Android-允许其他应用唤起 App

- Author:OuFungWah
- Date:2018.08.19
- CopyRight:crazywah.com

在刚刚接触到 Android 开发的时候我所学到的 Activity 启动就只有显式调用对应的 Activity。

即在 Intent 上指定本 App 中注册过的 Activity.class 的转跳方式

后来总结 Activity 的显式调用和隐式调用时，知道了除了显式调用以外，我们还可以通过隐式调用转跳到需要的 Activity（包括非本应用的 Activity）

即在 Intent 上仅指定 ACTION 不指定具体 Activity 的转跳方式

但是在现今非常多的应用都支持从 H5 唤起 Native App 或是转跳至 Native App 页面，在这一次中M3.2.0版本开发中就接触到了，所以需要好好地总结一下相关的知识点

1、基本实现

1.1、新建 Activity 类

```
public class MainActivity extend Activity{

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

1.2、在 manifest.xml 注册该 Activity 并声明 intent-filter

```
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
            <!-- 为该 Activity 添加 uri 拦截器 -->
```

```
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />

                <category android:name="android.intent.category.DEFAULT"
/>
                <category android:name="android.intent.category.BROWSABLE"
/>

                <!-- 该 Activity 要响应的 Url 的匹配模式 -->
                <!-- android:scheme - 匹配的协议 -->
                <!-- android:host   - 匹配的主机地址  -->
                <data
                    android:host="www.crazywah.com"
                    android:scheme="oufenghua" />

                <data
                    android:host="m.test.uri"
                    android:scheme="http"/>
            </intent-filter>
        </activity>
    </application>
```

1.3、唤起 Activity

至此，我们的 Activity 就可以被唤起了

我们可以用网页上的一个匹配我们的响应的匹配模式的链接来唤起 Activity

```
<html>
<head>
<title></title>
<script type="text/javascript">
    function openUrl(){
        window.open("oufenghua://www.crazywah.com");
    }
</script>
</head>
<body>
    <a href="javascript:;" onclick="openUrl">点击该链接唤起 App </a>
    <a href="http://m.test.uri">点击该链接也能唤起 App </a>
</body>
</html>
```

或是另外一个 App 通过隐式调用加匹配的 URL 亦可唤起（第三方 SDK 的惯常用法）

```
    //转跳方法一
    Uri httpUri = Uri.parse("http://m.test.uri");
    httpIntent = new Intent();
    httpIntent.setData(httpUri);
```

```
        httpIntent.setAction(Intent.ACTION_VIEW);
        startActivity(httpIntent);

        //转跳方法二
        Uri oufenghuaUri = Uri.parse("oufenghua://www.crazywah.com");
        oufenghuaIntent = new Intent();
        oufenghuaIntent.setData(oufenghuaUri);
        oufenghuaIntent.setAction(Intent.ACTION_VIEW);
        startActivity(oufenghuaIntent);
```

2、参数详解

2.1、ACTION_VIEW 解析

标识用于响应并展示数据的页面。

这是最通用的 action，使用 ACTION_VIEW 携带数据发出隐式调用可以获得最多响应。

例如说：

- 如果使用`mailto:(邮箱地址)`作为数据传入即可唤起邮件 App 发送邮件的页面(如 Gmail app)
- 如果使用`smsto:(电话号码)`作为数据传入即可唤起短信的 App 的发送短信页面；
- 如果使用`telto:(电话号码)`作为数据传入即可唤起电话的 App 的拨打电话页面；

响应页面获取数据的方法：调用接收到的 Intent 对象的 `getData()`

String 常量值: "android.intent.action.VIEW"

2.2、CATEGORY_DEFAULT 解析

任何用于响应并显示 Data 的可选 Activity 必须要设置该 category。

当有 Intent 需要被响应时，没有设置该 category 的 Activity 将会隐藏，不供用户选择。

该属性通常不会在Intent中设置 - 它用于在包中指定的intent过滤器（intent-fliter）。

如果该用于响应某些数据项的 Activity 没有设置该 category 则将无法被唤起。 若发起唤起动作的 app 中没有做 try catch 处理会抛出异常 `android.content.ActivityNotFoundException: No Activity found to handle Intent`

String 常量: "android.intent.category.DEFAULT"

2.3、CATEGORY_BROWSABLE 解析

任何用于响应浏览器唤起的 Activity 都应该设置该 category。

如： 用户在浏览网页时有一个发送 Email 的链接可以点击，一般这样的链接会发起一个带有 BROWSABLE 的 category，所以只有设置了 BROWSABLE 的 Activity 才会出现在响应请求的列表中。

设置了该 category 你必要保证你的页面是安全的，因为你的页面将会被任何符合的 Intent 唤起（如数据的处理、异常处理、页面逻辑等）

String 常量: "android.intent.category.BROWSABLE"

3、数据处理：

既然我们都可以让第三方应用或网页把我们唤起了，而且官方文档中说，ACTION_VIEW 就是用来展示数据的，那么少不免的我们就需要从唤起的动作中获取到我们需要的数据了

3.1、获取从第三方 App 转跳携带的参数

如果是从第三方 App 转跳的 Intent 中获取参数就和一般的启动 Activity 从 Intent 中获取参数即可。

如：

从发起 App 传来两个参数 id 和 name：

```
Intent intent = new Intent();
Intent.setAction(Intent.ACTION_VIEW);
Uri uri = Uri.prase("oufenghua://open.mudule.a");
//打包参数
Bundle arguments = new Bundle();
arguments.putLong("id",20180819);
arguments.putString("name","oufenghua");

intent.putExtras(arguments);
startActivity(intent);
```

被唤起的 Activity 获取参数

```
Bundle arguments = ActivityA.this.getIntent().getExtras();
long id = arguments.getLong("id",0);
String name = arguments.getString("name","");
```

3.2、获取接在 URL 后面传递的参数

常见的 Url 带参数转跳都形如 `url?key1=value1&key2=value2...`，如何获取该参数呢

我们可以通过 Uri 对象的 `getQueryParameter(String)` 方法进行获取

如：

我们需要获取 `oufenghua://open.mudule.a?id=0&name=crazywah` 中的 id 和 name

```
Uri data = ActivityA.this.getIntent().getData;
long id = data.getQueryParameter("id");
String name = data.getQueryParameter("name");
```

如果要保证数据安全，最好使用加密算法将传递参数加密，到了目标客户端再解密

4、进阶用法：

根据基础用法我们不难看见，响应第三方应用唤起需要为 Activity 在清单文件中设置 action 和 category，但是这样就显得很不灵活，任何要响应第三方唤起的 Activity 都要各自加一个。在这里我觉得在中M项目中的做法就非常靠谱，**就是用一个页面进行 URL 的集体处理，而后在再通过主机名（host）或传递的参数的不同进行页面分发。**

4.1、创建统一处理请求的 SchemeDispatcher类

该类中的核心有两个：

1. 获取当前 Activity 的上下文对象（Context）
2. 处理 Activity 的转跳分发

```
public class SchemeDispatcher {

    private static final String HOST_OPEN_MODULE_A = "open.module.a";
    private static final String HOST_OPEN_MODULE_B = "open.module.b";
    private static final String HOST_OPEN_MODULE_C = "open.module.c";
    private static final String HOST_OPEN_MODULE_D = "open.module.d";

    private Context mContext;

    public SchemeDispatcher(Context context) {
      //获取统一接收请求的 Activity 的上下文资源
        this.mContext = context;
    }

    //按照不同的 HOST 对唤起时间进行分发，唤起对应的 Activity
    public void launch() {
        //先获取 host
        Intent intent = ((Activity) mContext).getIntent();
        Uri data = intent.getData();
        String host = data.getHost();
        //再进行分发
        switch (host) {
            case HOST_OPEN_MODULE_A:
                //转跳 Activity 的具体操作，按业务需求来
                ActivityA.launch(mContext);
                break;
            case HOST_OPEN_MODULE_B:
                ActivityB.launch(mContext);
                break;
            case HOST_OPEN_MODULE_C:
                ActivityC.launch(mContext);
                break;
            case HOST_OPEN_MODULE_D:
                ActivityD.launch(mContext);
                break;
            default:
```

```
                Toast.makeText(mContext, "没有对应的Activity",
Toast.LENGTH_SHORT).show();
                break;
        }
    }
}
```

4.2、创建统一接收唤起请求的 Activity

该 Activity 只是一个空壳，不需要有界面，因为分发完了唤起请求以后它就被销毁了

```
public class SchemeDispatchActivity extends Activity {

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        SchemeDispatcher dispatcher = new SchemeDispatcher(this);
        dispatcher.launch();
        this.finish();
    }
}
```

4.3、在清单文件中注册 Activity

在清单文件中设置好我们需要的 action、category 和所支持的协议（或伪协议）

```
    <activity android:name=".SchemeDispatchActivity" >
          <intent-filter>
              <action android:name="android.intent.action.VIEW"/>
              <category android:name="android.intent.category.DEFAULT"/>
              <category android:name="android.intent.category.BROWSABLE"/>
              <!-- 我们拦截的是我们自己定义好的伪协议 oufenghua 下面的所有 host 的
唤起请求 -->
              <data android:scheme="oufenghua" android:host="*"/>
          </intent-filter>
    </activity>
```

至此，我们的唤起请求分发 Activity 就创建成功可以投入使用了。

4.4、唤起

我们可以通过网页进行唤起：

```
<!DOCTYPE html>
<html>
  <head>
  <title>Uri test</title>
```

```
    <script type="text/javascript">
        function openA(){window.open("oufenghua://open.module.a");}
        function openB(){window.open("oufenghua://open.module.b");}
        function openC(){window.open("oufenghua://open.module.c");}
        function openD(){window.open("oufenghua://open.module.d");}
    </script>
  </head>
  <body>
    <a href="oufenghua://www.crazywah.com">Click here to go to the test
app</a><br/>
    <a href="javascript:;" onclick="openA()">Click here to launch Module
A</a><br/>
    <a href="javascript:;" onclick="openB()">Click here to launch Module
B</a><br/>
    <a href="javascript:;" onclick="openC()">Click here to launch Module
C</a><br/>
    <a href="javascript:;" onclick="openD()">Click here to launch Module
D</a>
  </body>
</html>
```

也可以从其他 app 唤起

```
    Intent intent = new Intent();
    Uri uri = Uri.prase("oufenghua://open.module.a");
    intent.setAction(Intent.ACTION_VIEW);
    intent.setData(uri);
    startActivity(intent);
```

4.5、优势

采用这种统一分发的形式来处理唤起请求可以让我们后期的维护带来极大的方便：

- 添加，就只需要在 switch 上添加对应的 case
- 修改，也只需要修改对应 case 中的唤起逻辑
- 删除，直接把对应的 case 删除掉即可

非常利于后期的维护

5、总结

以上就是我在中M3.2.0版本的开发中发现自己不会的和比较好的方案的总结，开发的路还有好长好长，要抱着学习的心态继续前行。