G’s Academy Tokyo Pro Android デベロッパーコース

目次

G's ACADEMY TOKYO

- TableLayoutとViewPagerを使う。（タブメニュー毎のFragment（画面）をスワイプで切り替える。）
- Universal Image Loaderを使い、インターネット上の画像を表示する。
- SQLiteを利用する。
- SimpleCropViewを使い、画像の切り抜きを実装できる。
- floating-action-buttonを使い、お洒落なポップアップでメニュー表示を実装する。
- StickyHeaderを使い、Instagramのようなリスト表示を実装する。

TabLayoutとViewPagerを使う。

TabLayoutとViewPagerを使う。

G's ACADEMY TOKYO

TabLayoutとViewPagerについて

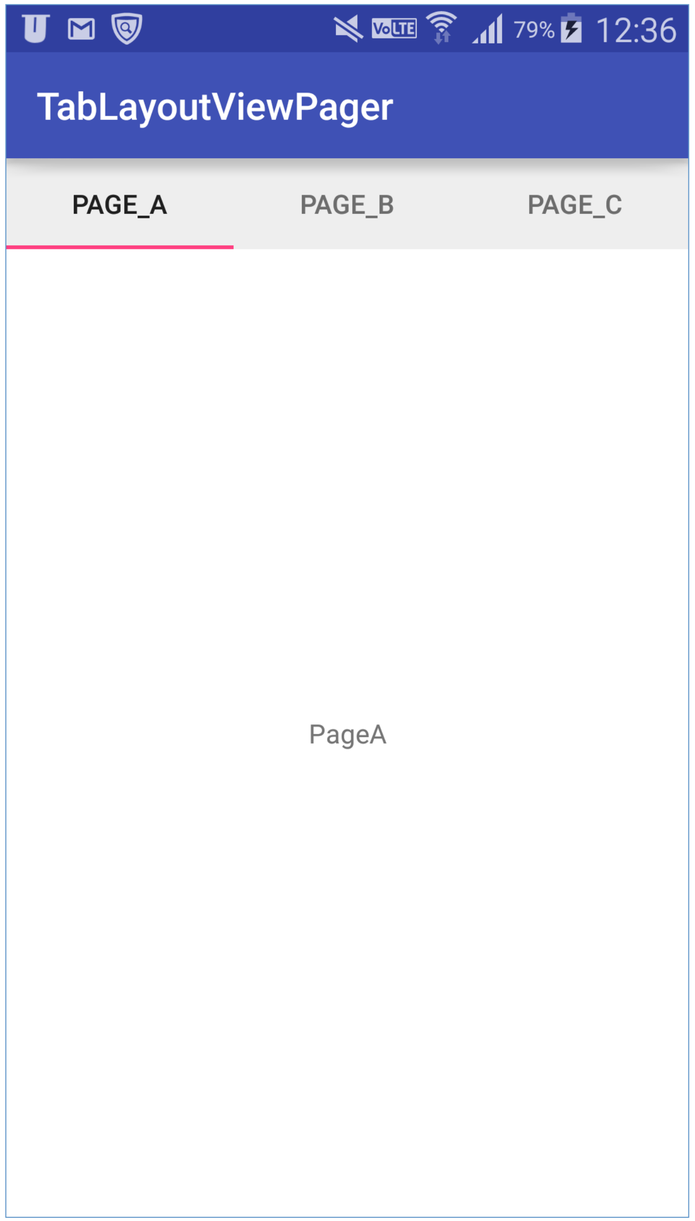
・Design Support Libraryで提供されているViewモジュール。

・タブメニュー毎の画面を左右のスワイプで切り替えることが出来る。
・タブメニュー毎の画面をFragment(部品)として、スワイプで切り替える。

・タブメニューは、TabLayoutのウィジェット。
・タブメニュー毎のFragment(部品)の スワイプ切り替えは、ViewPager。
・TabLayoutは、ViewPagerとの組み合わせで使うことが想定されている。

（参考）
http://qiita.com/furu8ma/items/1602a4bbed4303fec5b1
http://tiro105.hateblo.jp/entry/2015/08/25/235108

Line/Twitter/Facebook のようなUI/UXが実装出来る。
よく使われるUI/UXのパターンの一つ。



TabLayoutとViewPagerを使う。

G's ACADEMY TOKYO

オープンライブリーであるTabLayoutとViewPagerモジュールをロードするように、
build.gradleファイルに設定する。

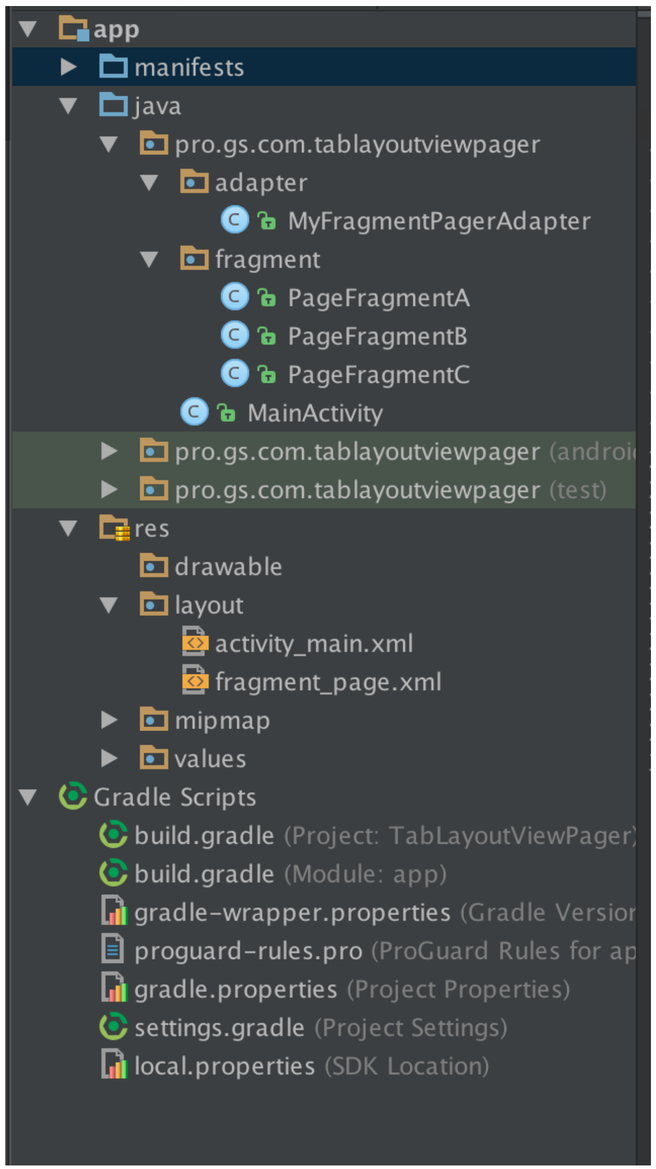
ビルドファイル build.gradle にて以下を記述します。

```
dependencies {
        compile fileTree(dir: 'libs', include: ['*.jar'])
        compile 'com.android.support:appcompat-v7:21.0.3'
        compile 'com.android.support:design:23.+'
}
```

TabLayoutとViewPagerを使う。

G's ACADEMY TOKYO

ファイル構成



- MainActivity.java
  メインの*Activity*クラス

- MyFragmentPagerAdapter.java
  ViewPagerのアダプタークラス
  ViewPagerウィジェットとfragmentを紐付ける。

- PageFragmentA.java
  ViewPagerの各画面に配置させるFragmentクラス
  （ここではPageA画面）

- PageFragmentB.java
  ViewPagerの各画面に配置させるFragmentクラス
  （ここではPageB画面）

- PageFragmentC.java
  ViewPagerの各画面に配置させるFragmentクラス
  （ここではPageC画面）

- *activity_main.xml*
  メイン*Activity*クラスのレイアウト

- fragment_page.xml
  タブメニュー毎のFragment(画面) のレイアウト

TabLayoutとViewPagerを使う。

G's ACADEMY TOKYO

ViewPagerオブジェクトとTabLayoutオブジェクトを表示する。

＜MainActivity.java＞

```
//ViewPagerのオブジェクトを取り出す
ViewPager viewPager = (ViewPager) findViewById(R.id.viewpager);

//ViewPagerウィジェットとfragmentを紐付けている。
viewPager.setAdapter(new MyFragmentPagerAdapter(getSupportFragmentManager()));

// TabLayoutのオブジェクトを取り出す
TabLayout tabLayout = (TabLayout) findViewById(R.id.sliding_tabs);

//タブメニューのUIを完成させる。
tabLayout.setupWithViewPager(viewPager);

//TabLayoutでタブを均等に配置する
tabLayout.setTabMode(TabLayout.MODE_FIXED);
tabLayout.setTabGravity(TabLayout.GRAVITY_FILL);
```

TabLayoutとViewPagerを使う。

G's ACADEMY TOKYO

レイアウトファイルにて、TabLayoutとViewPagerのウィジェットを記述する。

＜*activity_main.xml*＞

```
<android.support.design.widget.TabLayout
    android:id="@+id/sliding_tabs"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:tabMode="scrollable" />

<android.support.v4.view.ViewPager
    android:id="@+id/viewpager"
    android:layout_width="match_parent"
    android:layout_height="0px"
    android:layout_weight="1"
    android:background="@android:color/white" />
```

TabLayoutとViewPagerを使う。

G's ACADEMY TOKYO

ViewPagerのアダプタークラスを作成する。

＊ViewPagerウィジェットとfragmentを紐付ける。

＜MyFragmentPagerAdapter .java＞

```
public class MyFragmentPagerAdapter extends FragmentPagerAdapter {
    private String tabTitles[] = new String[] { ”PageA", "PageB", "PageC" };
    public MyFragmentPagerAdapter(FragmentManager fm) {
        super(fm);
    }

    @Override
    public int getCount() {
        return tabTitles.length;
    }

    @Override
    public Fragment getItem(int position) {
        if (position == 0) {
            return new PageFragmentA();
        } else if (position == 1) {
            return new PageFragmentB();
        } else {
            return new PageFragmentC();
        }
    }

    @Override
    public CharSequence getPageTitle(int position) {
        return tabTitles[position];
    }
}
```

TabLayoutとViewPagerを使う。

タブメニュー毎のFragment(画面) を作成する。

＜PageFragmentA .java＞

```
public class PageFragmentA extends Fragment {

    private String mPage;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mPage = “PageA”;
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,Bundle savedInstanceState) {
         //fragment_pageのレイアウトのオブジェクト自体が、TextViewのオブジェクト。暗黙的にキャスト（ダウンキャスト）が行われてる。
        View view = inflater.inflate(R.layout.fragment_page, container, false);
         TextView textView = (TextView) view;//TextView型へダウンキャストされている
        textView.setText(mPage);
        return view;
    }
}
```

＊この例では、３つのタブメニューとFragment(画面)があるので、３つのFragmentを作成する。
MyFragmentPagerAdapter .javaのgetItemメソッドで、
PageFragmentAとPageFragmentBとPageFragmentCの３つオブジェクトを返しているので、
これらの名前でFragmentを継承したクラスを３つ作成する。

TabLayoutとViewPagerを使う。

タブメニュー毎のFragment(画面) のレイアウトを作成する。

＜fragment_page.java＞

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

＊各Fragmentクラス（今回はPageFragmentA /PageFragmentB / PageFragmentAC ）で、
定義しているFragmentのレイアウトファイルをonCreateViewメソッドで指定しているので、
そのレイアウトファイルを作成する。

Universal Image Loaderを使い、インターネット上の画像を表示する。

Universal Image Loaderを使う

G's ACADEMY TOKYO

Universal Image Loaderを読み込む。

ビルドファイル build.gradle にて以下を記述します。

```
dependencies {
        compile fileTree(dir: 'libs', include: ['*.jar'])
        compile 'com.android.support:appcompat-v7:21.0.3'
        compile 'com.nostra13.universalimageloader:universal-image-loader:1.9.5’
}
```

Universal Image Loaderを使う

Universal Image Loaderについて

インターネット上の画像をダウンロードして表示する機能を簡単にできる。
非同期で読み込んでくれるライブラリ。
Volleyとほぼ同じだが、Universal Image Loaderの方が画像のローディングにより特化している。

Volleyと同様にメモリーキャッシュ/ディスクキャッシュも行ってくれる。
（ネット上の画像をロードして表示したい場合は、
Volleyを利用した場合もUniversal Image Loaderを利用した場合もパフォーマンスはほぼ変わらない。

・Android 2.0以降から使える
・導入が簡単
・非同期で画像を読み込み可能
・ディスクキャッシュやメモリキャッシュも簡単実装
・キャッシュする時はディレクトリの容量やファイル数で制限できる
・読み込み時のリスナーをセットできるので、失敗時の画像を表示したりも可能
・表示時の画像の拡大・縮小もやってくれる
・割りと細かいカスタマイズが出来る (Bitmap変換とかそこらへん)

Volleyだとロードした画像のBitmap変換などが簡単に出来なかった。
（やり方はあるのだろうけど・・）

（参考）
http://qiita.com/A-Ota/items/96e0905bcc4f58ad663c
http://qiita.com/chuross/items/e3ca79065d9b67716ace

Universal Image Loaderを使う

G's ACADEMY TOKYO

割りと細かいカスタマイズが出来る (Bitmap変換とかそこらへん)

ネットからロードした画像をリサイズしたり、フィルター加工などをする。
そしてまた、その加工した画像データをマイサーバーに送信して、保存したい場合が
画像を扱うアプリにはよくある。

その際、画像Bitmap型のデータに変換する必要がある。

Volleyはネットから画像をロードして、ImageViewに表示することは簡単に出来る。
画像データをBitmapに変換された状態で取り出すのに、Volleyでは苦労する。
（竹野の所感です！少なくてもvolleyでは、簡単には出来なそう・・）

Universal Image Loaderについて

```
final ImageView imageView = (ImageView) findViewById(R.id.imageView);

ImageLoader loader = ImageLoader.getInstance();

ImageLoaderConfiguration.Builder config = new ImageLoaderConfiguration.Builder(this);
loader.init(config.build());

// loadImageを使う場合
//Bitmapを取得出来る。
loader.loadImage(url, new SimpleImageLoadingListener() {

        //filter加工したり、リサイズ処理をしたい場合は、ここで取り出したBitmap画像に対して行う。
        @Override
        public void onLoadingComplete(String imageUri, View view, Bitmap loadedImage) {
                imageView.setImageBitmap(loadedImage);
    }
});

// displayImageを使う場合
//Bitmapは取得出来ない。表示のみ。
loader.displayImage(url, imageView);
```

SQLiteを使う。

SQLiteを使う。

SQLiteはMySQLなどと同じリレーショナルデータベースですが、
サーバとして動作させるのではなく、
単独のアプリケーション（アプリ側の組み込みDB）として動く。

SimpleCropViewを使い、画像の切り抜きを実装できる。

（参考）
http://qiita.com/issei_aoki/items/810f491da2e3d077b478

floating-action-buttonを使い、お洒落なメニュー表示を実装する。

StickyHeaderを使い、Instagramのような
リスト表示を実装する。