

補足資料

Intentによるアプリケーションとアクティビティの呼出し

Android アプリのキモとなるIntentとは何？

「Intent」(呼び出し要求)とは Android 独自の機能です。簡単にいえばアプリケーションや他のアクティビティを呼び出す機能ですが、他のアプリケーションを機能や扱うデータ型式で“検索”して呼び出すことができます。

たとえば、ウェブブラウザを呼び出したい場合、「View」(データの表示)というアクションと URL というデータを指定します。ウェブブラウザの名前などを指定する必要がありません。この Intent が実行されると、Android は、URL を扱うことができ、かつそれを表示できる自身が持つアプリケーションの中から探し、そのアプリケーションに URL を渡して表示を行なわせます。もし、複数のアプリケーションがある場合は、ユーザーがどのアプリケーションを使うのかを選ぶダイアログが表示されます。

	<p>Android 端末で追加のウェブブラウザをインストールしたのちに、何らかのアプリケーションで URL をタッチすると、アプリの選択画面が表示される。これは Intent によって、ウェブブラウザを呼び出している例だ</p>
---	---

このようにアプリケーションで_intentを使うと、コード中にブラウザの名前を入れる必要もないし、実際の実行時に別のブラウザに置き換わっていたり、複数のブラウザが端末にあっても問題ないのです。

また逆にブラウザを起動するアプリケーションがあったとしても、ユーザーは別のブラウザをインストールしたら、そちらを起動するようにも指定できます。このようにアプリケーション同士を直接相互依存させるのではなく、実際に行なう作業(アクションといいます)を指定して、外部のアプリケーションと間接的に関係させることができるわけです。(これを「暗黙的_intent」と呼ぶ)

また、intentは1つのアプリケーションの中で、アクティビティを切り替えるのにも利用します。アプリケーションは、必ずアクティビティを含むため、intentはアクティビティを呼び出しているともいえます。つまり、アプリケーション内での自作した別のアクティビティの呼び出しは、intentの特殊な形と考えることができるわけです。この場合は、機能などでなく、直接別のアクティビティ名を使い、呼び出します。(これを「明示的_intent」と呼ぶ)

まずは簡単な例(暗黙的_intent)を作成してみる

今回は、壁紙を変更するアクションを呼び出すソースコードを用意しました。

●AnmokuIntentDemo.java

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.content.Intent;

/**
 * 暗黙的_intentの一例。壁紙を変更するアクションを呼び出す
 */
public class AnmokuIntentDemo extends Activity
    implements OnClickListener {

    /** 起動時処理 */
```

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // main.xml レイアウトファイルを読み込み、画面に設定
    setContentView(R.layout.main);

    /* main.xml ファイル内のidを指定してボタンを取得し、
    イベント処理を設定*/
    ImageButton button =
        (ImageButton) this.findViewById(R.id.Button);
    button.setOnClickListener(this);
}

/** ボタンクリック時のイベント処理内容 */
public void onClick(View view) {
    /* 壁紙を変更するアクション処理要求 (intent) を作成
    とくに今回は必要がないのでデータは送らない。該当する任意
    のアプリケーションが起動 */
    Intent intent = new Intent(Intent.ACTION_SET_WALLPAPER);
    // intentを送信
    startActivity(intent);
}
} //classの終わり

```

●res/layout/main.xml (レイアウト指定 XML ファイル)

```

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

    android:orientation="vertical"

    android:layout_width="fill_parent"

    android:layout_height="fill_parent"

    >

```

<!-- 画像ファイルを扱う場合は res/drawable/ 以下に画像ファイル xxx を配置し、@drawable/xxx で指定すれば良い。 -->

```
<ImageButton android:id="@+id/Button"

    android:src="@drawable/icon"

    android:layout_width="wrap_content"

    android:layout_height="wrap_content">

</ImageButton>
```

```
</LinearLayout>
```

Intentの最も簡単な例は、自作したアプリケーション(アクティビティ)から別アプリケーションを呼び出す場合です。まずは Android の画像系を扱うアプリケーションを呼び出してみましょう。

Android 自身が用意しているIntentに関しては、Android Developer の Reference に情報が 있습니다。ただし、一カ所にすべてまとまっているわけではなく、Intent クラスで規定する標準のアクションと、他のクラス(たとえば、設定関連のアクションは、Settings クラスなど)が持つアクション(アクティビティの表示)に分かれています。今回は Intent クラスに定数定義があります。

```
Intent intent = new Intent( Intent.ACTION_SET_WALLPAPER );
startActivity(intent);
```

1 行目は **Intent オブジェクトの作成**で、このとき引数としてアクションを渡します。今回のアクション(Intent.ACTION_SET_WALLPAPER)は、任意の背景画像を設定するので、データを別途必要としないので、とくにになにもせずに startActivity でアクティビティを表示させることができます。

これが暗黙的Intentの一例です。このようにスマートフォン内の他のアプリを簡単に起動することができます。

以下は、URL を指定してブラウザを起動する暗黙的インテントの例です。

●BrouserIntentDemo.java

```
import android.app.Activity;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.content.Intent;

/**
 * 暗黙的インテントの一例。ブラウザを起動するアクションを呼び出す
 */
public class BrouserIntentDemo extends Activity
    implements OnClickListener {

    /** 起動時処理 */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // main.xml レイアウトファイルを読み込み、画面に設定
        setContentView(R.layout.main);

        // main.xml ファイル内のidを指定してボタンを取得
        Button button =
            (Button) this.findViewById(R.id.startBrouserButton);
        button.setOnClickListener(this);
    }

    /** ボタンクリック時のイベント処理内容 */
    public void onClick(View view) {
        //データの場所を特定するオブジェクトを生成
        Uri uri = Uri.parse("http://www.google.co.jp");
        //アクションは指定場所のデータをユーザに表示する
        Intent i = new Intent(Intent.ACTION_VIEW, uri);
    }
}
```

```
        //起動
        startActivity(i);
    }
} //class の終わり
```

●res/layout/main.xml (レイアウト指定 XML ファイル)

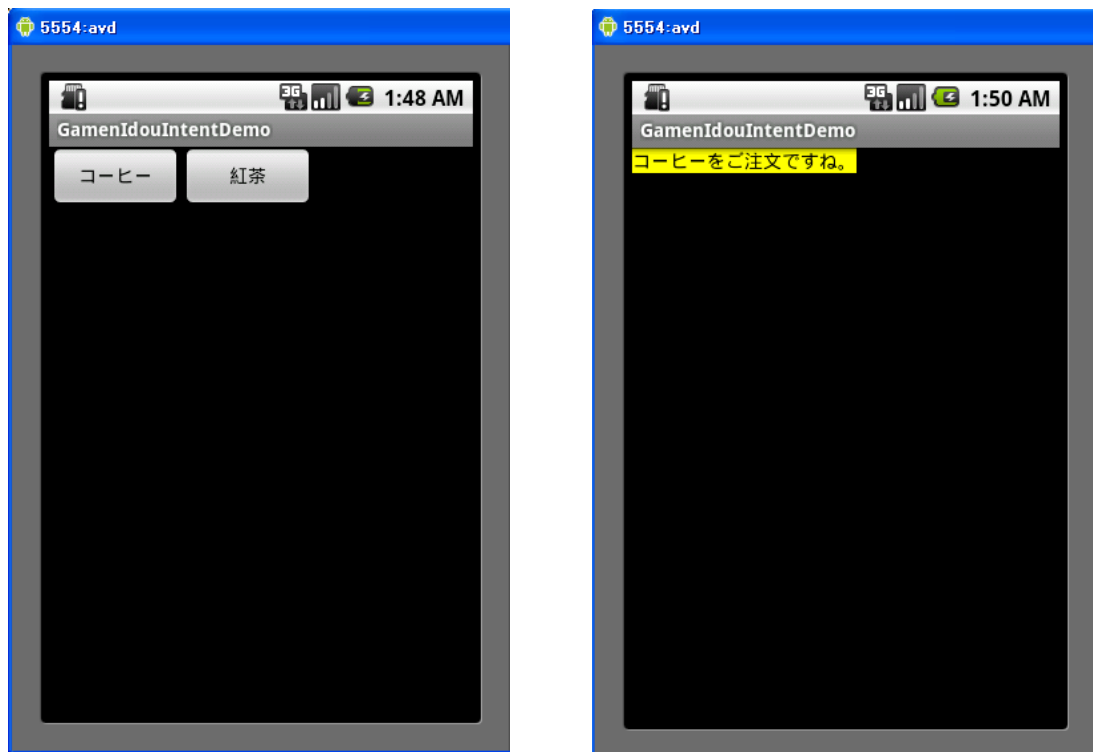
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <Button android:text="検索サイトへ行く"
        android:id="@+id/startBrowserButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </Button>

</LinearLayout>
```

次に明示的Intentの例を作成してみる

自作した別のアクティビティを呼び出す簡単な例を用意しました。最初の画面でコーヒーのボタンを選択すると、注文名が次画面に渡され、表示されます。



●MainGamen. java

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.content.Intent;

public class MainGamen extends Activity implements OnClickListener {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
```

```

    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // main.xmlファイル内のidを指定してボタンを取得し、イベント処理を設定
    Button coffeeButton =
        (Button) this.findViewById(R.id.coffeeButton);
    coffeeButton.setOnClickListener(this);
    Button teeButton =
        (Button) this.findViewById(R.id.teeButton);
    teeButton.setOnClickListener(this);

}

public void onClick(View v) {
    // このクラスオブジェクトと次画面のクラスオブジェクトをつなぐ
    Intent orderIntent = new Intent(this, OrderGame.class);

    //選択されたボタンを取得
    Button orderButton = (Button)v;

    //選択されたボタン上の文字列を取得
    String orderName = (String) orderButton.getText();

    //キーと値を付加情報としてセット
    orderIntent.putExtra("注文", orderName);

    //intentを送信
    startActivity(orderIntent);

}
} //終わり

```


●res/layout/main.xml (レイアウト指定 XML ファイル)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <Button android:text="コーヒー" android:id="@+id/coffeeButton"
        android:layout_width="100px"
        android:layout_height="wrap_content"></Button>

    <Button android:text="紅茶" android:id="@+id/teeButton"
        android:layout_width="100px"
        android:layout_height="wrap_content"></Button>

</LinearLayout>
```

次に呼び出される注文画面クラスを作ります。

●OrderGamen.java

```
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;

public class OrderGamen extends Activity {

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

```
// main.xmlファイル内のidを指定してボタンを取得し、イベント処理を設定
TextView orderTextView =
    (TextView) this.findViewById(R.id. orderTextView);

// getIntentでintentの中身を取得
Intent intent = this.getIntent();

// キーを使って送信された付加情報の値を取得し、表示
String orderName = intent.getStringExtra("注文");
orderTextView.setText(orderName + "をご注文ですね!");

    }
} // 終わり
```

●res/layout/order.xml (レイアウト指定 XML ファイル)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <TextView
        android:background="#ffff00"
        android:textColor="#000000"
        android:id="@+id/orderTextView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="" />

</LinearLayout>
```

ここでは複数のアクティビティを使用しますが、アプリケーション中にアクティビティを追加する際にはアプリケーションのマニフェストファイル (**AndroidManifest.xml**)にアクティビティを登録する必要があります。

(メインのアクティビティについては eclipse に よって自動的に登録されているため、登録する必要はありません。)

以下がそのマニフェストファイルです。

●AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="intentdemo.GamenIdouIntentDemo" android:versionCode="1"
    android:versionName="1.0">
    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity
            android:name=".MainGamen"
            android:label="@string/app_name">
            <intent-filter>
                <action
                    android:name="android.intent.action.MAIN" />
                <category
                    android:name="android.intent.category.LAUNCHER"
                    " />
            </intent-filter>
        </activity>

        <!-- 以下を追加 android:name にはアクティビティのクラス名を記載する。 -->
        <activity android:name=".OrderGamen">
        </activity>

    </application>
    <uses-sdk android:minSdkVersion="3" />

</manifest>
```

Android アプリのデータ保存方法の「プリファレンス」の使い方は？

Android アプリでは、いくつかのデータ保存方法が用意されていますが、最も簡単に扱える方法が、この「プリファレンス(Preference)」であると思います。

プリファレンスは、データを、キー名と値の組み合わせで保存する形をとります。

Java でいう HashTable や、VB 等の Dictionary のような感じですね。

データの量や保存したい形にもよりますが、数個の設定値を保持させておきたいぐらいであれば、このプリファレンスで事足りると思います。

それでは、サンプルコードとともに、プリファレンスの使い方をみていきましょう。

●プリファレンスの取得方法

次は、プリファレンスへ保存したデータの取得方法をみてみましょう。

```
SharedPreferences pref =  
    getSharedPreferences( "pref" ,MODE_PRIVATE);  
String str = pref.getString("key", "");
```

まずは、`getSharedPreferences()`メソッドを使って、`SharedPreferences` 型インスタンスを取得します。

次はデータを取り出しますが、書込み時と同じく、保存されているデータの型により、`getString()`、`getFloat()`、`getInt()`、`getLong()`、`getBoolean()`等を使い分けます。

第二引数は、もしそのキーの値が存在しないときの初期値とする値を指定できます。

アクティビティのライフサイクル

Android は、マルチタスクで複数の処理を同時に実行できますが、画面（フォアグラウンド）に表示されるアプリケーションは、常に 1 つと定められています。

トランプのカードが重なっている様子を想像してみてください。カードの 1 枚ずつがアクティビティで、直近で呼び出されたアクティビティが一番上（フォアグラウンド）に表示されるのです。

携帯端末上で動作するアプリケーションは、さまざまな要因で割り込みが発生し、その都度中断されます。例えば、電話が着信すれば着信画面に切り替わります。また、ユーザーが一定時間操作しなければ端末が“スリープ・モード”になります。

割り込みの要因が終了した場合、例えば電話が終わったり、ユーザーが操作してスリープ・モードを解除したりした際には、元のアクティビティを中断前の状態から再開する必要があります。

ゲームのアプリケーションの場合、電話の着信のタイミングで一時停止しておけば、電話が終わった時にゲームオーバーの画面が表示されているということは避けられます。

アクティビティのインスタンスが作成されてから破棄されるまでのサイクルをライフサイクルと呼びます。ライフサイクルの間は、アクティビティは色々な状態を遷移し、対応するメソッドが呼ばれます。

アクティビティの状態は下記 3 種類です。

実行中	画面のフォアグラウンドに表示されていて、操作が可能な状態。
一時停止中	見えているのに操作はできない状態。 停止前の状態は保持している。 システムが極端なメモリ不足の時に OS から強制終了させられる場合がある。
停止中	見えない状態。操作もできない。 停止前の状態は保持している。 システムがメモリ不足になった時に強制終了させられる場合がある。

アクティビティのライフサイクルに関わる onXX()関数は下記です。

メソッド	説明	強制終了	次
onCreate()	<p>アクティビティが初めて作成されるときに呼び出されます。</p> <p>通常の静的な設定（ビューの作成、リストへのデータのバインドなど）は、すべてのこのメソッドで行う必要があります。</p> <p>このアクティビティの 以前の状態が保存されていた場合、このメソッドにはその状態を保持している Bundle オブジェクトが引数となります。</p> <p>この後には、必ず onStart()が呼び出されます。</p>	不可	onStart()
onRestart()	<p>アクティビティが停止した後、それをもう一度開始する直前に呼び出されます。</p> <p>この後には、必ず onStart()が呼び出されます。</p>	不可	onStart()
onStart()	<p>アクティビティがユーザーから見えるようになる直前に呼び出されます。</p> <p>その後、アクティビティがフォアグラウンドに表示された場合は onResume()が、他のアクティビティの後ろに隠れた場合は onStop()が呼び出されます。</p>	不可	onResume() または onStop()
onResume()	<p>アクティビティがユーザーとの対話を開始する直前に呼び出されます。</p> <p>この時点で、アクティビティはアクティビティスタックの最上位にあり、ユーザーからの入力はこのアクティビティに対して行われます。</p> <p>この後には、必ず onPause()が呼び出されます。</p>	不可	onPause()
onPause()	<p>システムが別のアクティビティを開始しようとしているときに呼び出されます。</p> <p>このメソッドは、保存されていない変更を永続データにコミットする場合や、アニメーションのように CPU を大量に消費する処理を停止する場合に使用するのが一般的です。</p> <p>このメソッドが終了するまでは次のアクティビティが開始されないため、できる限り短時間</p>	可能	onResume() または onStop()

	<p>で実行できるようにしておく必要があります。</p> <p>その後、アクティビティがフォアグラウンドに戻った場合は <code>onResume()</code> が、ユーザーから見えなくなった場合は <code>onStop()</code> が呼び出されます。</p>		
<code>onStop()</code>	<p>アクティビティがユーザーから見えなくなったときに呼び出されます。</p> <p>見えなくなる状況としては、アクティビティが破棄された場合や、再開された別のアクティビティ（既存か新規かを問わず）によって隠された場合が考えられます。</p> <p>その後、アクティビティがユーザーとの対話に戻った場合は <code>onRestart()</code> が、アクティビティが完全に終了する場合は <code>onDestroy()</code> が呼び出されます。</p>	可能	<code>onRestart()</code> または <code>onDestroy()</code>
<code>onDestroy()</code>	<p>アクティビティが破棄される前に呼び出されます。これが、アクティビティが受け取る最後の呼び出しとなります。</p> <p>このメソッドが呼び出される状況としては、アクティビティが完了する場合（<code>finish()</code> が呼び出されたとき）や、システムが領域を確保するために一時的にそのアクティビティのインスタンスを破棄する場合が考えられます。</p> <p>これらの 2 つの状況は、<code>isFinishing()</code> メソッドを使用して識別できます。</p>	可能	なし

まとめ

アクティビティ全体は、`onCreate()` で始まり、`onDestroy()` で終わる。

画面上で見ることができる期間は、必ず `onStart()` で始まり、`onStop()` で終わる。この間、フォアグラウンドにあるとは限らないし、操作が可能とも限らない。

フォアグラウンドの期間は、必ず `onResume()` で始まり、`onPause()` で終わる。

テストプログラム

まあ、実際にプログラムで試してみましょう。

●ActivityMethods.java

```
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class ActivityMethods extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        Log.i("##### TEST #####", "onCreate() called.");
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onStart() {
        Log.i("##### TEST #####", "onStart() called.");
        super.onStart();
    }

    public void onRestart() {
        Log.i("##### TEST #####", "onRestart() called.");
        super.onRestart();
    }

    public void onResume() {
        Log.i("##### TEST #####", "onResume() called.");
        super.onResume();
    }

    public void onPause() {
        Log.i("##### TEST #####", "onPause() called.");
    }
}
```



```

        super.onPause();
    }

    public void onStop() {
        Log.i("##### TEST #####", "onStop() called.");
        super.onStop();
    }

    public void onDestroy() {
        Log.i("##### TEST #####", "onDestroy() called.");
        super.onDestroy();
    }
}

```

このプログラムは、各ライフサイクル関数にログを表示させるようにしています。
これを実行すれば、どういう時にどのon関数が呼ばれるかがわかると思います。

アプリ開始時には、onCreate()、onStart()、onResume()が呼ばれます。
ホームキーを押してホーム画面に戻ってみると、onPause()、onStop()が呼ばれます。
再びアプリに戻ると、onRestart()、onStart()、onResume()が呼ばれます。
戻るボタンを押すと、onPause()、onStop()、onDestroy()が呼ばれ、終了します。

なお、ライフサイクルに関わるonなんとか関数には、super.onXXXを最初に呼ばなければ
いけないそうです。

以上。