Android Fragment

[Android][Android Tips]

・http://developer.android.com/guide/components/fragments.html

概要

・複数のフラグメントを一つのアクティビティに組み合わせて、マルチペイン UI とし、フラグメントを複数のアクティビティで再利用できる
・フラグメントを自分自身のライフサイクルヲもつ、アクティビティのモジュールと見なすことができる。
・フラグメントは、それぞれ入力イベントを持ち、アクティビティが起動している間に追加したり取り除いたりできる。
・フラグメントは常にアクティビティに組み込まれ、ライフサイクルは、直接ホストしているアクティビティのライフサイクルの影響を受ける

・フラグメントをアクティビティレイアウトの一部に追加すると、アクティビティのビュー階層の ViewGroup の中で、自分のビューレイアウトを定義し生存する
・アクティビティレイアウトに、レイアウトファイルに <fragment> と宣言する、もしくはアプリケーションコードから ViewGroup に追加することでことで挿入できる。
・しかしながらフラグメントは、アクティビティレイアウトの一部であることは必須ではない。
・フラグメントを自身の UI を持たない、アクティビティのための不可視なワーカーとして使うこともできる

デザイン

・Android では、フラグメントを Android30(API Level11) で導入した。
・一義的には、タブレットなどの大画面でさらに動的で柔軟な UI デザインをサポートするため。
・タブレットのスクリーンは、ハンドセットと比較して十分に大きく、ユーザーとやりとりする UI コンポーネントのための場所が残されている。
・フラグメントはこのようなビュー階層の複雑な変更の管理が必要内容にデザインされている。

・アクティビティレイアウトにフラグメントを投入することによって、レイアウトに起動中のアクティビティの表示やそれら変更の維持管理を変更できる様になる。

・例えば、ニュースアプリケーションでは、一つのフラグメントに記事のリストを表示し左に配置し、もう一つのフラグメントに記事を表示し、右側に配置することができる。
・どちらのフラグメントも一つのアクティビティに並んで表示され、それぞれライフサイクルコールバックメソッドのセットを持ち、それぞれユーザーの入力イベントを処理する。
・言い換えれば、記事のリスト表示と記事を読むことを一つの同じアクティビティで行うことができるということ。

・フラグメントを再利用可能なモジュールコンポーネントとしてデザインすべき。
・それぞれのフラグメントは、自分自身のレイアウトを定義し、それぞれの振る舞いを自身のライフサイクルコールバックに持つため。
・一つのフラグメントを複数のアクティビティに組み込むことができるので、再利用可能かつ、フラグメント間で、直接操作し合うことは行うべきではない。
・モジュール化されたフラグメントは、フラグメントの組合せを異なったスクリーンサイズにおいて可能にする
・アプリケーションにタブレットとハンドセットをサポートするようにデザインする場合、フラグメントを可能なスクリーンスペースに依存したユーザーエクスペリエンスに応じた、別のレイアウト定義に再利用できる。

フラグメントの生成

・フラグメントを生成するには、Fragment のサブクラスを作成する必要がある。
・Fragment クラスは、Activity によく似ている。
・アクティビティと同じく、onCreate(),onStart(),onPause,onStop() といったコールバックメソッドを持つ
・実際、既存の Android アプリケーションをフラグメントを利用するようにコンバートする場合、単純にアクティビティのコールバックメソッドをそれぞれフラグメントのメソッドに移動する。
・通常、少なくとも以下のライフサイクルメソッドを実装する
  ・onCreate(): フラグメント生成時に呼ばれる。主要なコンポーネントの初期化、ポーズやストップから再開したときの処理を実装
  ・onCreateView():UI を最初に描画するときに呼ばれる。フラグメントのルートビューを返す。null を返すとフラグメントは UI を提供しない
  ・onPause(): ユーザーがフラグメントから離れた時に呼び出される。
・多くのアプリケーションは、少なくとも上記の 3 つのメソッドを各フラグメントに実装するべきである。
・しかし、いくつか他のコールバックもフラグメントのライフサイクルで利用できる。
  ・http://developer.android.com/guide/components/fragments.html#Lifecycle

・Fragment クラスを継承する代わりに、いくつかのサブクラスを利用できる
  ・DialogFragment : フローティングダイアログ。アクティビティのダイアログヘルパーメソッドを利用するのの良い代替となる。
  ・ListFragment : アダプターにより管理されるアイテムをリストする。ListActivity と同様
  ・PreferenceFragment : Preference オブジェクトの階層をリストとして表示する。PreferenceActivity と同様

ユーザーインターフェースの追加

・フラグメントは、通常アクティビティ UI の一部として利用される。
・フラグメントにレイアウトを与えるには、onCreateView() コールバックメソッドを実装する。
・onCreateView() は、フラグメントがレイアウトを描画するタイミングで呼び出される。
・戻りとして、フラグメントのルートとなる View を返すように実装する。

ListFragment のサブクラスである場合、デフォルトの実装では、ListView を返すため、実装する必要はない

・onCreateView() からレイアウトを返すために、XML で定義されたレイアウトリソースから View を生成することができる。
・それを助けるために、onCreateView() は LayoutInflater オブジェクトを提供する

・example_fragment.xml ファイルから、レイアウトをロードする例

```
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.example_fragment, container, false);
    }
}
```

・container パラメータは、親の ViewGroup(アクティビティレイアウトによる)。
・フラグメントは、container に挿入される。
・savedInstanceState パラメータは、Bundle で、以前のフラグメントインスタンスについてのデータを提供する。

・inflate() メソッドは 3 つの引数をとる
  ・resourceID: 生成したいレイアウトのリソース ID
  ・ViewGroup: 生成したレイアウトの親
  ・boolean: 生成したレイアウトが、ViewGroup にアタッチされるべきか。

フラグメントをアクティビティに追加

・通常、フラグメントはホストアクティビティの一部となる。
・アクティビティのレイアウトにフラグメントを追加するには 2 つの方法がある。

■フラグメントをアクティビティレイアウトファイルの中で宣言

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.ArticleListFragment"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
            android:id="@+id/viewer"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

・<fragment> の android:name 属性はレイアウトの中でインスタンス化する Fragment を特定する
・システムがアクティビティレイアウトを生成するとき、それぞれのフラグメントをインスタンス化し、おん CreateView（）メソッドを呼び出す。

フラグメント ID

・それぞれのフラグメントは一意の ID を要求する。それは、システムが、フラグメントをアクティビティが再開するときに、再構築するのに利用される。
・フラグメントに ID を与えるには 3 つの方法がある
  ・android:id 属性 を一意な ID として与える
  ・android:tag 属性 を一意な文字列として与える
  ・上記 2 つを提供できない場合、システムはコンテナビューの ID を利用する

■存在する ViewGroup にフラグメントをプログラムから追加する

・アクティビティが起動している間はいつでも、アクティビティレイアウトにフラグメントを追加することができる。
・単純にフラグメントを追加する ViewGroup を指定するだけ。
・アクティビティの中のフラグメントトランザクション (フラグメントの追加、削除、置き換えなど) を作成するには、FragmentTransaction API を利用する必要が有る。
・FragmentTransaction は、Actiity から、以下の様にインスタンスを取得できる。

```
FragmentManager fragmentManager = getFragmentManager()
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

・add() メソッドによって、フラグメントを追加することができる。

```
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

・add() の最初の引数は、フラグメントを挿入すべき ViewGroup のリソース ID、2 つめの引数は、追加するフラグメント
・FragmentTransaction に変更をかけたら、commit() を呼び出し、変更を確定する。

■ UI のないフラグメントの追加

・UI を持たずにアクティビティのためにバックグラウンドで動作するフラグメントを利用することもできる。
・フラグメントを UI なしに追加するには、add(Fragment,String)(String には、一意となるフラグメントのタグか View ID を与える) を利用する
・これは、onCreateView() を受け取らないので、フラグメントを追加するが、アクティビティのレイアウトにビューを結びつけない。
・なので、onCreateView() メソッドは実装する必要がない。

・UI フラグメントでないフラグメントにに与えるタグは、厳密ではない。
・UI を持たないフラグメントのタグは、その識別にのみ利用される。
・アクティビティから後でフラグメントを取得したい場合、findFragmentByTag() を利用する。

フラグメントの管理

・アクティビティのフラグメントを管理するには、FragmentManager を getFragmentManager() から取得し、を利用する。

FragmentManager でできるいくつかのこと

・アクティビティに存在するフラグメントを findFragmentById()(アクティビティで UI を提供するフラグメント) または、findFragmentByTag()(UI を提供しないフラグメント) で取得する
・popBackStack() でフラグメントをバックスタックから取り出す。
・バックスタックの変更リスナーを addOnBackStackChangedListener() で登録する

フラグメントトランザクションの実行

・アクティビティでフラグメントを利用する上で主要な機能は、追加、削除、置き換え、他のアクションを実行する能力。
・アクティビティへコミットする変更セットのそれぞれついて、トランザクションが呼び出され、FragmentTransaction API を利用してトランザクションを実行できる。
・それぞれのトランザクションをアクティビティにより管理されるバックスタックへ保存することもできる。
・ユーザーは、フラグメントの変更を「戻る」ことができる。

・適切な FragmentTransaction インスタンスを FragmentManager から取得できる。

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

・それぞれのトランザクションは、同時に実行したい変更のセット
・add(),remove() や replace() などトランザクションメソッドが呼ばれたときに実行させたいすべての変更をセットアップできる。
・アクティビティにトランザクションを適用するには、commit() する。

・しかしながら、commit() する前に、トランザクションをフラグメントトランザクションのバックスタックに追加するために、addToBackStack() を実行したい場合もあるだろう。
・バックスタックは、アクティビティにより管理され、ユーザーに直前のフラグメントの状態に戻ることを可能にする。

・以下は、どのようにフラグメントを置き換え、直前の状態をバックスタックに保持するかの例

```
// あたらしいフラグメントとトランザクションの生成
Fragment newFragment = new ExampleFragment();
FragmentTransaction transaction = getFragmentManager().beginTransaction();

// fragment_container ビューを fragment に置き換え、
// トランザクションをバックスタッグに積む
transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);

// トランザクションのコミット
transaction.commit();
```

・newFragment は、R.id.fragment_container ID を置き換え、addToBackStack() を呼び出すことで、この置き換えトランザクションをバックスタックに保存する。
・これにより、ユーザーがトランザクションをリバースし、直前のフラグメントにバックボタンで戻すことができる。

・複数の変更をトランザクションに追加し、addToBackStack() を呼び出した場合、commit() を呼び出しすべての変更が適用される前のすべての変更がバックスタックに一つのトランザクションとして追加され、バックボタンでそれらすべてを元に戻すことができる。

・変更を FragmentTransaction に追加するのに、以下を除いて、順序は関係ない
  ・commit() は最後に呼び出さなければならない
  ・同じコンテナに複数のフラグメントを追加した場合、それらを追加した順序は、ビューの階層に出現する順序となる。

・もし、フラグメントを削除するトランザクションを実行したときに、addToBackStack() を呼ばなければ、フラグメントは、トランザクションのコミット時に破棄され、ユーザーは、戻ることができない。
・それに対して、addToBackStack() を呼び出した場合、フラグメントは停止され、ユーザーが戻る場合に、再開される。

それぞれのフラグメントトランザクションに、setTransaction() を コミット前に呼ぶことによって、トランザクションアニメーションを適用できる。

・commit() を呼び出すことは、トランザクションを直ちに実行することではない。
・性格には、アクティビティの UI スレッドに直ちに実行するようにスケジュールする。
・しかしながら必要な場合、UI スレッドから直ちにトランザクションの commit() を実行するように、executePendingTransaction() を呼ぶことができる。
・通常これは、トランザクションが、他のスレッドに依存していないかぎり、必要ではない。

注意 :commit() でトランザクションをコミットできるのは、アクティビティの保存時まで。この時点を過ぎて commit() を試みると例外となる。なぜなら、コミット後の状態はアクティビティがリストアするときに失われてしまう

ため。