内容 | 版本 | 编写人 |
| --- | --- | --- | --- |
| 2016－7-1 | 创建 | 1.0 | guozhifeng（郭智丰） |
| 2016－7-1 | 1，信鸽sdk简介  2，信鸽适用场景  3，信鸽sdk完整接入流程  4，IM SDK 接入 |  | guozhifeng（郭智丰） |
|  |  |  |  |
|  |  |  |  |

信鸽sdk简介

推送场景定义

通知定义为Android和iOS开发者指南中的Notification

1，服务器定向将信息实时送达手机

2，本地通知定义为Android开发者指南中的Local Notification。

应用通过自定义的日期、时间和消息内容，无需通过服务器即可向用户推送一条可见的信息。

消息命令

消息命令可以支持的场景非常广泛，可以任由开发人员扩展。例如静默增量更新。

标签

在信鸽产品中，标签通常是指给某个一群用户打上标签，一个应用最多有10000个 标签（tag）， 每个token在一个应用下最多100个 标签（tag）， 标签（tag）中不准包含空格。

帐号

别名/帐号通常是指给某特定用户推送消息。别名/帐号可以是终端在注册时上报的QQ号、openid、邮箱帐号、手机号等 ，必须首先将账号与token进行绑定，否则将无法推送成功，目前一个帐号下，最多绑定15个设备token，超出后会随机顶掉前面绑定的一个token

信鸽适用场景

信鸽可以保证到答，结合到医学大帮中的功能有：

1，开平广告推送

2，系统广播

3，离线消息通知

4，活动开始提醒

5，灰度测试

信鸽sdk完整接入流程

1，启动与注册

配置AccessId和AccessKey。

调用注册接口 获取 token

XGPushManager.registerPush(this, new XGIOperateCallback() {

@Override

public void onSuccess(Object data, int flag) {

Log.d("TPush", "注册成功，设备token为：" + data);

}

@Override

public void onFail(Object data, int errCode, String msg) {

Log.d("TPush", "注册失败，错误码：" + errCode + ",错误信息：" + msg);

}

})

2，绑定账号注册

XGPushManager.registerPush(this, "UserAccount",

new XGIOperateCallback() {

@Override

public void onSuccess(Object data, int flag) {

Log.d("TPush", "注册成功，设备token为：" + data);

}

@Override

public void onFail(Object data, int errCode, String msg) {

Log.d("TPush", "注册失败，错误码：" + errCode + ",错误信息：" + msg);

}

});

一个账号可能在多个设备登陆 注册APP，这样可以通过后台向指定的账号发送推送消息，有2个版本的API接口。

注意：这里的帐号可以是邮箱、QQ号、手机号、用户名等任意类别的业务帐号。目前一个帐号可以绑定最多15个设备token，超限后最新的会随机顶掉之前绑定的一个。

3，账号解绑

registerPush(context, "\*")或registerPush(context, "\*", xGIOperateCallback )

4，获取注册结果

1. 使用callback版本的注册接口

/\*\*

\* 操作回调接口

\*/

public interface XGIOperateCallback {

/\*\*

\* 操作成功时的回调。

\* @param data 操作成功的业务数据，如注册成功时的token信息等。

\* @param flag 标记码

\*/

public void onSuccess(Object data, int flag);

/\*\*

\* 操作失败时的回调

\* @param data 操作失败的业务数据

\* @param errCode 错误码

\* @param msg 错误信息

\*/

public void onFail(Object data, int errCode, String msg);

}

2. 重载XGPushBaseReceiver

/\*\*

\* 注册结果

\*

\* @param context

\* APP上下文对象

\* @param errorCode

\* 错误码，{@link XGPushBaseReceiver#SUCCESS}表示成功，其它表示失败

\* @param registerMessage

\* 注册结果返回

\*/

@Override

5，反注册

public static void unregisterPush(Context context)

反注册结果

可通过重载XGPushBaseReceiver的onUnregisterResult方法获取。

6，通知和消息

<!-- APP实现的Receiver，用于接收消息和结果反馈 -->

<!-- com.tencent.android.xgpushdemo.CustomPushReceiver需要改为自己的Receiver -->

<receiver android:name="com.tencent.xgpushdemo.CustomPushReceiver" >

<intent-filter>

<!-- 接收消息透传 -->

<action android:name="com.tencent.android.tpush.action.PUSH\_MESSAGE" />

<!-- 监听注册、反注册、设置/删除标签、通知被点击等处理结果 -->

<action android:name="com.tencent.android.tpush.action.FEEDBACK" />

</intent-filter>

</receiver>

推送通知（展现在通知栏）

透传消息命令（可自定义展示任意位置）

IM SDK 接入

1，初始化

1 获取通讯管理器

public static TIMManager getInstance()

2 新消息通知

public void addMessageListener(TIMMessageListener listener)

public boolean onNewMessages(java.util.Listmsgs)

public void removeMessageListener(TIMMessageListener listener)

3 网络事件通知

public void setConnectionListener(TIMConnListener listener)

4 日志事件

public void setLogListener(TIMLogListener listener)

5 用户状态变更

目前允许多平台同时介入（不会被踢下线）

public void setUserStatusListener(TIMUserStatusListener userStatusListener)

6 Crash上报

TIMManager.getInstance().disableCrashReport();

7 设置日志级别

public void setLogLevel(TIMLogLevel level)

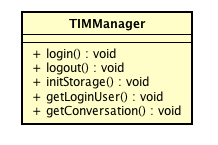
8 控制台不打印日志或修改日志路径

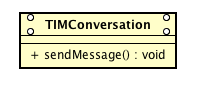
9 禁用存储

public void disableStorage()

10 通讯管理器初始化

TIMManager.getInstance().init(context);





2，登录

1 登录

// accountType 和 sdkAppId 通讯云管理平台分配

// identifier为用户名，userSig 为用户登录凭证

// appidAt3rd 在私有帐号情况下，填写与sdkAppId 一样

TIMUser user = new TIMUser();

user.setAccountType(accountType);

user.setAppIdAt3rd(appIdAt3rd);

user.setIdentifier(identifier);

public void login(int sdkAppId,

TIMUser user,

java.lang.String userSig,

TIMCallBack callback)

2 登出

public void logout(TIMCallBack callback)

3 无网络情况下查看消息

public void initStorage(int sdkAppId,

TIMUser user,

java.lang.String userSig,

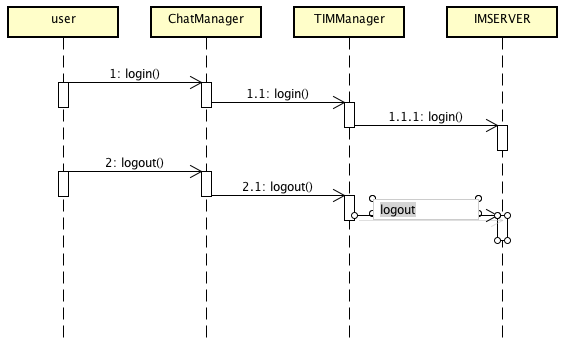
TIMCallBack cb)

4 获取当前登录用户

public String getLoginUser()

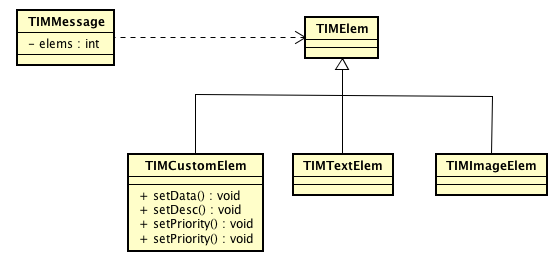
5 ImSDK同步离线消息

public void setRefreshListener(TIMRefreshListener listener)



3，自定义消息收发

1发送



TIMMessage msg = new TIMMessage();

// xml协议的自定义消息

String sampleXml = "<!--?xml version='1.0' encoding="utf-8"?-->testTitlethis is custom msgtest msg body";

//向TIMMessage中添加自定义内容

TIMCustomElem elem ＝ new TIMCustomElem();

elem.setData(sampleXml.getBytes()); //自定义byte[]

elem.setDesc("this is one custom message"); //自定义描述信息

//将elem添加到消息

if(msg.addElement(elem) != 0) {

Log.d(tag, "addElement failed");

return;

}

//发送消息

conversation.sendMessage(msg, new TIMValueCallBack<TIMMessage>() {//发送消息回调

@Override

public void onError(int code, String desc) {//发送消息失败

//错误码code和错误描述desc，可用于定位请求失败原因

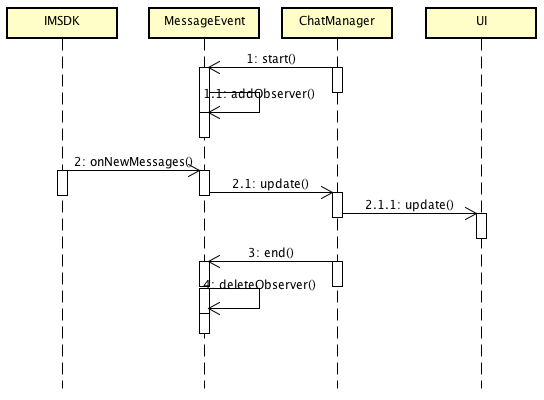
//错误码code含义请参见错误码表

Log.d(tag, "send message failed. code: " + code + " errmsg: " + desc);

}

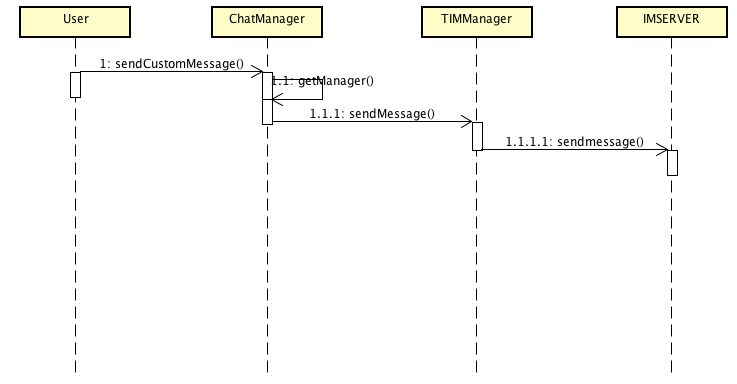
@Override

public void onSuccess(TIMMessage msg) {//发送消息成功

Log.e(tag, "SendMsg ok");

}

});



2接收

public void addMessageListener(TIMMessageListener listener)

public boolean onNewMessages(java.util.Listmsgs)

public void removeMessageListener(TIMMessageListener listener)

4，未读消息计数（Android SDK）

1，获取未读消息数

TIMConversation

conversation.getUnreadMessageNum();

2，将此消息之前的所有消息标记为已读

public void setReadMessage(TIMMessage msg)

3，将所有消息标记为已读

public void setReadMessage()

4， 禁用自动上报

public void disableAutoReport()

5，群组管理

1,创建群

public void createGroup(java.lang.String type,

java.util.List<java.lang.String> members,

java.lang.String groupName,

TIMValueCallBack<java.lang.String> cb)

type 群类型: Private 私有群，Public 公开群，ChatRoom 聊天室，AVChatRoom 直播大群

members 待加入群组的成员列表，创建者默认加入，无需指定（群内最多10000

groupName 群组名称（最长30字节）

cb 回调，OnSuccess函数的参数中将返回创建成功的群组Id

之前mqtt中所用的topic区分 群组类型（直播活动，工作室群聊）现在通过自定义群组id来实现

自定义群组Id创建群组

public void createGroup(String type, List<String> members, String groupName, String groupId, TIMValueCallBack<String> cb)

2，申请加入群组

我们现有的业务不需要申请进入 群组类型等同于 私有群(具体要同服务器商量)

3， 退出群组

public void quitGroup(java.lang.String groupId,TIMCallBack cb)

6，离线推送（Android SDK）

1，注册离线推送监听器

public void setOfflinePushListener(TIMOfflinePushListener listener)

TIMManager.getInstance().setOfflinePushListener(new TIMOfflinePushListener() {

@Override

public void handleNotification(TIMOfflinePushNotification notification) {

Log.d("MyApplication", "recv offline push");

// 这里的doNotify是ImSDK内置的通知栏提醒，应用也可以选择自己利用回调参数notification来构造自己的通知栏提醒

notification.doNotify(getApplicationContext(), R.drawable.ic\_launcher);

}

});

7，数据结构

会话类型

**1**，单聊（**c2c**）

assistant("医大帮小助手")

medbigbang("欢迎"),首次进入

hotcourseware("导师热门课件"),

activityreply("导师活动回放"

private\_chat("私聊") 老师同学生聊天

**2**，群聊

group\_chat("群聊") 工作室聊天

transfer("转诊")现在用群聊实现

activity("互动") 医探究竟 右侧 群聊

live\_activity("直播") 一探究竟左侧

video\_activity("直播讲堂"),视频直播

3，推送

system("通知")

issue("同行圈")帖子

timeline("关注圈")（关注圈，新帖子）

hot("热门") 热门，

auth("已认证") 审核状态

notice("跑马灯通知")

advertise("开屏广告")

new\_bbs("工作室讨论社区新帖子") 工作室 讨论组 新帖子

自定义消息类型：

text, //文本

image, //("图片"),

article, //("文章"), //文章详情同理 活动详情 文图Custom

homework, //("作业") ,//文章详情同理 作业详情 文图Custom

courseware, //(课件)//文章详情同理 课件详情 文图Custom

//topic, //主题，转诊 问题等

//location,

voice, //音频

video, //视频

message, //(小字提示 eg:欢迎xxx加入导师团队) 文Custom

issue, //(问答 一问 一答) 语音文字的文Custom

file, //文件

pop, //泡泡 文字文Custom

online, //在线人数 文字文Custom