Evan

Only let oneself become strong enough, good enough, can afford the life that you want to.

android极光推送初步了解

标签： android 极光推送

2016-01-04 16:34 309人阅读 评论(0) 收藏 举报

分类： 移动开发（38）

目录(?) [-]

推送可以及时,主动的与用户发起交互

(1)继承jar包,照示例AndroidManifest.xml添加.

(2)自定义MyApp继承自Application,在onCreate方法中调用JPushInterface.init(MainActivity.this);

或者在Activity的onCreate中调用.

(3)另外,在activity的onResume方法要调用JPushInterface.onResume(this);否则,推送不会出现,

在onPause中调用JPushInterface.onPause(this);

这样,可以通过服务器往安装了App的所有用户发送一条推送.

通过Alias往客户端发送信息.

在客户端的onCreate中

```
[java]
JPushInterface.setAlias(MainActivity.this, "aa", new TagAliasCallback() {

                    @Override

                    public void gotResult(int arg0, String arg1, Set<String> arg2) {

                        Log.e("info",arg1+"-----------");

                        //arg1是tag

                    }

                });
```

这句就是将"aa"当成该设备的别名,达到往指定客户端发送消息的目的.

别名和签名设置的异常处理

有时会因为网络原因,有一定几率设置别名或标签失败.

```
[java]
```

```
privatevoidsetAlias() {

EditText aliasEdit = (EditText) findViewById(R.id.et_alias);

String alias = aliasEdit.getText().toString().trim();

if(TextUtils.isEmpty(alias)) {

Toast.makeText(PushSetActivity.this,R.string.error_alias_empty, Toast.LENGTH_SHORT).show();

return;

}

if(!ExampleUtil.isValidTagAndAlias(alias)) {

Toast.makeText(PushSetActivity.this,R.string.error_tag_gs_empty, Toast.LENGTH_SHORT).show();

return;

}
```

// 调用 Handler 来异步设置别名

```
[java]
mHandler.sendMessage(mHandler.obtainMessage(MSG_SET_ALIAS, alias));
}

privatefinalTagAliasCallback mAliasCallback =newTagAliasCallback() {

@Override

publicvoidgotResult(intcode, String alias, Set<String> tags) {

String logs ;

switch(code) {

case:

logs ="Set tag and alias success";

Log.i(TAG, logs);

break;

case:

logs ="Failed to set alias and tags due to timeout. Try again after 60s.";

Log.i(TAG, logs);

mHandler.sendMessageDelayed(mHandler.obtainMessage(MSG_SET_ALIAS, alias),*);

break;

default:

logs ="Failed with errorCode = "+ code;

Log.e(TAG, logs);

}



ExampleUtil.showToast(logs, getApplicationContext());

}
};
```

```
privatestaticfinalintMSG_SET_ALIAS =;

privatefinalHandler mHandler =newHandler() {

@Override

publicvoidhandleMessage(android.os.Message msg) {

super.handleMessage(msg);

switch(msg.what) {

caseMSG_SET_ALIAS:

Log.d(TAG,"Set alias in handler.");

// 调用 JPush 接口来设置别名。

JPushInterface.setAliasAndTags(getApplicationContext(), (String) msg.obj,null, mAliasCallback);

break;

default:

Log.i(TAG,"Unhandled msg - "+ msg.what);

}

}
};
```

自定义通知栏的样式

自定义样式放在init()之后.

```
[java]

                CustomPushNotificationBuilder builder=new CustomPushNotificationBuilder(MainActivity.this, R.layout.my_push, R.id.iv

                builder.statusBarDrawable=R.drawable.ic_category_2;//最顶层状态栏小图标

                builder.layoutIconDrawable=R.drawable.ic_category_2;  //下拉状态时显示的通知图标.

                JPushInterface.setPushNotificationBuilder(2, builder);
JPushInterface.setDefaultPushNotificationBuilder(builder); //设置该对话框为默认
```

. 自定义消息:

所接收的消息不再局限于Notification,而是可以直接取出消息中的内容,从而用自己的方式显示给用户.

此时需要自定义一个MyReceiver继承自BroadcastReceiver.

```
[java]
public class MyReceiver extends BroadcastReceiver {

    @Override

    public void onReceive(Context ctx, Intent intent) {

        Bundle bundle =intent.getExtras();   //接受到消息



        Log.e("info", "[MyReceiver] onReceive - " + intent.getAction() + ", extras: " + printBundle(bundle));

          if (JPushInterface.ACTION_REGISTRATION_ID.equals(intent.getAction())) {

                String regId = bundle.getString(JPushInterface.EXTRA_REGISTRATION_ID);

```

```
                Log.d("info", "[MyReceiver] 接收Registration Id : " + regId);

                //send the Registration Id to your server...


            } else if (JPushInterface.ACTION_MESSAGE_RECEIVED.equals(intent.getAction())) {

                Log.d("info", "[MyReceiver] 接收到推送下来的自定义消息: " + bundle.getString(JPushInterface.EXTRA_MESSAGE));

//                processCustomMessage(ctx, bundle);


            } else if (JPushInterface.ACTION_NOTIFICATION_RECEIVED.equals(intent.getAction())) {

                Log.d("info", "[MyReceiver] 接收到推送下来的通知");

                int notifactionId = bundle.getInt(JPushInterface.EXTRA_NOTIFICATION_ID);

                Log.d("info", "[MyReceiver] 接收到推送下来的通知的ID: " + notifactionId);


            } else if (JPushInterface.ACTION_NOTIFICATION_OPENED.equals(intent.getAction())) {

                Log.d("info", "[MyReceiver] 用户点击打开了通知");

                JPushInterface.reportNotificationOpened(ctx, bundle.getString(JPushInterface.EXTRA_MSG_ID));


//                //打开自定义的Activity

                Intent i = new Intent(ctx, TwoActivity.class);

                i.putExtras(bundle);

                i.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

                ctx.startActivity(i);


            }


    }

    // 打印所有的 intent extra 数据

    private static String printBundle(Bundle bundle) {

        StringBuilder sb = new StringBuilder();

        for (String key : bundle.keySet()) {

            if (key.equals(JPushInterface.EXTRA_NOTIFICATION_ID)) {

                sb.append("/nkey:" + key + ", value:" + bundle.getInt(key));

            } else {

                sb.append("/nkey:" + key + ", value:" + bundle.getString(key));

            }

        }

        return sb.toString();

    }
```

在类中接收完消息后,还需要在AndroidManifest.xml中添加

<!--自定义接收 -->

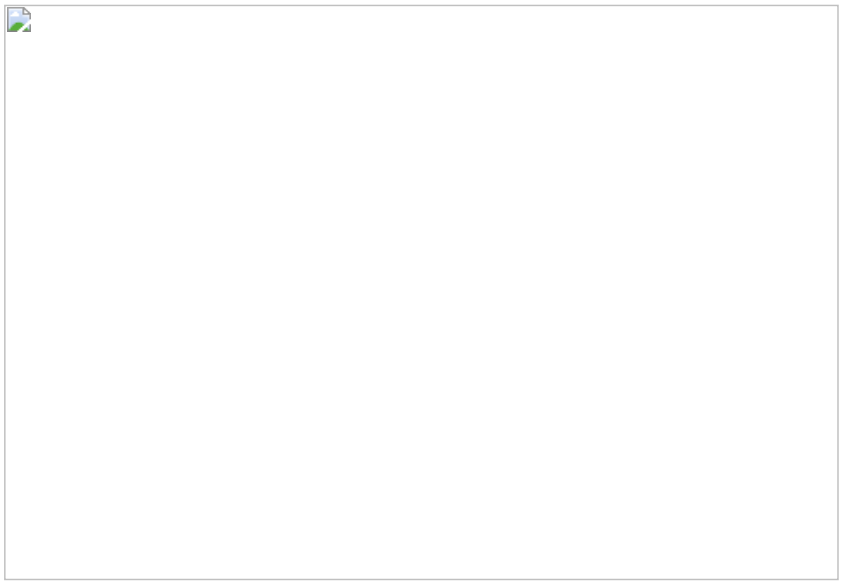
```
[html]
<receiver

    android:name="com.lj.pushdemo1.MyReceiver"

    android:enabled="true">

    <intent-filter>

        <action android:name="cn.jpush.android.intent.REGISTRATION" />

        <action android:name="cn.jpush.android.intent.MESSAGE_RECEIVED" />

        <action android:name="cn.jpush.android.intent.NOTIFICATION_RECEIVED" />

        <action android:name="cn.jpush.android.intent.NOTIFICATION_OPENED" />

        <category android:name="com.lj.pushdemo1" />

    </intent-filter>

</receiver>
```

获取 RegistrationID API

集成了JPush SDK的应用程序第一次注册到JPush服务器时,服务器会返回一个唯一的该设备的标识:RegistertionID.

String id=JPushInterface.getRegistrationID(MainActivity.this);

调用网络接口来发送消息



sendno:发送的编号.

app_key: 应用程序的appKey

receiver_type:接受者的类型 ----2.指定tag----3.指定alias----4.广播----5.根据registrationId进行推送.

msg_content:发送的内容,在这里必须要JSON格式.

platform:要发送的平台

verfication_code:将sendno+receiver_typ+receiver_values+*API MasterSecret(在应用的详细信息里面)字符串拼接起来后*

设置保留最近通知条数 API

```
[java]
01. JPushInterface.init(context);
02.
03. JPushInterface.setLatestNotificationNumber(context,);保留最近的3条
```


- 上一篇 Android实现推送方式解决方案
- 下一篇 Android数据过滤器：Filter

相关文章推荐

- Fragment的setUserVisibleHint方法实现懒加载,...
- 使用极光推送实现分组发送和服务端集成（Jpush）
- Android Studio集成极光推送
- 极光推送整合注意事项
- android系列学习：tab切换，fragment中嵌套list...
- Android Fragment 你应该知道的一切
- 使用第三方推送功能变相实现一些即时通讯操作
- Fragment之间的数据传递 三种数据传递方式的相...
- 环信集成EaseUI自定义消息拓展
- Android实现推送方式解决方案

查看评论

暂无评论