2.4 第三者による拡張を可能にする

こんなケース

師範！ 先日アプリを公開して以降、改善要望が自分のメールボックスにひっきりなしに届きます。しかし不適当なのではと思う要望も中にはあり、その度に対応に苦慮しています。
このような時、どのように行動するのが筋だとお考えになりますか？

心得るべし

すべての改善要望にお主自身が対処する必要はないのじゃ。この極意をもってすれば、必ずや切り抜けることができるじゃろう。

1. 改善したい者をもってして拡張させることを考えるべし。
2. 拡張させたい機能は、プラグインアプリとして個別に分離すべし。
3. 連携手段の1つ、暗黙的Intentの威力を知るべし。
4. 作者を限定するには証明書の一致を要求すべし。

公開したアプリの知名度が上がってくると、決まっていろいろな要望を受けることになるものです。中には非常に良いアイデアもあるでしょうし、このケースのように一部にしか受けないだろうと思われるものもあります。このような要望を全て最善な形で処理し、万人受けを目指すという選択肢もありますが開発・テストなどによる開発側の負担も大きくなりますし、アプリのサイズも肥大化しがちになってしまいます。また、そもそも何か良い改善案を持っていても、あまり表面に出してこないユーザもいるかもしれません。

このような場合にとれる最善策の1つとして、日本語IME「Simeji」における「マッシュルーム」やTwitterクライアント「twicca」における「プラグイン」などのように第三者が自由に機能拡張してもらえるような構造にしてしまうことがあります。

スキルと自信のある開発者が自分で機能を追加できるようになっていれば、このような場合でも自分が直接開発しているアプリでの機能はあくまでも基本的かつ最小限度のものに止めておきます。そのようにしておけば要望のある個々人に拡張を促すといったような柔軟な対応がとれるようになります。

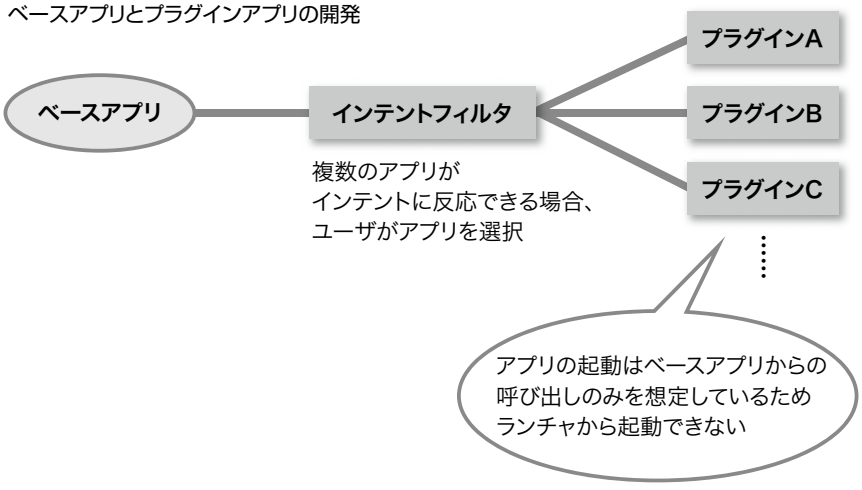
また、このように拡張の余地を残しておくことで、例えばプラグイン開発業務を受託したり有料プラグインとしてニーズを汲み取ったりするなど、そのアプリ自体のマネタイズの観点からも新しいチャンスを生み出すことができます。

◆暗黙的Intentによるプラグイン構造

このようなアプリの構造は一般的には「プラグイン構造」と呼ばれているもので、それを実現する方法はいろいろありますが、ここでは簡単な例として、プライベートなaction（動詞）を持つ暗黙的なIntentを介してベースアプリからプラグインに制御を移す方法を説明します。暗黙的なIntentとは、普段ギャラリで画像を共有したり、ファイルマネージャなどでファイルを開いたりする際に使用される、何か（extra）についてしたいこと（action/category）だけ定義してあって特に宛先（component）を持たないIntent、いわば「これにこういうことしたいんだけど誰かできる人いませんか？」的なIntentのことです。

この方法では、プラグインAPI仕様としてあらかじめベースアプリが決めたプライベートなactionを満たすようなフィルタを実装するように告知しておきます。そしてベースアプリはプラグイン可能にしたい操作を実行する時にプライベートなactionを持つ暗黙的なIntentを発行することで適切なプラグインをユーザに選ばせ、実行します（**図2.4.1**）。

図2.4.1　ベースアプリとプラグインアプリの開発



◆ベースアプリ

まず基本のアプリから見ていくことにします。前述のとおりプラグインを使いたくなった時に暗黙的なIntentを発行します。

Activity#startActivity()やActivity#startActivityForResult()を使って暗黙的なIntentを発行すると、Androidシステムはそのintentに対応可能なアプリを検索しようとすることを思いだしてください。Intentに対応可能なアプリとは、そのIntentが持つaction/categoryを満たすフィルタを持つアプリのことを指します。つまり、actionやcategoryをプライベートなものにしておくことで、そのIntentに興味があるアプリ全体の中から好きなものに制御を渡すことができるのです。

●リスト 2.4.1

SampleBaseActivity.java:

```
package com.example.plugin.base;

...
import com.example.plugin.protocol.SamplePluginProtocol;

public class SampleBaseActivity extends Activity {
    @Override
    public void onCreate(…) {
        …
        Button invokePluginButton = (Button)findViewByid(R.id.invoke_plugin_button);
        invokePluginButton.setClickListener(new InvokePluginButtonListener());
        …
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        switch (requestCode) {
            case SamplePluginProtocol.ID_SOMETHING:
                Log.d("SampleActivity", String.format(" SOMETHING plugin has completed,
                resultCode=%d", resultCode));
        }
    }

    // プラグインを起動するボタンのクリックリスナークラス
    private class InvokePluginButtonListener implements View.OnClickListener {
        @Override
        public void onClick(View v) {
            // プラグインの起動
            Intent intent = new Intent();
            intent.setAction(SamplePluginProtocol.ACTION_SOMETHING);
            intent.putExtra(SamplePluginProtocol.PARAM," parameter");
            …
            SampleBaseActivity.this.startActivityForResult(
                Intent.createChooser(intent,"プラグインを選んで下さい"),
                SamplePluginProtocol.ID_SOMETHING
            );
        }
    }
}
…
```

SamplePluginProtocol.java:

```
package com.example.plugin.protocol;

// プラグインAPIを定義するクラス
```

```
// これは広く公開しておく
public class SamplePluginProtocol {
    // パラメータのキー名
    public static final String PARAM = "param" ;
    // SOMETHING: 何かする
    public static final int ID_SOMETHING = 1;
    public static final String ACTION_SOMETHING = "com.example.plugin.action.SOMETHING";
}
```

見てのとおり、プラグインの起動は他のアプリを起動するのとなんら差はありません。しかし、Activity#startActivityForResult()に渡したパラメータはきちんとプラグインへ渡されますし、プラグインがActivity#setResult()した結果はもちろん、Activity#onActivityResult()で受け取ることができます。このようにかなり単純な形ですが、これだけである機能（ここでは単なるボタンのクリック操作のハンドリング）で「第三者のコードを使う」という意味では十分役割を果たすことができます。

◆プラグイン

次にプラグインの方を見ていきましょう。前述のようにベースアプリは決まったactionの暗黙的なIntentを発行しますから、その特徴に合致するフィルタを宣言すればよいことになります（**リスト2.4.2**）。

●リスト2.4.2

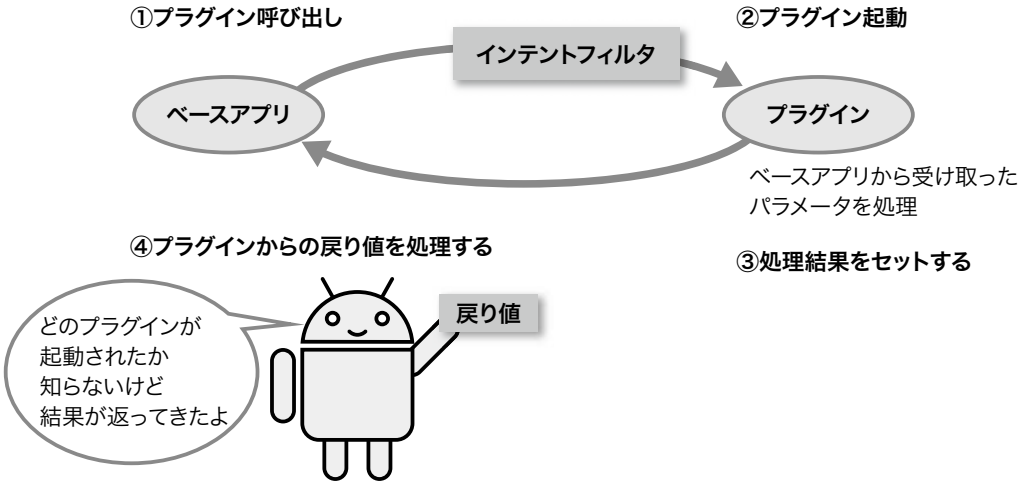
```
AndroidManifest.xml:
…
<!-- 唯一のActivity。理由は本文を -->
<activity android:name=".SamplePluginActivity" android:label="SampleBase Plugin" >
    <intent-filter>
        <action android:name="com.example.plugin.action.SOMETHING" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
…
SamplePluginActivity.java:

package com.example.ubercool.plugin;

public class SamplePluginActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        // プラグインが起動された！
    }
}
```

このように、こちらも何も変哲のないただのアプリでかまいません。<intent-filter>で、先のベースアプリで使用しているactionに合致したフィルタを定義していることはもちろんのことですが、プラグインとして動作させる性格上、一点工夫をしています。そもそもプラグインというものは、単にベースアプリから使われるだけのものであって、ランチャから単独起動されることを想定していません。そこで、このようにaction「android.intent.action.MAIN」およびcategory「android.intent.category.LAUNCHER」に合致するActivityをいっさい持たないようにすることでランチャのリストに上がること自体を防いでいます(**図2.4.1**)。

図2.4.2　プラグイン処理の流れ



これで、曲がりなりにもここではcom.example.plugin.action.SOMETHINGに合致するフィルタを持つアプリを書くだけで誰でもベースアプリの一機能を拡張することができるようになったわけです(**図2.4.2**)。さらにベースアプリ側で複数の機能についてこれを行うことによって、拡張性を一段と強化していくことができます(**図2.4.3**)。

図2.4.3　ベースアプリを作成したらWebページなどで告知しよう!

- ベースアプリがプラグインを呼び出す条件を明記
- パラメータ、戻り値の仕様を明記
- 開発手順・実装例を明記(サンプルコードのダウンロードがあると良い)