**一14一 一14一 一12一 一11一** 一10.5一 一10.5一 '一10一"

**目 录**

[1. shortcuts长按APP图标弹出快捷方式 2](#_Toc24235)

[1.1. 静态注册 2](#_Toc31255)

[1.2. 动态注册 3](#_Toc9290)

[1.3. 通过拖动固定到桌面的 Shortcuts是固定的 3](#_Toc826)

[1.4. 动态 Shortcuts 与静态 Shortcuts 区别 4](#_Toc5379)

[1.5. 其他 4](#_Toc15952)

[2. ActivityLifecycleCallbacks 4](#_Toc10013)

[2.1. 调用时机 4](#_Toc31874)

[2.2. 需求：通知消息点击的跳转决策 5](#_Toc21281)

[3. 仿微信授权SDK编写 6](#_Toc30799)

[3.1. 流程图 7](#_Toc17749)

[3.2. 云讯App中的认证界面 7](#_Toc16768)

[3.3. SDK中的代码 10](#_Toc6269)

[3.3.1. modelbase包 10](#_Toc4477)

[3.3.2. api包 12](#_Toc19270)

[3.4. 使用 13](#_Toc4994)

[3.4.1. 引入jar包，时间总线库，manifest注册YXEntryActivity 13](#_Toc28215)

[3.4.2. 在包名.yxapi文件夹中创建YXEntryActivity类 14](#_Toc18249)

[3.4.3. 在AppLication中初始化全局的YXAPI，并传入appId 15](#_Toc13871)

[3.4.4. 在LoginActivity中调用sdk向云讯App发起授权请求 15](#_Toc6683)

[3.4.5. YXEntryActivity拿到code，通过事件总线传回LoginActivity的后续操作 16](#_Toc16300)

# shortcuts长按APP图标弹出快捷方式

在android7.0上就有的新特性

## 静态注册

首先, 我们需要在res/xml目录下创建一个新的xml文件, 这里我们命名为shortcuts.xml.

<?xml version="1.0" encoding="utf-8"?>  
<shortcuts xmlns:android="http://schemas.android.com/apk/res/android">  
 <shortcut  
 android:shortcutId="settings"  
 android:enabled="true"  
 android:icon="@drawable/icon1"  
 android:shortcutShortLabel="@string/short\_name"  
 android:shortcutLongLabel="@string/long\_name"  
 android:shortcutDisabledMessage="@string/can\_not">  
  
 <intent  
 android:action="android.intent.action.VIEW"  
 android:targetPackage="com.cgy.databindingdemojava"  
 android:targetClass="com.cgy.databindingdemojava.SettingsActivity" />  
 <categories android:name="android.shortcut.conversation"/>  
 </shortcut>  
 <shortcut  
 android:shortcutId="settings2"  
 android:enabled="true"  
 android:icon="@drawable/icon2"  
 android:shortcutShortLabel="@string/short\_name"  
 android:shortcutLongLabel="@string/long\_name"  
 android:shortcutDisabledMessage="@string/can\_not">  
  
 <intent  
 android:action="android.intent.action.VIEW"  
 android:targetPackage="com.cgy.databindingdemojava"  
 android:targetClass="com.cgy.databindingdemojava.SettingsActivity" />  
 <categories android:name="android.shortcut.conversation"/>  
 </shortcut>  
</shortcuts>

以shortcuts元素为根，可以包含多个shortcut元素，每个shortcut元素表示一个 shortcut。其中属性分别表示：

1. shortcutId表示 shortcut 唯一标识符，相同的 shortcutId 会被覆盖。必选字段。
2. shortcutShortLabel为将 shortcut 拖动到桌面时显示的名字，官方建议不超过 10 个字符，必选字段。
3. shortcutLongLabel为 shortcut 列表中每个 shortcut 的名字，不宜过长，如果过长或未设置默认会显示 ShortLabel，官方建议不超过 25 个字符。可选字段。
4. icon为 shortcut 的 icon，在列表展示和拖动到桌面时显示需要，可选字段。
5. enabled表示 shortcut 是否可用，false 表示禁用。xml 中这个属性几乎没有被设置为 false 的实际场景，具体原因可见6.7 如何更好的删除(废弃)老的 Shortcut中介绍。
6. shortcutDisabledMessage为已固定在桌面的 shortcut 被 Disabled 后点击时的 Toast 提示内容。可选字段。
7. intent为点击 shortcut 时响应的 intent，必选字段。
8. 这里可以添加多个 intent，但点击时不会启动所有 intent，而是启动最后一个 intent，在这个 intent 回退时会启动它前面一个 intent，相当于自动将所有 intent 添加到了堆栈。
9. intent可设置属性包括：android:action、android:data、android:mimeType、android:targetClass、android:targetPackage，其中android:action为必选属性。
10. categories, 这个东西目前位置官方只给提供了android.shortcut.conversation

然后AndroidManifest.xml 的 action是.MAIN和category是LAUNCHER!对应的 Activity 内添加 meta-data，它的name 为android.app.shortcuts，如下：

<application

……>

<activity android:name=".main.MainActivity">

<intent-filter>

<action android:name="android.intent.action.MAIN"/>

<category android:name="android.intent.category.LAUNCHER"/>

</intent-filter>

<meta-data

android:name="android.app.shortcuts"

android:resource="@xml/shortcuts"/>

</activity>

</application>

## 动态注册

动态 ShortcutsDynamic Shortcuts 通过 ShortcutManager API 进行操作。可以动态添加、修改、删除。

if (Build.VERSION.SDK\_INT < Build.VERSION\_CODES.N\_MR1) {

return;

}

ShortcutManager mShortcutManager = getSystemService(ShortcutManager.class);  
List<ShortcutInfo> infos = new ArrayList<>();  
for (int i = 0; i < mShortcutManager.getMaxShortcutCountPerActivity(); i++) {  
 Intent intent = new Intent(this, SettingsActivity.class);  
 intent.setAction(Intent.ACTION\_VIEW);  
 intent.putExtra("msg", "我和" + strs[i] + "的对话");  
 ShortcutInfo info = new ShortcutInfo.Builder(this, "id" + i)  
 .setShortLabel(strs[i])  
 .setLongLabel("联系人:" + strs[i])  
 .setIcon(Icon.createWithResource(this, rs[i]))  
 .setIntent(intent)  
 .build();  
 infos.add(info);  
}  
mShortcutManager.setDynamicShortcuts(infos);

1. 通过ShortcutInfo.Builder新建 ShortcutInfo，再通过shortcutManager添加即可；
2. setDynamicShortcuts(List)可以替换并添加所有 shortcut 列表；
3. addDynamicShortcuts(List)可以添加新的 shortcut 到列表，超过最大个数会报异常；
4. updateShortcuts(List)可以更新一组 shortcuts；
5. removeDynamicShortcuts(List)和removeAllDynamicShortcuts() 可以删除部分或所有 shortcuts。
6. ShortcutInfo的属性与 xml 中定义字段含义一致，shortcutId shortcutShortLabel intent 是必须设置的字段，并且intent必须设置Action。

## 通过拖动固定到桌面的 Shortcuts是固定的

指通过拖动固定到桌面的 Shortcuts，App 无法添加、修改、删除这些 Shortcuts，只能禁用他们。即便 App 内删除了某个 Shorcut，对应的已固定到桌面的 Shortcuts 也不会被删除。

禁用动态的 Shortcuts

通过getPinnedShortcuts()得到所有固定的 Shortcuts 的信息。

通过disableShortcuts(List<id>)和enableShortcuts(List<id>)禁用和启用的 Shortcuts

List<ShortcutInfo> pinnedShortcuts = mShortcutManager.getPinnedShortcuts();

List<String> ids = new ArrayList<>();

for (ShortcutInfo info : pinnedShortcuts) {

ids.add(info.getId());

}

mShortcutManager.enableShortcuts(ids, "不可用了");

静态动态的 Shortcuts

需要在资源文件中设置android:enabled="false"进行禁用，不过没有必要，静态 Shortcuts 可直接通过删除达到禁用的效果。

## 动态 Shortcuts 与静态 Shortcuts 区别

1. 静态 Shortcuts 只能通过升级应用修改，动态 Shortcuts 随时可以修改；
2. 静态 Shortcuts 的 Intent 无法设置 Flag，默认为FLAG\_ACTIVITY\_NEW\_TASK和FLAG\_ACTIVITY\_CLEAR\_TASK Flag，即若应用运行中会清除所有已存在的 Activity(不推荐)。而动态 Shortcuts 的 Intent 可以设置 Flag(推荐，不设置的话，就是默认的普通Intent，会添加到当前Task后边)
3. 静态 Shortcuts 的rank系统默认根据声明顺序设置，动态 Shortcuts 的rank可以通过setRank(int rank)接口主动设置，rank 不能小于 0，值越大表示在 shortcut 列表展示时离 App Icon 越远。静态 Shortcuts 默认比动态 Shortcuts 离 App Icon 更近。
4. 静态 Shortcuts 删除可以直接删除，动态 Shortcuts 建议通过禁用删除；

## 其他

动态 Shortcuts 操作的频率问题

当应该完全退到后台(无 Activity 或 Service 在前台时)，其操作 Shortcut(包括添加、删除、修改) 的频率是受限的。可通过isRateLimitingActive()查询是否已受限，true表示已受限。

跟踪 Shorcut 使用情况

在 Shortcut 被选择或者其关联的操作被操作时需调用reportShortcutUsed(String shortcutId)接口上报数据，为了方便启动器收集应用 Shortcuts 使用情况，以便未来进行预测或者向开发者展示哪些操作适合作为 Shortcuts 以及其优先级。

PS：这个接口其实挺尴尬的，一方面需要 App 主动上报，侵入性太强。另一方面这个预测功能未来也不好加到 Shortcuts 推荐里，更多是个开发工具相关功能。

最好是由启动器自己纯粹收集 Shortcut 被选择的使用情况数据，而不需要统计 Shortcut 被关联操作通过其他方式调用的使用情况数据。至于哪些操作适合作为 Shortcuts，开发者大可通过其他监控 SDK 去判断。

应用备份

如果应用通过备份恢复到另外一台机器上，固定的 Shortcuts 是可以直接恢复的，不过启动器不保存这些 Shortcut 的 icon，所以应用内需要存在这些 icon 对应的资源以便启动器能找到。静态 Shortcuts 需要应用重新安装、升级才能生效。动态 Shortcuts 需要相应代码被执行过才能生效。

# ActivityLifecycleCallbacks

## 调用时机

Activity源码中：

@MainThread  
@CallSuper  
protected void onCreate(@Nullable Bundle savedInstanceState) {  
 if (*DEBUG\_LIFECYCLE*) Slog.v(*TAG*, "onCreate " + this + ": " + savedInstanceState);  
 ...

getApplication().dispatchActivityCreated(this, savedInstanceState);  
 ...  
}

getApplication().dispatchActivityCreated(this, savedInstanceState)将回调ActivityLifecycleCallbacks中的onActivityCreated，其他的几个回调大同小异

## 需求：通知消息点击的跳转决策

MyActivityState.java

public interface MyActivityState {  
 Activity current();//得到当前Activity  
 int count();//任务栈中Activity的总数  
 boolean isFront();//判断应用是否处于前台，即是否可见  
}

MyActivityLifecycleCallbacks.java

public class MyActivityLifecycleCallbacks implements Application.ActivityLifecycleCallbacks, MyActivityState {  
 private List<Activity> activityList = new ArrayList<>();  
 private List<Activity> resumeActivity = new ArrayList<>();  
 @Override  
 public int count() {  
 return activityList.size();  
 }  
 @Override  
 public boolean isFront() {  
 return resumeActivity.size() > 0;  
 }  
 @Override  
 public Activity current() {  
 return activityList.size() > 0 ? activityList.get(0) : null;  
 }  
 @Override  
 public void onActivityCreated(Activity activity, Bundle savedInstanceState) {  
 activityList.add(activity);  
 }  
 @Override  
 public void onActivityStarted(Activity activity) {  
 }  
 @Override  
 public void onActivityResumed(Activity activity) {  
 if (!resumeActivity.contains(activity)) {  
 resumeActivity.add(activity);  
 }  
 }  
 @Override  
 public void onActivityPaused(Activity activity) {  
 }  
 @Override  
 public void onActivityStopped(Activity activity) {  
 resumeActivity.remove(activity);  
 }  
 @Override  
 public void onActivitySaveInstanceState(Activity activity, Bundle outState) {  
 }  
 @Override  
 public void onActivityDestroyed(Activity activity) {  
 activityList.remove(activity);  
 }  
}

App.java

public class App extends Application {  
 private static MyActivityLifecycleCallbacks *mActivityLifecycleCallbacks*;  
 @Override  
 public void onCreate() {  
 super.onCreate();  
 *mActivityLifecycleCallbacks* = new MyActivityLifecycleCallbacks();

registerActivityLifecycleCallbacks(*mActivityLifecycleCallbacks*);

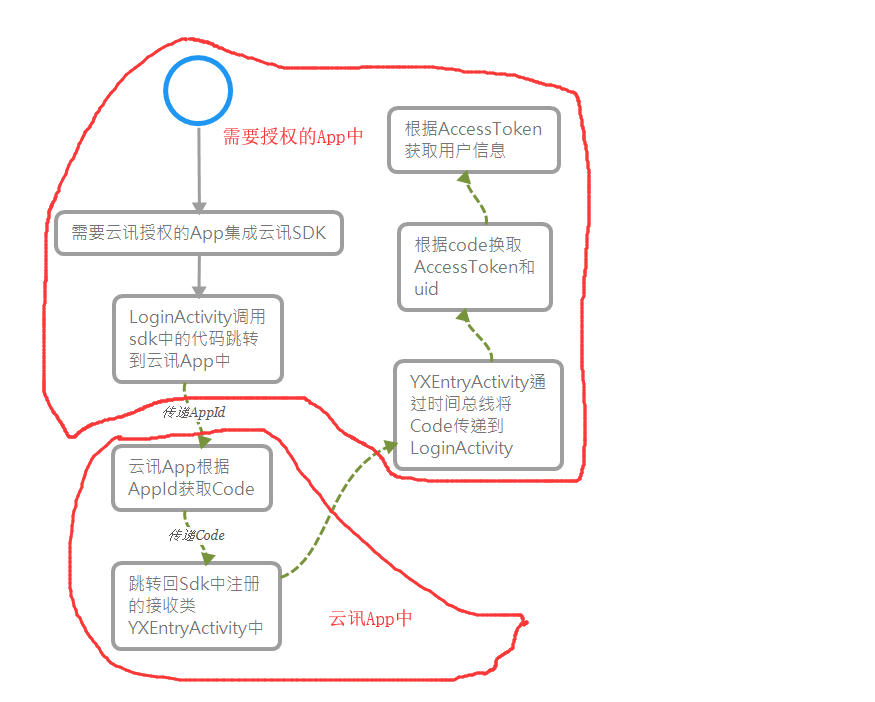
}  
 public static MyActivityLifecycleCallbacks getmActivityLifecycleCallbacks() {  
 return *mActivityLifecycleCallbacks*;  
 }  
}

JPushReceiver.java

public class JPushReceiver extends BroadcastReceiver {  
 private Context context;  
 private MyActivityLifecycleCallbacks myActivityLifecycleCallbacks;  
 @Override  
 public void onReceive(Context context, Intent intent) {  
 this.context = context;  
 myActivityLifecycleCallbacks = App.*getmActivityLifecycleCallbacks*();  
 // 如果应用处于前台做某些操作（一般此时，收到通知和点击通知栏做相同的操作）  
 if (myActivityLifecycleCallbacks.isFront()) {  
 if (myActivityLifecycleCallbacks.current() instanceof SomeActivity) {  
 // 如果当前Activity是要跳转的Activity，做某些操作，一般不跳转，可用EventBus  
 } else {  
 // 否则执行跳转或其他相关操作等  
 }  
 } else if ("cn.jpush.android.intent.NOTIFICATION\_OPENED".equals(intent.getAction())) {  
 // 应用处于不可见状态，一般点击通知栏是跳转到某个页面  
 }  
 }  
}

# 仿微信授权SDK编写

## 流程图



## 云讯App中的认证界面

sdk跳转的目标界面，sdk传递appId，用于获取Code

class AuthorActivity : BaseActivity() {  
 private var appId = ""//需要授权的app的Appid  
 private var state = ""//需要授权的app的原样返回字段  
 private var code = ""//需要授权的app的code  
 private lateinit var backIntent: Intent//返回需要授权的app的intent  
 override fun getLayoutId() = R.layout.*activity\_author* override fun initView() {  
 appId = *intent*.getStringExtra(*REQ\_APPID*)  
 val scope = *intent*.getStringExtra(*REQ\_SCOPE*)  
 val backPackage = *intent*.getStringExtra(*REQ\_PACKAGE*)  
 state = *intent*.getStringExtra(REQ\_STATE)  
 //获取请求授权app信息  
 getOtherAppInfoByAppId()  
 //为第三方获取授权码  
 getAuthorCode()  
 //初始化返回需要授权的app的intent  
 backIntent = Intent().*apply* **{** setClassName(backPackage, "$backPackage$classNameSuffix")  
 putExtra(*RESP\_SCOPE*, scope)  
 **}** //同意授权按钮  
 tv\_author.setOnClickListener **{** backIntent.*apply* **{** putExtra(*RESP\_ERRCODE*, *ERR\_OK*)  
 putExtra(RESP\_CODE, code)  
 putExtra(RESP\_STATE, state)  
 **}** startActivity(backIntent)  
 finish()  
 **}** //关闭按钮  
 tv\_close.setOnClickListener **{** backIntent.*apply* **{** putExtra(*RESP\_ERRCODE*, *ERR\_AUTH\_DENIED*)  
 putExtra(*RESP\_ERRSTR*, "拒绝授权")  
 **}** startActivity(backIntent)  
 finish()  
 **}** }  
 */\*\*  
 \* 重写返回按钮逻辑  
 \*/* override fun onBackPressed() {  
 backIntent.*apply* **{** putExtra(*RESP\_ERRCODE*, *ERR\_AUTH\_DENIED*)  
 putExtra(*RESP\_ERRSTR*, "拒绝授权")  
 **}** startActivity(backIntent)  
 super.onBackPressed()  
 }  
 */\*\*  
 \* 获取请求授权app信息，显示在界面上---是否允许xxx登录  
 \*/* private fun getOtherAppInfoByAppId() {  
 showLoadingDialog()  
 ApiRequest.getOtherAppInfoByAppId(bActivity, object : OnNetWorkBackListener<AuthorOtherAppInfo> {  
 override fun Success(result: AuthorOtherAppInfo?) {  
 if (result != null) {  
 runOnUiThread **{** dismissLoadingDialog()  
 tv\_name.*text* = result.*data*.*app\_name* GlideLoadUtils.getInstance().glideLoad(this@AuthorActivity,

result.*data*.*app\_icon*, img\_logo, R.drawable.*url\_default\_icon*)  
 **}** }  
 }  
 override fun Failure(obj: Any?) {  
 dismissLoadingDialog()  
 ToastUtils.showShort(obj?.toString() ?: "未知错误")  
 }  
 override fun otherData(result: String?) {  
 dismissLoadingDialog()  
 ToastUtils.showShort(result ?: "未知错误")  
 }  
 }, appId)  
 }  
 */\*\*  
 \* 为第三方获取授权码  
 \*/* private fun getAuthorCode() {  
 showLoadingDialog()  
 ApiRequest.getAuthorCode(bActivity, object : OnNetWorkBackListener<AuthorCodeInfo> {  
 override fun Success(result: AuthorCodeInfo?) {  
 if (result != null) {  
 runOnUiThread **{** dismissLoadingDialog()  
 code = result.*data*.*code* state = result.*data*.*state* **}** }  
 }  
 override fun Failure(obj: Any?) {  
 dismissLoadingDialog()  
 ToastUtils.showShort(obj?.toString() ?: "未知错误")  
 }  
 override fun otherData(result: String?) {  
 dismissLoadingDialog()  
 ToastUtils.showShort(result ?: "未知错误")  
 }  
 }, appId, state)  
 }  
 companion object {  
 val REQ\_STATE = "req\_state"  
 val RESP\_CODE = "req\_code"  
 val RESP\_STATE = "req\_state"  
 val classNameSuffix = ".yxapi.YXEntryActivity"//类名后缀

//云讯App内不能测试调用  
 fun start(context: Context) {  
 val starter = Intent(context, AuthorActivity::class.*java*).*apply* **{** putExtra(*REQ\_APPID*, "111603115735320")  
 putExtra(REQ\_STATE, "lalala")  
 putExtra(*REQ\_PACKAGE*, "com.cgy.demofactory")  
 putExtra(*REQ\_SCOPE*, "author")  
 **}** context.startActivity(starter)  
 }  
 }  
}

## SDK中的代码

### modelbase包

BaseReq .java

public abstract class BaseReq {  
 public String scope;  
  
 public BaseReq() {  
 }  
  
 public void toBundle(Bundle bundle) {  
 bundle.putString(KeyName.*REQ\_SCOPE*, scope);  
 }  
  
 public void fromBundle(Bundle bundle) {  
 scope = bundle.getString(KeyName.*REQ\_SCOPE*);  
 }  
  
 public interface KeyName {  
 String *REQ\_APPID* = "req\_appid";  
 String *REQ\_SCOPE* = "req\_scope";  
 String *REQ\_PACKAGE* = "req\_packageName";  
 }  
}

BaseResp.java

public abstract class BaseResp {  
 public String scope;  
 public int errCode;  
 public String errStr;  
 public BaseResp() {  
 }  
 public void toBundle(Bundle bundle) {  
 bundle.putString(KeyName.*RESP\_SCOPE*, scope);  
 bundle.putInt(KeyName.*RESP\_ERRCODE*, errCode);  
 bundle.putString(KeyName.*RESP\_ERRSTR*, errStr);  
 }  
 public void fromBundle(Bundle bundle) {  
 scope = bundle.getString(KeyName.*RESP\_SCOPE*);  
 errCode = bundle.getInt(KeyName.*RESP\_ERRCODE*);  
 errStr = bundle.getString(KeyName.*RESP\_ERRSTR*);  
 }  
 public interface KeyName {  
 String *RESP\_SCOPE* = "resp\_scope";  
 String *RESP\_ERRCODE* = "resp\_errCode";  
 String *RESP\_ERRSTR* = "resp\_errStr";  
 }  
 public interface ErrCode {  
 int *ERR\_OK* = 0;  
 int *ERR\_COMM* = -1;  
 int *ERR\_USER\_CANCEL* = -2;  
 int *ERR\_SENT\_FAILED* = -3;  
 int *ERR\_AUTH\_DENIED* = -4;  
 int *ERR\_UNSUPPORT* = -5;  
 int *ERR\_BAN* = -6;  
 }  
}

SendAuth.java

public final class SendAuth {  
 private SendAuth() {  
 }  
 public static class Req extends BaseReq {  
 public String state;  
 public Req() {  
 }  
 public Req(Bundle bundle) {  
 fromBundle(bundle);  
 }  
 public void fromBundle(Bundle bundle) {  
 super.fromBundle(bundle);  
 state = bundle.getString(KeyName.*REQ\_STATE*);  
 }  
 public void toBundle(Bundle bundle) {  
 super.toBundle(bundle);  
 bundle.putString(KeyName.*REQ\_STATE*, state);  
 }  
 public interface KeyName {  
 String *REQ\_STATE* = "req\_state";  
 }  
 }  
 public static class Resp extends BaseResp {  
 public String code;  
 public String state;  
 public Resp() {  
 }  
 public Resp(Bundle bundle) {  
 this.fromBundle(bundle);  
 }  
 public void fromBundle(Bundle bundle) {  
 super.fromBundle(bundle);  
 code = bundle.getString(KeyName.*RESP\_CODE*);  
 state = bundle.getString(KeyName.*RESP\_STATE*);  
 }  
 public void toBundle(Bundle bundle) {  
 super.toBundle(bundle);  
 bundle.putString(KeyName.*RESP\_CODE*, code);  
 bundle.putString(KeyName.*RESP\_STATE*, state);  
 }  
 public interface KeyName {  
 String *RESP\_CODE* = "req\_code";  
 String *RESP\_STATE* = "req\_state";  
 }  
 }  
}

YXConstants.java

public interface YXConstants {  
 interface Scope {  
 String *AUTHOR* = "author";  
 }  
}

### api包

IYXAPIEventHandler.java

public interface IYXAPIEventHandler {  
 void onResp(BaseResp var1);  
}

YXAPI.java

public class YXAPI {  
 private static final String *TAG* = "com.outim.yxopen";  
 private static final String *YX\_PACKAGE\_NAME* = "com.outim.mechat";  
 private Context context;  
 private String appId;  
 private YXAPI(Context context, String appId) {  
 this.context = context;  
 this.appId = appId;  
 }  
 */\*\*  
 \* 创建YXAPI  
 \*/* public static YXAPI createYXAPI(Context context, String appId) {  
 Log.*d*(*TAG*, "createYXAPI");  
 return new YXAPI(context, appId);  
 }  
 */\*\*  
 \* 发送意图参数  
 \*/* public void sendReq(SendAuth.Req req) {  
 if (req.scope.equals(YXConstants.Scope.*AUTHOR*)) {  
 String className = *YX\_PACKAGE\_NAME* + ".ui.activity.AuthorActivity";  
 Intent intent = new Intent();  
 intent.setClassName(*YX\_PACKAGE\_NAME*, className);  
 intent.putExtra(BaseReq.KeyName.*REQ\_APPID*, appId);  
 intent.putExtra(BaseReq.KeyName.*REQ\_PACKAGE*, context.getPackageName());  
 intent.putExtra(BaseReq.KeyName.*REQ\_SCOPE*, req.scope);  
 intent.putExtra(SendAuth.Req.KeyName.*REQ\_STATE*, req.state);  
 intent.addFlags(Intent.*FLAG\_ACTIVITY\_EXCLUDE\_FROM\_RECENTS*);  
 context.startActivity(intent);  
 }  
 }  
 */\*\*  
 \* 处理YX APP返回的数据  
 \*/* public void handleIntent(Intent intent, IYXAPIEventHandler handler) {  
 String scope = intent.getStringExtra(BaseResp.KeyName.*RESP\_SCOPE*);  
 int errCode = intent.getIntExtra(BaseResp.KeyName.*RESP\_ERRCODE*, BaseResp.ErrCode.*ERR\_UNSUPPORT*);  
 String errStr = intent.getStringExtra(BaseResp.KeyName.*RESP\_ERRSTR*);  
 if (scope.equals(YXConstants.Scope.*AUTHOR*)) {  
 SendAuth.Resp resp = new SendAuth.Resp();  
 resp.scope = scope;  
 resp.errCode = errCode;  
 resp.errStr = errStr;  
 switch (errCode) {  
 case BaseResp.ErrCode.*ERR\_OK*:  
 resp.code = intent.getStringExtra(SendAuth.Resp.KeyName.*RESP\_CODE*);  
 break;  
 case BaseResp.ErrCode.*ERR\_AUTH\_DENIED*:  
 resp.errStr = intent.getStringExtra(BaseResp.KeyName.*RESP\_ERRSTR*);  
 break;  
 default:  
 resp.errStr = "未知错误";  
 break;  
 }  
 handler.onResp(resp);  
 }  
 }  
 */\*\*  
 \* 检查YX App是否已经安装  
 \*/* public boolean isYXAppInstalled() {  
 PackageInfo packageInfo = null;  
 try {  
 packageInfo = context.getPackageManager().getPackageInfo("com.outim.mechat", 0);  
 } catch (PackageManager.NameNotFoundException e) {  
 e.printStackTrace();  
 }  
 return packageInfo != null;  
 }  
}

## 使用

### 引入jar包，时间总线库，manifest注册YXEntryActivity

//引入yx sdk包  
implementation files('libs/yxopen.jar')  
implementation 'org.greenrobot:eventbus:3.1.1'

------------------------------------------------

<activity  
 android:name=".yxapi.YXEntryActivity"  
 android:exported="true" />

### 在包名.yxapi文件夹中创建YXEntryActivity类

public class YXEntryActivity extends Activity implements IYXAPIEventHandler {  
 @Override  
 protected void onCreate(@Nullable Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 //指定本Activity为云讯返回Intent信息的接收容器  
 App.*getInstance*().getYxapi().handleIntent(getIntent(), this);  
 }  
 @Override  
 public void onResp(BaseResp resp) {  
 String scope = resp.scope;//返回信息的  
 String resultTip;  
 if (resp.errCode == BaseResp.ErrCode.*ERR\_OK*) {  
 resultTip = "成功";  
 if (scope.equals(YXConstants.Scope.*AUTHOR*)) {  
 //请求登录的返回的授权码  
 String code = ((SendAuth.Resp) resp).code;  
 //通过数据总线等工具，将授权码传回LoginActivity  
 MyBusEvent event = new MyBusEvent();  
 event.setEventType(MyBusEvent.EventBusType.*LOGIN*);  
 event.setT(code);  
 EventBus.*getDefault*().post(event);  
 }  
 } else if (resp.errCode == BaseResp.ErrCode.*ERR\_USER\_CANCEL*) {  
 resultTip = "取消";  
 } else {  
 switch (resp.errCode) {  
 case BaseResp.ErrCode.*ERR\_AUTH\_DENIED*:  
 if (!TextUtils.*isEmpty*(resp.errStr)) {  
 resultTip = resp.errStr;  
 } else {  
 resultTip = "发送被拒绝";  
 }  
 break;  
 case BaseResp.ErrCode.*ERR\_UNSUPPORT*:  
 resultTip = "不支持错误";  
 break;  
 default:  
 resultTip = "发送返回";  
 break;  
 }  
 }  
 Toast.*makeText*(this, resultTip, Toast.*LENGTH\_SHORT*).show();  
 finish();  
 }  
}

### 在AppLication中初始化全局的YXAPI，并传入appId

public class App extends Application {  
 private static App *mInstance*;  
 private YXAPI yxapi;//全局的yxapi实例，通过App.getInstance().getYxapi()获取  
 @Override  
 public void onCreate() {  
 super.onCreate();  
 *mInstance* = this;  
 yxapi = YXAPI.*createYXAPI*(this, Constants.*YX\_APPID*);  
 }  
 public static App getInstance() {  
 return *mInstance*;  
 }  
 public YXAPI getYxapi() {  
 return yxapi;  
 }  
}

### 在LoginActivity中调用sdk向云讯App发起授权请求

public class LoginActivity extends AppCompatActivity {  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_login*);  
 //时间总线-监听  
 EventBus.*getDefault*().register(LoginActivity.this);  
 //点击登录按钮  
 findViewById(R.id.*btn\_login*).setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 //云讯登录需要传递的参数  
 SendAuth.Req req = new SendAuth.Req();  
 req.state = "yxopendemo\_state";//用于保持请求和回调的状态，授权请求后原样带回给第三方  
 req.scope = YXConstants.Scope.*AUTHOR*;//授权作用域，登录请求传YXConstants.Scope.AUTHOR  
 //云讯登录  
 App.*getInstance*().getYxapi().sendReq(req);  
 }  
 });  
 }  
 @Override  
 protected void onDestroy() {  
 super.onDestroy();  
 //时间总线-注销监听  
 EventBus.*getDefault*().unregister(LoginActivity.this);  
 }  
 @Subscribe(threadMode = ThreadMode.*MAIN*)  
 public void onEvent(MyBusEvent event) {  
 //根据code获取token  
 if (event.getEventType().equals(MyBusEvent.EventBusType.*LOGIN*)) {  
 String code = (String) event.getT();  
 //拿到code之后，根据code换取AccessToken  
 getTokenByCode(code);  
 }  
 }

...

}

### YXEntryActivity拿到code，通过事件总线传回LoginActivity的后续操作

通常把code传给后台，后台根据code和secret获取AccessToken和UID（OpenId），再根据AccessToken和UID获取用户信息，把获取下来的用户信息等保存到自己的后台，然后再把数据传给移动端，移动端登录成功