**22.3 프래그먼트**

프래그먼트는 액티비티그룹의 자식 액티비티 기능을 모두 가진다. 다만 액티비티는 독립적으로도 실행될 수 있지만 프래그먼트는 액티비티에 포함된 형태로만 실행되는 점이 다르다. 그렇다고 그 차이점이 단점이라고 볼 수 없다. 그 이유는 안드로이드의 실행단위는 오직 액티비티, 리시버, 서비스, 프로바이더 이렇게 4가지 컴포넌트기 때문이다. 예전 방식인 액티비티가 액티비티그룹에 포함된 경우 안드로이드 컴포넌트로써 기능을 상실해버리는 것은 오히려 애매모호하다.

우리에게 필요한 기능은 액티비티의 레이아웃을 여러 개로 분리하고 별도로 관리할 수 있는 구조다. 따라서 프래그먼트는 그 요구에 딱 맞는 기술이라 볼 수 있다. 그리고 정확한 목적에 맞춰진 프래그먼트는 그 범위 안에서 액티비티그룹 보다 더 많은 기능들이 존재한다. 가장 대표적인 기능은 액티비티그룹에는 없었던 백스택이다.

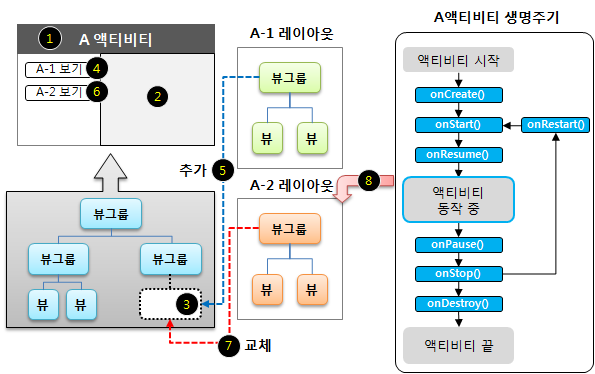
**22.3.1 시작에 앞서 프래그먼트를 우습게 보자.**

프래그먼트는 안드로이드 API 11 허니콤부터 나왔다. 플랫폼 버전으로 보면 API 10은 2.3.4고 API11은 3.0이다. 즉 메이저major 버전이 바뀐 것이다. 따라서 기존에 안드로이드 개발자들은 뭔가 새롭고 진보된 대단한 기능이라 생각했다. 하지만 앱을 개발함에 있어 꼭 활용해야 할 필요성을 느낄 수 없었을 것이다. 그 이유는 먼저 개발자들이 새로운 기술을 익힘에 있어 귀차니즘 때문일 것이다. 하지만 반드시 필요한 기능이라면 귀차니즘을 극복하고 활용할 수밖에 없었을 것이다. 더 큰 이유는 기존의 방식으로 개발하더라도 크게 불편함이 없었기 때문이다. 사실 프래그먼트는 레이아웃을 구성하는 뷰와 뷰그룹을 정확히 이해하고 활용할 수 있다면 다른 방법으로도 쉽게 대체될 수 있다. 이뿐만 아니라 직접 프래그먼트와 유사한 클래스를 만들 수도 있다. 그래서 선뜻 개발자들은 프래그먼트에 손을 대지 않았다. 그리고 안드로이드 메이저 버전이 바뀌면서 나온 프래그먼트를 어렵게 느껴지기 조차 한다.

*프래그먼트를 우습게 보자.*

저자가 프래그먼트를 설명하기 전에 가장 먼저 하고 싶은 말이다. 즉 어렵게 생각하지 말자는 의미다. 알고 보면 완전히 새로운 기술도 아니라 기존의 액티비티와 레이아웃을 적절히 이용해서 활용한 것뿐이다. 무슨 말인지 다음의 그림을 통해 이해해보자.

그림 22-17 프래그먼트가 없다면



①     A액티비티은 좌/우측으로 분리된 레이아웃인데 좌측 영역은 우측 영역의 레이아웃을 동적으로 변경하는 구조다.

②     따라서 우측 영역은 레이아웃이 가변적으로 변한다.

③     A액티비티의 뷰 구조를 살펴보면 ② 영역에 레이아웃을 추가 및 변경할 수 있도록 비워진 상태다.

④     만일 실행 중에 A액티비티 좌측 레이아웃에서 ‘A-1 보기’ 버튼을 누르면

⑤     A-1 레이아웃을 생성inflate시키고 ③ 영역에 추가하면 된다.

⑥     그리고 A액티비티 좌측 레이아웃에서 ‘A-2 보기’ 버튼을 누르면

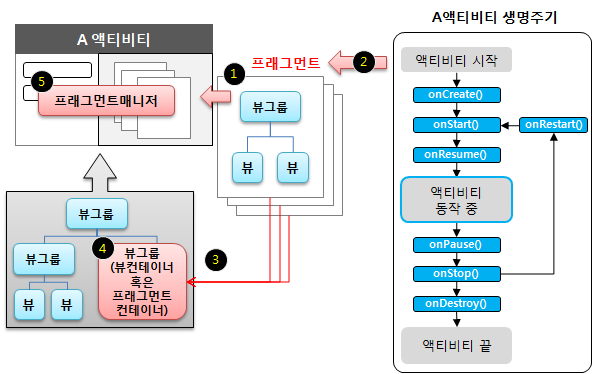
⑦     A-2 레이아웃을 생성시키고 ③ 영역에 추가된 A-1 레이아웃과 교체하면 된다.

⑧     그리고 A액티비티는 생명주기 함수가 호출될 때, 현재 활성화된 레이아웃의 생명주기 함수를 호출해준다. 레이아웃이 생명주기가 필요한 이유는 레이아웃 스스로의 상태를 관리하기 위해서다. 하지만 레이아웃은 뷰로 구성되어 있기 때문에 생명주기 함수가 없다. 따라서 레이아웃을 감싸는 별도의 클래스를 만들고 해당 클래스에 생명주기 함수를 구현해두면 되겠다.

[ REF \_Ref397011153 \r \h 그림 22-17]을 통해 프래그먼트를 사용하지 않고도 액티비티의 레이아웃을 분리하고 관리할 수 있었다. 물론 [ REF \_Ref397011153 \r \h 그림 22-17]은 추상적인 얘기므로 이해되지 않아도 상관없다. 다만 프래그먼트는 독립적인 레이아웃 조각으로 분리하고 관리해야 하는 환경에서 도움을 주는 기능이지 안드로이드의 전혀 새로운 개념이 아니라는 것만 기억하자.

이제 [ REF \_Ref397011153 \r \h 그림 22-17]의 구조에 프래그먼트 적용을 [ REF \_Ref397011686 \r \h 그림 22-18]의 그림으로 설명하는데, 프래그먼트 관련 용어도 같이 배워보도록 한다.

그림 22-18 프래그먼트 적용과 프래그먼트 관련 용어 정리



①     [ REF \_Ref397011153 \r \h 그림 22-17] ⑧에서 레이아웃을 감싸는 클래스가 바로 프래그먼트다.

②     프래그먼트는 내부적으로 생명주기 함수들이 존재하는데 프래그먼트가 포함된 액티비티는 자신의 생명주기 함수가 호출될 때, 동일한 프래그먼트 생명주기 함수도 호출해준다. 예를 들어 액티비티의 [안내]태그제한으로등록되지않습니다-onPause 생명주기 함수가 호출되었다면, 액티비티의 [안내]태그제한으로등록되지않습니다-onPause 함수에서 프래그먼트의 [안내]태그제한으로등록되지않습니다-onPause 함수를 이어서 호출하는 방식이다. 실제로 프래그먼트는 [안내]태그제한으로등록되지않습니다-onPause라는 생명주기 함수를 가진다. 그리고 프래그먼트는 액티비티와 같이 생명주기 함수를 통해 자신의 상태를 관리한다.

③     각 프래그먼트는 액티비티의 특정 뷰그룹내에 추가될 수 있다. 정확히 말하자면 프래그먼트가 가진 레이아웃을 추가한다는 의미다. 그리고 너무 당연한 것이지만 액티비티의 뷰그룹에는 여러 개의 프래그먼트를 추가할 수 있다. 뷰그룹은 여러 개의 자식 뷰를 추가할 수 있기 때문인데, 프래그먼트의 레이아웃도 자식 뷰에 해당한다.

④     뷰그룹은 일반적으로 컨테이너Container라고 불리기도 한다. 뷰그룹 내부에 여러 자식 뷰를 담을 수 있기 때문이다. 따라서 뷰그룹이 자식 뷰들을 포함할 때는 뷰 컨테이너라 하고, 프래그먼트를 포함할 때는 프래그먼트 컨테이너라고 부른다. 이러한 용어는 안드로이드 API를 사용하다 보면 함수의 매개변수나 개발자 사이트의 설명 글에서 빈번히 사용된다. 따라서 기억해두지 않으면 잘못 이해되기 쉽다. 그리고 뷰 컨테이너든, 프래그먼트 컨테이너든 여러분을 혼란스럽게 하려고 생겨난 것이 아니라, 좀더 쉽게 이해시키기 위해 사용되는 것이다.

⑤     액티비티 내에는 프래그먼트매니저가 존재하는데, 프래그먼트 추가, 삭제, 변경 등을 담당한다.

마지막으로 질문 한가지를 던져본다.

*프래그먼트를 활용하지 않고 조금 불편하더라도 예전 방식을 고집해도 되나?*

아쉽게도 이젠 그럴 수 없다. 안드로이드는 버전이 올라갈수록 여러 가지 편리한 뷰/뷰그룹, 어댑터 등의 클래스를 추가적으로 제공하는데 해당 클래스에서 프래그먼트를 요구하는 경우가 많다. 좀더 쉽게 예를 들자면 A라는 뷰그룹이 생겼는데 자식뷰로 프래그먼트 형태만 지원하는 경우다. 참고로 그중 안드로이드 API 11버전부터 프래그먼트와 함께 등장한 액션바가 대표적이다.

어쨌든 피할 수 없다면 프래그먼트를 즐겨라. 분명한 것은 개발자들을 괴롭히려고 등장한 것이 아니라 돕기 위해 생겨난 것이다.

**22.3.2 프래그먼트를 사용해보자**

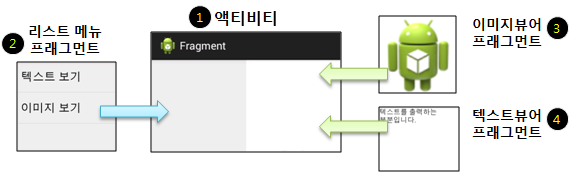
직접 예제를 구현하면서 프래그먼트 사용법을 익혀보자. 좀더 이해를 높히기 위해 [ REF \_Ref397104062 \r \h 그림 22-19]와 같이 액티비티그룹 예제와 동일한 구조의 앱을 구현해본다.

그림 22-19 프래그먼트 예제 미리보기



[ REF \_Ref397104062 \r \h 그림 22-19]에서 사용될 프래그먼트는 다음과 같이 총 세가지다.

그림 22-20 예제에서 사용되는 프래그먼트들



①     프래그먼트를 포함할 액티비티다. 액티비티의 레이아웃은 수평 배치 LinearLayout이며, 좌/우측으로 두 가지 자식뷰를 가진다.

②     액티비티 좌측 영역에는 리스트 메뉴 프래그먼트가 존재한다.

③     액티비티 우측 영역에는 이미지뷰어 혹은 텍스트뷰어 프래그먼트가 존재한다. 따라서 총 세 가지의 프래그먼트가 필요하다.

먼저 프래그먼트 예제의 AndroidManifest.xml을 작성해보자.

예제 22-19 프래그먼트 예제의 AndroidManifest.xml

|  |
| --- |
| **AndroidManifest.xml**    <?xml version=*"1.0"* encoding=*"utf-8"*?>  <manifest xmlns:android=*"http://schemas.android.com/apk/res/android"*      package=*"com.superdroid.fragment"*      android:versionCode=*"1"*      android:versionName=*"1.0"* >        <uses-sdk          android:minSdkVersion=*"16"*          android:targetSdkVersion=*"16"* />        <application          android:allowBackup=*"true"*          android:icon=*"@drawable/ic\_launcher"*          android:label=*"@string/app\_name"*          android:theme=*"@style/AppTheme"* >  **<activity android:name=*".MainActivity"***              android:label=*"@string/app\_name"* >              <intent-filter>                  <action android:name=*"android.intent.action.MAIN"* />                  <category android:name=*"android.intent.category.LAUNCHER"* />              </intent-filter>          </activity>      </application>    </manifest> |

[예제 22-19]는 간단히 앱에서 사용될 액티비티 하나만 정의하고 있다. 다음은 액티비티에 포함될 세 가지 프래그먼트의 레이아웃을 구현하자. 22.2절 액티비티그룹에서 사용된 레이아웃과 동일하기 때문에 상세한 설명은 생략한다.

예제 22-20 메뉴 리스트 프래그먼트 레이아웃

|  |
| --- |
| **res/layout/fragment\_menu\_list.xml**    <ListView xmlns:android=*"http://schemas.android.com/apk/res/android"*      android:id=*"@+id/listview"*      android:layout\_width=*"match\_parent"*      android:layout\_height=*"match\_parent"*  **android:entries=*"@array/IDSA\_LIST\_ITEM"***      android:background=*"#EEE"*/> |

예제 22-21 메뉴 리스트뷰의 아이템 명으로 사용될 문자열 배열 리소스

|  |
| --- |
| **res/values/string\_arrays.xml**    <resources>      <string-array name=*"IDSA\_LIST\_ITEM"*>          <item>텍스트 보기</item>          <item>이미지 보기</item>      </string-array>  </resources> |

예제 22-22 텍스트뷰어 프래그먼트 레이아웃

|  |
| --- |
| **res/layout/fragment\_text\_viewer.xml**    <TextView xmlns:android=*"http://schemas.android.com/apk/res/android"*      android:layout\_width=*"match\_parent"*      android:layout\_height=*"match\_parent"*      android:text=*"텍스트를 출력하는 부분입니다."*/> |

예제 22-23 이미지뷰어 프래그먼트 레이아웃

|  |
| --- |
| **res/layout/fragment\_image\_viewer.xml**    <ImageView xmlns:android=*"http://schemas.android.com/apk/res/android"*      android:layout\_width=*"match\_parent"*      android:layout\_height=*"match\_parent"*      android:src=*"@drawable/ic\_launcher"*/> |

다음은 세 가지 프래그먼트를 구현한다. 가장 먼저 텍스트뷰어 프래그먼트부터 구현하자.

예제 22-24 텍스트뷰어 프래그먼트

|  |
| --- |
| **src/TextViewerFragment.java**    // ① 텍스트 뷰어 프래그먼트 객체를 생성하는 함수다.  **public** **class** TextViewerFragment **extends Fragment** {        // ② 텍스트 뷰어 프래그먼트 객체를 생성하는 함수다.  **public static TextViewerFragment newInstance()**      {          TextViewerFragment f = **new** TextViewerFragment();    **return** f;      }        // ③ 부모 액티비티는 해당 프래그먼트를 구동하고, 액티비티에 추가될 프래그먼트의      //    레이아웃을 onCreateView 함수의 반환값으로 요구한다.      //    따라서 해당 프래그먼트는 onCreateView 재정의 함수에서 자신의 레이아웃을      //    생성하고 반환한다.      @Override  **public View onCreateView( LayoutInflater inflater,**  **ViewGroup container,**  **Bundle savedInstanceState )**      {  **return** inflater.inflate( R.layout.*fragment\_text\_viewer*,                                   container,  **false**);      }  } |

①     프래그먼트를 상속받아 클래스를 구현한다. 이로써 해당 클래스는 프래그먼트의 역할을 할 수 있게 된다.

②     액티비티 등에서 간단히 해당 프래그먼트 객체를 생성하기 위한 newInstance 함수를 구현해둔다. 이 함수는 자신의 객체를 생성하여 반환한다. 일반적으로 사용되는 방법이니 기억해두자.

③     onCreateView 함수는 프래그먼트의 생명주기 함수 중 하나다. 이 함수의 프래그먼트를 액티비티의 레이아웃에 추가할 때 호출되는데, 프래그먼트가 가진 레이아웃을 액티비티에 제공하는 역할을 한다. 따라서 프래그먼트를 구현할 때 필수로 onCreateView 함수를 재정의하고, 프래그먼트의 레이아웃을 생성 및 반환해주어야 한다.

다음은 이미지뷰어 프래그먼트를 구현한다. 구현 과정은 텍스트뷰어 프래그먼트와 차이가 없으므로 상세한 설명은 생략한다.

예제 22-25 이미지뷰어 프래그먼트

|  |
| --- |
| **src/ImageViewerFragment.java**    **public** **class** ImageViewerFragment **extends** Fragment {        // ① 이미지 뷰어 프래그먼트 객체를 생성하는 함수다.  **public** **static** ImageViewerFragment newInstance()      {          ImageViewerFragment f = **new** ImageViewerFragment();    **return** f;      }        // ② 부모 액티비티는 해당 프래그먼트를 구동하고, 액티비티에 추가될 프래그먼트의      //    레이아웃을 onCreateView 함수의 반환값으로 요구한다.      //    따라서 해당 프래그먼트는 onCreateView 재정의 함수에서 자신의 레이아웃을      //    생성하고 반환한다.      @Override  **public** View onCreateView( LayoutInflater inflater,                                ViewGroup container,                                Bundle savedInstanceState )      {  **return** inflater.inflate( R.layout.*fragment\_image\_viewer*,                                   container,  **false**);      }  } |

다음은 리스트 메뉴 프래그먼트를 구현해본다. 해당 프래그먼트에서 중점적으로 봐야 것은 바로 아이템 클릭 리스너다. 이는 액티비티그룹의 [예제 22-17]에서 설명된 것과 같이 프래그먼트의 독립성을 위해 리스너를 제공한다. 구현된 소스를 살펴보면서 이해하자.

예제 22-26 리스트 메뉴 프래그먼트

|  |
| --- |
| **src/ListMenuFragment.java**    **public** **class** ListMenuFragment **extends Fragment** **implements** OnItemClickListener  {      ListView                mListView = **null**;      OnListItemClickListener mListItemClickListener = **null**;    **public** **static** **final** **int** *ITEM\_TYPE\_TEXT\_VIEWER* = 1;  **public** **static** **final** **int** *ITEM\_TYPE\_IMAGE\_VIEWER* = 2;        @Override  **public** View onCreateView( LayoutInflater inflater,                                ViewGroup container,                                Bundle savedInstanceState )      {          View v = inflater.inflate( R.layout.*fragment\_menu\_list*,                                     container,  **false**);            mListView = (ListView)v.findViewById( R.id.*listview* );            // ① 리스트 클릭 리스너를 설정한다.  **mListView.setOnItemClickListener( this );**    **return** v;      }        // ② 해당 프래그먼트를 사용하는 액티비티에서 리스트 아이템 클릭시 그 이벤트를      //    수신받기 위한 리스너 인터페이스를 정의한다.      //    이 리스너를 통해 해당 프래그먼트는 독립적으로 동작될 수 있다.  **public interface OnListItemClickListener**  **{**  **void onItemClick( int itemType );**  **}**        // ③ 외부에서 해당 프래그먼트에서 발생되는 리스트 아이템 클릭 이벤트 수신받기 위한      //    리스너를 등록함수다.  **public void setOnListItemClickListener(**  **OnListItemClickListener ln )**      {          mListItemClickListener = ln;      }        @Override  **public** **void** onItemClick( AdapterView<?> parent,                               View view,  **int** position,  **long** id )      {  **int** itemType = 0;    **switch**( position )          {  **case** 0: itemType = *ITEM\_TYPE\_TEXT\_VIEWER*; **break**;  **case** 1: itemType = *ITEM\_TYPE\_IMAGE\_VIEWER*; **break**;          }            // ④ 해당 프래그먼트에서 아이템 클릭 이벤트가 발생되면 등록된 리스너의 핸들러          //    함수를 호출해준다. 여기서 핸드러의 인자로 선택된 아이템 타입을          //    전달함으로써 리스너를 등록한 액티비티가 클릭된 아이템을 구분할 수 있다.  **if**( mListItemClickListener != **null** )          {  **mListItemClickListener.onItemClick( itemType );**          }      }  } |

①     리스트 메뉴 프래그먼트는 내부적으로 리스트뷰를 가진다. 그리고 리스트뷰의 아이템을 클릭하면 액티비티 레이아웃 우측 영역에 프래그먼트를 변경한다. 따라서 우선 리스트뷰의 아이템이 클릭 이벤트를 받기 리스너를 등록한다. 참고로 아이템 클릭 리스너는 OnItemClickListener를 상속은 프래그먼트 클래스 내부에 구현되었다. 구현된 onItemClick 리스너 핸들러 함수는 ④에서 설명한다.

②     리스트뷰에서 선택된 프래그먼트는 해당 프래그먼트가 직접 제어해서는 안된다. 그 이유는 해당 프래그먼트가 다른 프래그먼트와 관계가 생기기 때문이다. 따라서 반드시 부모 액티비티에게 그 역할을 넘기고, 리스트 메뉴 프래그먼트는 부모 액티비티가 처리할 수 있도록 리스트 아이템 클릭 리스너만 제공하는 것이 좋다. 이를 위해 리스트 아이템 클릭 리스너 인터페이스를 구현해둔다.

③     이 함수는 ②에서 구현된 리스트 아이템 클릭 리스너를 등록하는 함수다. 해당 함수를 통해 부모 액티비티는 리스트 아이템 클릭 리스너를 구현 및 등록할 수 있다. ②, ③의 구현 내용은 프래그먼트의 독립성을 위해 일반적인 방법이기도 하다.

④     마지막으로 리스트뷰의 아이템이 클릭하면, 선택된 아이템 종류를 인자로 등록된 리스트 아이템 클릭 핸들러 함수를 호출해준다.

지금까지 액티비티에서 사용될 프래그먼트를 모두 구현했다. 이제 액티비티의 레이아웃과 액티비티 소스를 구현한다.

예제 22-27 액티비티 레이아웃

|  |
| --- |
| **res/layout/activity\_main.xml**    <LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*      android:layout\_width=*"match\_parent"*      android:layout\_height=*"match\_parent"*      android:orientation=*"horizontal"*>        <!-- ① 좌측 영역에 메뉴 리스트 프래그먼트를 설정한다. -->  **<fragment class=*"com.superdroid.fragment.ListMenuFragment"***  **android:id=*"@+id/menu\_fragment"***          android:layout\_width=*"0dp"*          android:layout\_height=*"match\_parent"*          android:layout\_weight=*"1"*/>        <!-- ② 우측 영역에 텍스트뷰어 혹은 이미지뷰어 프래그먼트를 포함할                프래그먼트 컨테이너다. -->  **<FrameLayout android:id=*"@+id/viewer\_fragment\_container"***          android:layout\_width=*"0dp"*          android:layout\_height=*"match\_parent"*          android:layout\_weight=*"1"*/>    </LinearLayout> |

①     액티비티 레이아웃의 좌측 영역에는 리스트 메뉴 프래그먼트가 배치된다. 이 영역은 여러 프래그먼트가 가변적으로 변하는 곳이 아니라, 오직 리스트 메뉴 프래그먼트만 배치된다. 이렇게 정적인 프래그먼트 영역은 레이아웃 내에 <fragment> 요소를 추가하고 class 속성 혹은 name 속성을 이용해서 직접 프래그먼트 클래스명을 적어주면 된다. 그리고 id 속성을 추가하여 액티비티 소스에서 프래그먼트를 참조할 수 있도록 ID를 등록한다.

②     액티비티 레이아웃의 우측 영역에는 여러 종류의 프래그먼트가 가변적으로 배치될 수 있다. 따라서 ①과 같이 정적인 특정 프래그먼트를 정의하지 않고 FrameLayout 뷰그룹을 배치한다. 이 뷰그룹에는 여러 종류의 프래그먼트를 바꿔가면서 배치하게 된다. 이렇게 여러 프래그먼트를 포함하는 뷰그룹을 프래그먼트 컨테이너라고 부른다. 프래그먼트 컨테이너는 레이아웃 XML에서 특정 프래그먼트를 설정하지 않고, 자바 소스에서 동적으로 프래그먼트를 추가한다.

|  |
| --- |
| **레이아웃 구조에서 뷰와 뷰그룹이 아닌 프래그먼트도 포함될 수 있나?**    [예제 22-27] ①을 보면 레이아웃 XML에는 뷰와 뷰그룹이 아닌 프래그먼트도 포함하고 있다. 이러한 레이아웃 형태로 뷰구조에 프래그먼트도 포함될 수 있다고 착각하는 경우가 있다. 하지만 이것은 큰 오해다. 화면은 오직 뷰/뷰그룹으로만 구성될 수 있고, 프래그먼트 자체는 뷰가 아니기 때문에 뷰그룹의 자식뷰로 추가될 수 없다. 여기서 <fragment> 요소가 뷰그룹에 포함된 것은 다만 [22.1.2]절의 <include> 요소와 같이 프래그먼트가 가진 레이아웃이 추가되는 것뿐이다. 안드로이드는 내부적으로 레이아웃 XML을 파싱할 때 <fragment> 요소를 만나면 해당 프래그먼트를 생성하고, 프래그먼트의 레이아웃을 <fragment> 요소 부분에 추가하게 된다. 즉 결국 레이아웃은 뷰와 뷰그룹만으로 구성되는 것이다. 별로 중요치 않은 내용이라고 생각할 수 있지만 안드로이드를 제대로 이해하려면 그냥 넘길 것이 결코 아니다.  이해를 위해 한가지 더 생각해보자. 안드로이드 앱은 오직 4대 콤포넌트인 액티비티, 리시버, 서비스, 프로바이더로 구성되어있다. 내가 어떤 기능을 구현하던 4가지 콤포넌트에 포함되어 있어야 한다는 말이다. 저자가 하고 싶은 말은 바로 기술적 한계다. 화면을 구성하는 것은 뷰/뷰그룹이고 다른 것은 있을 수 없다. 그리고 앱을 구성하는 것은 4대 콤포넌트고 다른 것은 있을 수 없다. 이렇게 정확한 한계를 알고 나면 안드로이드의 수많은 기술들이 복잡하게 느껴지지 않고, 자연스럽게 분류된다. 그리고 그 분류마다 정확한 역할이 있기 때문에 새로운 기술이 나오더라도 쉽게 이해가 된다. |

이제 마지막으로 액티비티 소스를 구현해보자. 액티비티에서는 여러 프래그먼트를 추가, 제거, 변경하는 등 관리가 주된 역할이다. 참고로 [예제 22-28]은 프래그먼트와 관련된 다양한 객체와 함수에 대한 설명이 생략되었다. 그 이유는 먼저 전반적인 프래그먼트 활용 구조를 익히면, 다음 상세한 설명이 쉽게 이해되기 때문이다.

예제 22-28 액티비티

|  |
| --- |
| **src/MainActivity.java**    **public** **class** MainActivity **extends** Activity {        @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);    // ① 액티비티에 사용될 레이아웃을 생성한다. 레이아웃 좌측 영역에 배치될  //    메뉴 리스트 프래그먼트는 액티비티 레이아웃 XML에 정적으로 정의되어  //    있어서, 액티비티 레이아웃 생성시 같이 생성 및 추가된다.  // ====================================================================  **setContentView(R.layout.*activity\_main*);**  // ====================================================================            // ② 액티비티 레이아웃 우측에 텍스트뷰어 프래그먼트를 추가한다.          // ====================================================================  **TextViewerFragment textViewerFragment =**  **TextViewerFragment.*newInstance*();**  **getFragmentManager()**  **.beginTransaction()**  **.add( R.id.*viewer\_fragment\_container*, textViewerFragment )**  **.commit();**          // ====================================================================            // ③ 액티비티 레이아웃 좌측에 리스트 메뉴 프래그먼트에서 아이템을 선택했을 때          //    이벤트를 처리하기 위한 리스너를 구현 및 등록한다.          // ====================================================================  **ListMenuFragment listMenuFragment = (ListMenuFragment)**  **getFragmentManager().findFragmentById(**  **R.id.*menu\_fragment* );**            listMenuFragment.**setOnListItemClickListener**(  **new** ListMenuFragment.OnListItemClickListener()          {              @Override  **public** **void** onItemClick( **int** itemType )              {  // ④ 액티비티 우측 영역 프래그먼트 컨테이너에 현재 보여지고 있는  //    프래그먼트를 참조한다. 만일 선택된 아이템이 현재 보여지고 있는  //    프래그먼트라면 아무 처리도 하지 않고 끝내고, 아니라면  //    보여줘야 할 프래그먼트를 생성해둔다.                  // ============================================================                  Fragment fragment = getFragmentManager().                              findFragmentById( R.id.*viewer\_fragment\_container* );    **if**( itemType == ListMenuFragment.*ITEM\_TYPE\_TEXT\_VIEWER*)                  {  **if**( fragment **instanceof** TextViewerFragment == **true** )                      {  **return**;                      }    **fragment = TextViewerFragment.*newInstance*();**                  }  **else** **if**( itemType == ListMenuFragment.*ITEM\_TYPE\_IMAGE\_VIEWER* )                  {  **if**( fragment **instanceof** ImageViewerFragment == **true** )                      {  **return**;                      }    **fragment = ImageViewerFragment.*newInstance*();**                  }  // ============================================================                    // ⑤ 선택된 아이템에 해당하는 프래그먼트를 액티비티 우측에 배치한다.                  // ============================================================                  getFragmentManager()                    .beginTransaction()  **.replace( R.id.*viewer\_fragment\_container*, fragment )**                    .commit();                  // ============================================================              }          });          // ====================================================================      }  } |

①     setContentView 함수를 통해 액티비티의 레이아웃을 생성한다. 여기서 액티비티 레이아웃 XML에 포함된 정적 프래그먼트와 프래그먼트가 가진 레이아웃도 같이 생성된다. 그리고 생성된 프래그먼트 레이아웃은 액티비티 레이아웃 좌측 영역에 추가된다. 따라서 레이아웃 좌측 영역에는 메뉴 리스트 프래그먼트 레이아웃이 포함되었다.

②     액티비티 레이아웃 우측 영역은 정적 프래그먼트가 존재하지 않기 때문에 빈 상태다. 따라서 최초 보여줄 프래그먼트를 동적으로 생성하여 추가해주어야 한다. 텍스트뷰어 프래그먼트를 생성 및 추가한다. 여기서 프래그먼트를 추가하기 위해 프래그먼트매니저를 사용한다. 프래그먼트매니저는 간단히 getFragmentManager 함수를 통해 참조할 수 있다. 그런데 프래그먼트는 프래그먼트매니저로 직접 추가하지 않고 프래그먼트 트랜잭션 객체를 이용하는데, 그 이유는 [ REF \_Ref398309821 \r \h 그림 22-21]에서 상세히 설명된다. 그리고 프래그먼트를 추가하는 프래그먼트 트랜잭션의 add 함수는 [ REF \_Ref398309917 \r \h 그림 22-22]에서 설명되니 일단 넘어가자.

③     다음은 ①에서 추가된 정적 메뉴 리스트 프래그먼트에 리스트 아이템 클릭 리스너를 등록해야 한다. 이를 위해 먼저 이미 추가된 메뉴 리스트 프래그먼트 객체가 필요한데, 프래그먼트 매니저의 findFragmentById를 이용하면 참조할 수 있다. findFragmentById의 인자는 참조하고 싶은 프래그먼트 ID를 넘겨주면 된다. 프래그먼트 ID에 대해서는 [ REF \_Ref398310483 \r \h 그림 22-26]에서 설명되니 넘어가자. 어쨌든 참조된 메뉴 리스트 프래그먼트의 setOnListItemClickListener 함수를 통해 리스트 아이템 클릭 리스너를 등록한다. 다음은 리스트 아이템 클릭 리스너 구현부를 살펴보자.

④     메뉴 리스트에서 특정 아이템을 클릭하면, 선택된 아이템 종류의 프래그먼트를 생성한다.

⑤     생성된 프래그먼트는 프래그먼트 트랜잭션의 replace 함수를 기존의 프래그먼트와 교체한다. 프래그먼트 교체는 [ REF \_Ref398311410 \r \h 그림 22-23]에서 상세히 설명한다.

예제를 실행하면 [ REF \_Ref398305213 \r \h 그림 22-19]와 같은 결과를 확인할 수 있다. 대략적인 프래그먼트 구조가 머릿속에 그려 졌는가! 이제 [예제 22-28]에서 사용된 프래그먼트 객체와 함수에 대해서 좀더 자세히 살펴보자.

[참고 예제 소스]

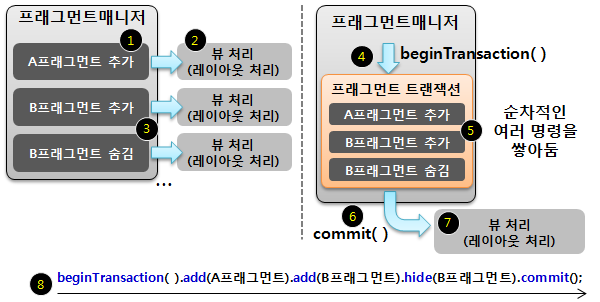
[[첨부파일](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D6%26fileid%3D12%26regdt%3D20140825193759&url=http%3A%2F%2Fcfile248.uf.daum.net%2Fattach%2F255FC23F5417D4B412FE29'))22-4. 프래그먼트 사용.zip](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D6%26fileid%3D12%26regdt%3D20140825193759&url=http%3A%2F%2Fcfile248.uf.daum.net%2Fattach%2F255FC23F5417D4B412FE29'))

**22.3.3 프래그먼트 트랜잭션과 프래그먼트 제어**

먼저 프래그먼트 트랜잭션에 대해서 살펴보자. 모든 프래그먼트의 관리는 프래그먼트매니저가 담당한다. 하지만 레이아웃에 프래그먼트의 추가, 제거, 교체 등으로 화면에 변화가 생기는 부분은 프래그먼트매니저가 직접하지 않고, 꼭 프래그먼트 트랜잭션FragmentTransaction을 통해야 한다. 프래그먼트 트랜잭션은 프래그먼트매니저의 beginTransaction 함수를 통해 참조할 수 있다.

그렇다면 왜 직접 프래그먼트매니저를 이용하지 않고 트랜잭션 객체를 통해 프래그먼트를 다루는지 살펴보자.

그림 22-21 beginTransaction과 commit 함수의 필요성



①     예를 들어 레이아웃에 A, B프래그먼트를 추가하고, 추가된 B프래그먼트는 겹쳐 보이지 않도록 잠시 숨긴다고 가정하자. 이는 연속적으로 처리되어야 한다. 먼저 프래그먼트매니저는 A프래그먼트를 추가할 것이다.

②     이후 프래그먼트매니저는 추가된 A프래그먼트를 화면에 보여주기 위해 내부적으로 여러 가지 작업을 하고, 이후 시스템은 화면에 A프래그먼트의 레이아웃이 화면에 보이게 된다.

③     다음 B프래그먼트를 추가하고, 숨기면 매번 프래그먼트의 레이아웃을 처리하고 시스템은 변경된 내용을 화면 계속 반영해야 한다. 하지만 이렇게 화면의 변화가 발생되는 연속된 작업은 성능을 떨어트리는 등 비효율적이다. 그렇다면 해결방법이 뭘까? 바로 연속된 작업을 모아 한꺼번에 처리하는 것이다. 이를 위해 프래그먼트 트랜잭션이 존재한다.

④     프래그먼트매니저의 beginTransaction 함수를 호출하면 연속될 작업을 모아둘 준비를 한다. 그리고 해당 함수는 프래그먼트 트랜잭션 객체를 반환하고, 이 객체를 통해 처리할 작업을 추가할 수 있다.

⑤     프래그먼트 트랜잭션 객체를 통해 순차적으로 처리해야 할 작업을 추가한다. 여기서 추가된 작업은 트랜잭션 객체가 쌓아둘 뿐 실제로 처리되지는 않는다. 참고로 프래그먼트 트랜잭션 객체 내부에는 작업을 쌓아두는 큐가 존재한다.

⑥     연속된 작업들의 추가가 끝나면 마지막으로 commit 함수를 호출한다.

⑦     commit 함수가 호출되면 프래그먼트매니저는 쌓여있는 작업을 한꺼번에 처리하고, 시스템은 처리된 결과를 화면에 반영하게 되는 것이다. 즉 프래그먼트 트랜잭션은 일련의 작업들을 모아 프래그먼트매니저가 한번에 처리할 수 있도록 도와주는 객체다.

⑧     그런데 재미있게도 프래그먼트 트랜잭션 관련 소스를 보면, 일련의 작업들을 시작부터 끝까지 순서대로 연결하여 함수를 호출하고 있다. 이는 연속된 작업을 소스로 표현하는데 있어 가독성을 높여 준다. 물론 각 함수들이 프래그먼트 트랜잭션 객체를 반환하기 때문에 가능하다. 이러한 방식을 전문적으로 빌더 패턴이라 부르는데, 자주 활용되는 패턴이니 기억해두면 좋다.

정리해보자면 프래그먼트 트랜잭션은 연속된 작업을 쌓아두는 역할을 할 뿐이다. 즉 쌓여있는 작업 단위는 프래그먼트 트랜잭션이고, 프래그먼트매니저는 프래그먼트 트랜잭션 단위로 처리한다.

다음은 프래그먼트를 처리하는 여러 가지 함수들을 살펴보자.

●  FragmentTransaction add(**int** containerViewId, Fragment fragment)

프래그먼트 추가

●   FragmentTransaction remove(Fragment fragment)

추가된 프래그먼트 삭제

●   FragmentTransaction replace(**int** containerViewId, Fragment fragment)

추가된 프래그먼트를 새로운 프래그먼트와 교체

●   FragmentTransaction detach(Fragment fragment)

추가된 프래그먼트의 레이아웃을 떼어냄

●   FragmentTransaction attach(Fragment fragment)

detach 함수로 떼어낸 프래그먼트 레이아웃을 다시 붙임

●   FragmentTransaction hide(Fragment fragment)

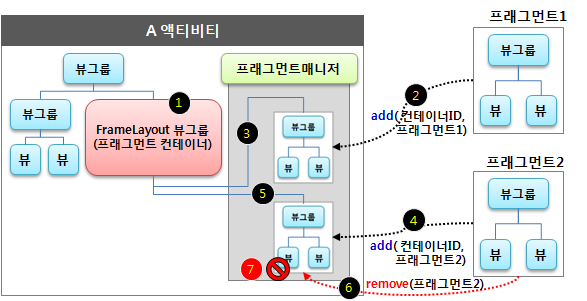
프래그먼트의 레이아웃을 숨김

●   FragmentTransaction show(Fragment fragment)

hide 함수로 숨겨진 프래그먼트 레이아웃을 다시 보여줌

각 함수에 대해서 상세히 살펴보자. 먼저 프래그먼트 추가, 삭제하는 add, remove 함수에 대해서 다음의 그림으로 설명한다.

그림 22-22 프래그먼트 추가, 삭제 함수



①     액티비티 레이아웃에 포함된 특정 뷰그룹에 프래그먼트 두 개를 추가해보자. 참고로 프래그먼트를 담는 뷰그룹은 프래그먼트 컨테이너라 부른다.

②     프래그먼트1을 생성하고 add 함수를 통해 컨테이너에 추가한다. add 함수는 두 개의 인자가 필요한데, 첫 번째는 프래그먼트가 추가될 컨테이너ID고 두 번째는 추가할 프래그먼트다.

③     이렇게 추가된 프래그먼트는 프래그먼트매니저가 관리한다. 그리고 프래그먼트의 레이아웃을 컨테이너의 자식뷰로 추가해준다.

④     다음 프래그먼트2 역시 add 함수를 통해 컨테이너로 추가하고

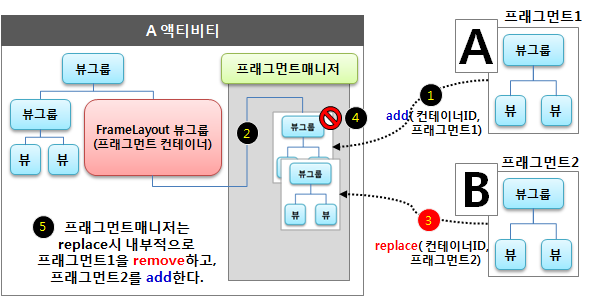
⑤     프래그먼트매니저는 프래그먼트2의 레이아웃을 컨테이너의 자식뷰로 추가한다. 따라서 화면에는 프래그먼트 두 개가 화면에 표시될 것이다.

⑥     다음은 추가된 프래그먼트2를 제거해보자. 이를 위해 remove 함수를 사용한다. remove 함수 인자는 제거하려는 프래그먼트를 넘겨주면 된다.

⑦     프래그먼트매니저는 자신이 관리하던 프래그먼트2를 제거한다. 따라서 화면에는 프래그먼트2의 레이아웃이 사라진다.

[ REF \_Ref398309917 \r \h 그림 22-22]에서 설명한 add, remove는 프래그먼트매니저에 프래그먼트를 추가 혹은 삭제하는 기능이다. 그리고 프래그먼트에는 레이아웃이 포함되어 있기 때문에 추가 혹은 삭제하면 화면에 프래그먼트 레이아웃도 나타나거나 사라지게 된다. 다음은 replace 함수에 대해서 살펴보자.

그림 22-23 프래그먼트 교체 함수



①     먼저 add 함수를 호출하여 프래그먼트1을 컨테이너에 추가한다.

②     추가된 프래그먼트는 프래그먼트매니저를 통해 레이아웃이 컨테이너의 자식뷰로 연결된다.

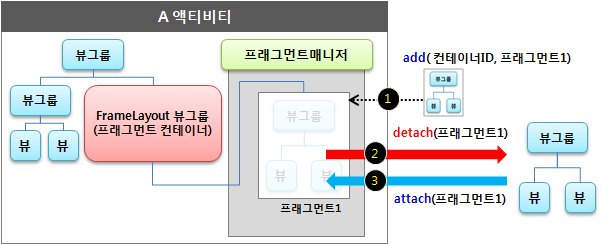
③     다음은 replace 함수로 프래그먼트1을 프래그먼트2로 교체한다. replace 함수도 add 함수와 같이 프래그먼트를 추가하는 기능이다. 따라서 함수 인자도 add 함수와 같다. 하지만 replace 함수로 추가된 프래그먼트는 컨테이너에 가장 마지막으로 추가된 프래그먼트를 제거하고 추가되는 것이 다르다.

④     따라서 마지막에 추가된 프래그먼트는 프래그먼트매니저가 관리하는 목록에서 제거된다.

⑤     사실 replace는 [ REF \_Ref398309917 \r \h 그림 22-22]에서 배웠던 add, remove 함수를 동시에 사용한 것과 같다. replace 내부적으로 마지막 추가된 프래그먼트를 remove하고, 새로운 프래그먼트를 add하기 때문이다.

다음은 설명할 것은 attach, detach 함수다. 이 두 가지 함수도 외형상으로는 add, remove와 같이 화면에 프래그먼트가 가진 레이아웃을 추가했다 제거하는 기능을 가진다. 하지만 분명한 차이가 있다. [ REF \_Ref398558139 \r \h 그림 22-24]를 통해 그 차이점을 살펴보자.

그림 22-24 프래그먼트 레이아웃을 화면에서 떼거나 붙이는 함수



①     먼저 add 함수를 통해 프래그먼트를 프래그먼트매니저에 추가한다. 추가된 프래그먼트는 프래그먼트매니저를 통해 레이아웃이 컨테이너에 연결된다. 사실 프래그먼트를 추가하는 것은 add 함수가 유일하다. 이 점을 기억하고 계속 살펴보자.

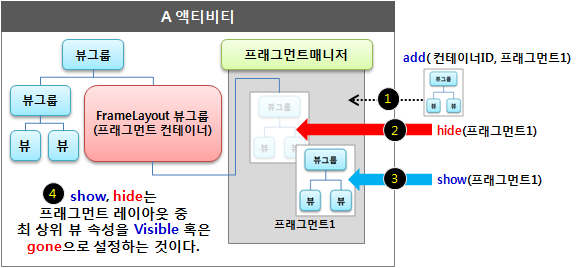
②     다음은 추가된 프래그먼트1을 화면에서 제거해보자. 앞서 배운 remove 함수가 아닌 detach 함수로 제거한다. 인자로는 제거할 프래그먼트1 객체를 전달하면 된다. 결과로 보면 프래그먼트1이 화면에서 사라지는 것은 같다. 하지만 프래그먼트매니저 내부적으로 detach 함수는 프래그먼트1 객체 자체를 제거하지는 않는다. 다만 프래그먼트1이 가진 레이아웃만 제거할 뿐이다.

③     따라서 attach 함수를 통해 다시 프래그먼트1을 추가하면, 이전 detach 함수로 제거된 레이아웃을 다시 생성하여 붙인다.

[ REF \_Ref398558139 \r \h 그림 22-24]에서 중요한 점은 detach, attach 함수의 경우 프래그먼트 자체를 추가하거나 제거하는 것이 아니라, 이미 추가된 프래그먼트의 레이아웃을 제거하거나 추가한다는 점이다. 따라서 detach, attach 함수는 반드시 add 함수로 프래그먼트가 추가된 후 사용할 수 있다. 그리고 attach는 detach 함수로 레이아웃이 제거된 상태여야 의미가 있다. 참고로 이러한 차이는 프래그먼트의 생명주기에서 확인할 수 있다. 즉 add, remove와 attach, detach가 될 경우 프래그먼트의 생명주기도 차이가 있다는 의미다. 프래그먼트 생명주기에 대해서는 [22.3.6]절에서 살펴보도록 한다.

다음은 설명할 것은 show, hide 함수다. 이 두 가지 함수도 외형상으로는 add, remove 혹은 attach, detach와 같이 화면에 프래그먼트가 가진 레이아웃이 나타났다 사라지는 것은 같다. 하지만 어떤 차이가 있는지 [ REF \_Ref398563128 \r \h 그림 22-25]를 통해 살펴보자.

그림 22-25 프래그먼트 레이아웃을 화면에서 보이거나 숨기는 함수



①     먼저 add 함수를 통해 프래그먼트를 프래그먼트매니저에 추가한다. 추가된 프래그먼트는 프래그먼트매니저를 통해 레이아웃이 컨테이너에 연결된다. show, hide 함수 역시 프래그먼트가 추가되지 않은 상태에서는 의미가 없다. 프래그먼트가 추가되었기 때문에 화면에는 추가된 프래그먼트의 레이아웃이 보일 것이다.

②     여기서 hide 함수를 호출하면 화면에 보이던 프래그먼트 레이아웃이 숨겨진다. hide 함수의 인자는 숨길 프래그먼트 객체다.

③     show 함수를 호출하면 숨겨진 레이아웃이 다시 보이게 된다.

④     사실 show, hide 함수는 프래그먼트 레이아웃의 최상위 뷰 속성값에 Visible 혹은 gone 속성값을 설정할 뿐이다. 따라서 프래그먼트매니저가 관리하는 프래그먼트가 추가/제거되거나 프래그먼트 레이아웃이 추가/제거되지 않는다.

지금까지 add, remove, replace와 attach, detach 그리고 show, hide 프래그먼트매니저 함수에 대해서 살펴보았다. 각 함수는 화면상 결과가 유사해서 정확한 원리를 모르면 잘못 활용하는 경우가 생긴다. 그렇다면 어떻게 활용해야 하는지 정리해보자.

●   **add, remove** : 최초 프래그먼트를 추가할 때에는 add 함수가 유일하다. 하지만 가변적으로 추가된 프래그먼트를 교체할 때에는 replace(혹은 remove + add) 함수를 써야 하는데 이때 고려해야 할 것이 있다. remove 함수는 프래그먼트 객체 자체가 제거된다는 점이다. 따라서 차후 다시 제거된 프래그먼트를 보여주려면 새로운 프래그먼트를 생성해야 한다. 그리고 새로 생성된 프래그먼트는 제거된 프래그먼트의 값들을 유지할 수 없다. 만일 프래그먼트의 값을 유지할 필요가 없고, 늘 새로운 내용을 반영해야 하는 경우라면 replace(혹은 remove + add)  함수를 사용하자.

●   **attach, detach** : 이 함수들은 추가된 프래그먼트의 레이아웃만 제거하거나 다시 추가한다. 따라서 레이아웃이 제거되었다 다시 보여질 때, 레이아웃 변경이 필요한 경우 사용되는 것이 좋다. 그리고 remove, add와 달리 이 함수들은 프래그먼트 객체 자체가 제거되지 않기 때문에 프래그먼트 자체에 값들은 유지된다.

●   **show, hide** : 이 함수는 프래그먼트나 프래그먼트가 가진 레이아웃을 제거하고 새로 추가하지 않는다. 순수하게 레이아웃만 숨기거나 보여주기 때문이다. 따라서 프래그먼트와 레이아웃의 설정은 그대로 재사용할 수 있다. 또한 제거 및 다시 생성 과정이 전혀 없기 때문에 시스템 부하가 거의 없다. 하지만 프래그먼트가 보여질 때 전혀 다른 내용을 반영해야 한다면 replace 혹은 attach, detach를 사용해야 할 것이다.

그렇다면 [예제 22-28]에서 사용했던 add와 replace 함수는 적절한 사용 예일까?

생각해보면 화면 우측 영역에 교체되면서 보여지는 프래그먼트는 레이아웃의 변경도 없고, 프래그먼트 객체 자체의 설정 변화도 없다. 단지 두 가지 프래그먼트 레이아웃이 숨겼다 보였다 할 뿐이다. 따라서 이 경우는 show, hide 함수를 사용하는 것이 맞다. 물론 최초 두 가지 프래그먼트를 보여줄 때는 add 함수를 사용해야 한다. 그렇다면 [예제 22-28]을 수정해보자.

예제 22-29 show, hide 프래그먼트 트랜잭션 함수로 수정 적용

|  |
| --- |
| **src/MainActivity.java**    **public** **class** MainActivity **extends** Activity {        TextViewerFragment  mTextViewerFragment = **null**;      ImageViewerFragment mImageViewerFragment = **null**;        @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);            setContentView(R.layout.*activity\_main*);            // ① 액티비티 레이아웃 우측에 나타날 수 있는 뷰어 프래그먼트를 미리 추가하고,        //    당장 보여지지 말아야 할 추가한 프래그먼트는 감춘다.          // ====================================================================          mTextViewerFragment = TextViewerFragment.*newInstance*();          mImageViewerFragment = ImageViewerFragment.*newInstance*();            getFragmentManager()            .beginTransaction()  **.add(R.id.*viewer\_fragment\_container*, mTextViewerFragment)**  **.add(R.id.*viewer\_fragment\_container*, mImageViewerFragment)**  **.hide( mImageViewerFragment )**            .commit();          // ====================================================================            ListMenuFragment listMenuFragment = (ListMenuFragment)                     getFragmentManager().findFragmentById( R.id.*menu\_fragment* );            listMenuFragment.setOnListItemClickListener(  **new** ListMenuFragment.OnListItemClickListener()          {              @Override  **public** **void** onItemClick( **int** itemType )              {                  // ② 당장 보여져야 할 프래그먼트는 show 하고, 보이지 말아야 할                  //    프래그먼트는 hide 한다.                  // =============================================================  **if**( itemType == ListMenuFragment.*ITEM\_TYPE\_TEXT\_VIEWER*)                  {                      getFragmentManager()                        .beginTransaction()  **.hide( mImageViewerFragment )**  **.show( mTextViewerFragment )**  .commit();                  }  **else** **if**( itemType == ListMenuFragment.*ITEM\_TYPE\_IMAGE\_VIEWER* )                  {                      getFragmentManager()                        .beginTransaction()  **.hide( mTextViewerFragment )**  **.show( mImageViewerFragment )**                      .commit();                  }                  //=============================================================              }          });      }  } |

①     미리 우측에 보여질 두 가지 뷰어 프래그먼트를 생성하고, 프래그먼트매니저에 추가한다. 하지만 이렇게 미리 추가하면 화면에는 두 가지 프래그먼트의 레이아웃이 컨테이너에 연결되기 때문에 겹쳐서 보여지게 된다. 따라서 당장 보이지 않아야 할 이미지뷰어는 hide 함수를 통해 숨긴다.

②     이제 메뉴 리스트의 아이템이 선택될 때, 보여져야 할 프래그먼트는 show 함수를 호출하여 보여주고, 숨겨야 할 프래그먼트는 hide 함수를 호출해 숨기면 된다.

수정된 [예제 22-29]를 실행하면 [예제 22-28]과 동일한 결과를 확인할 수 있다.

[참고 예제 소스]

[[첨부파일](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D6%26fileid%3D13%26regdt%3D20140825193759&url=http%3A%2F%2Fcfile267.uf.daum.net%2Fattach%2F265B043F5417D4CC181DE7'))22-5. 프래그먼트 수정(show, hide 함수 사용).zip](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D6%26fileid%3D13%26regdt%3D20140825193759&url=http%3A%2F%2Fcfile267.uf.daum.net%2Fattach%2F265B043F5417D4CC181DE7'))

**22.3.4 프래그먼트 ID와 TAG**

프래그먼트매니저에 여러 프래그먼트를 추가해서 사용하다보면, 프래그먼트매니저가 관리하는 특정 프래그먼트를 찾아 참고하고 싶을 때가 있다. 하지만 프래그먼트 이름이 없다면 찾고자하는 대상 프래그먼트를 지칭할 방법이 없다. 이를 위해 프래그먼트는 ID와 TAG라는 두 가지 방식의 이름을 설정할 수 있다. 하나씩 살펴보자.

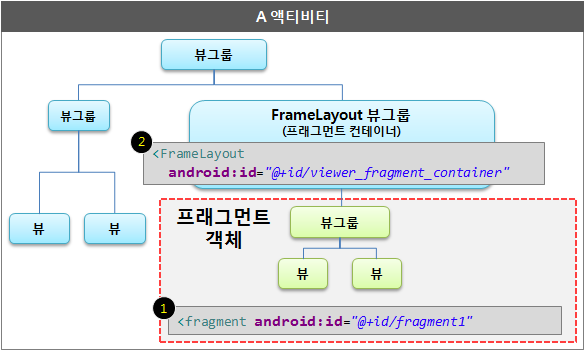
**프래그먼트 ID와 프래그먼트매니저의 findFragmentById 함수**

[예제 22-28] ③에서 메뉴 리스트 프래그먼트를 찾기 위해 사용된 다음의 함수가 바로 ID 방식의 이름을 사용한 예이다.

●   findFragmentById( R.id.*menu\_fragment* );

프래그먼트매니저 findFragmentById 함수는 프래그먼트 ID로 추가된 프래그먼트 객체를 찾아 반환해준다. 그렇다면 프래그먼트의 ID는 어떻게 설정할까? 다음의 그림을 살펴보자.

그림 22-26 프래그먼트 ID 설정



①     프래그먼트 ID는 레이아웃 XML에 정의하는 <fragment> 요소의 id 속성으로 설정할 수 있다. 이 방법은 뷰의 id를 정의하는 방법과 같다. 레이아웃 XML에서 정의된 특정 뷰를 참조하기 위해서도 id를 사용하지 않는가! 이 방식의 ID 설정은 [예제 22-27] ②에서 정적 프래그먼트를 정의할 때 사용한 바가 있다.

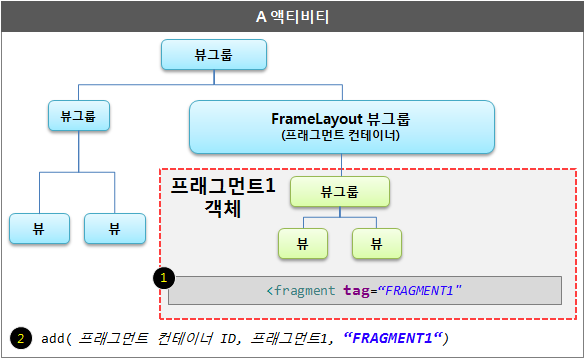
②     그런데 레이아웃 XML에 정의하지 않는 동적 프래그먼트는 어떻게 ID를 설정할까? 사실 동적 프래그먼트는 ID를 설정할 방법은 없다. 그래서 프래그먼트 ID가 아닌 다른 것의 ID를 활용하는데, 바로 프래그먼트 컨테이너의 ID다. 즉 프래그먼트를 포함하는 뷰그룹의 ID를 이용하는 것이다. 안드로이드는 동적으로 프래그먼트를 추가하면, 자동으로 프래그먼트를 포함하는 컨테이너의 ID를 프래그먼트의 ID로 설정한다. 하지만 이 방식은 완벽하지 않다. 그 이유는 프래그먼트 컨테이너에 여러 프래그먼트를 추가할 경우 모두 같은 ID를 가지게 된다. 따라서 특정 프래그먼트를 찾을 방법이 없다. 그래서 안드로이드는 차선책으로 이 경우 컨테이너 ID로 프래그먼트를 찾으면 제일 마지막에 추가된 프래그먼트를 반환한다. 그리고 마지막 추가한 프래그먼트가 dettach 되었거나 remove 되었다면 그 전에 추가된 프래그먼트를 반환한다. 좀 부족한 방법이긴 하지만 대부분의 경우 해결책은 될 수 있다.

[ REF \_Ref398581348 \r \h 그림 22-26] ②의 프래그먼트 컨테이너 ID를 이용한 방식은 완벽하지 못했다. 따라서 안드로이드는 또 다른 방식으로 프래그먼트를 찾을 수 있는 TAG를 제공한다.

**프래그먼트 TAG와 프래그먼트매니저의 findFragmentByTag 함수**

프래그먼트 이름으로 ID가 아닌 사용자가 정의하는 문자열의 이름을 줄 수 있는데, 그 것이 바로 태그TAG다. 태그는 특정 객체 등을 구분하기 위한 사용자 정의 이름표 정도로 생각하면 된다. 태그 설정 방법을 다음의 그림으로 살펴보자.

그림 22-27 프래그먼트 TAG 설정



①     정적 프래그먼트도 태그를 설정할 수 있다. 레이아웃 XML< fragment> 요소에 tag 속성을 사용하는데 tag 속성값으로는 원하는 문자열을 추가하면 된다.

②     동적 프래그먼트는 프래그먼트매니저에 프래그먼트를 추가하는 add 함수를 이용하여 태그 설정이 가능하다. add 함수의 세 번째 인자로 태그명을 설정하면 된다.

태그를 설정한 프래그먼트는 프래그먼트매니저의 findFragmentByTag 함수를 통해 원하는 해당 프래그먼트 객체를 반환받을 수 있다. 그렇다면 태그를 직접 사용해보자. [예제 22-29]을 수정한다.

예제 22-30 프래그먼트 태그를 활용하도록 수정 적용

|  |
| --- |
| **src/MainActivity.java**    **public** **class** MainActivity **extends** Activity {        @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);            setContentView(R.layout.*activity\_main*);            // ① 프래그먼트를 생성하고 추가할 때, addd 함수의 세 번째 인자로 프래그먼트  //    태그를 설정한다.          // ====================================================================  TextViewerFragment textViewerFragment = TextViewerFragment.*newInstance*();          ImageViewerFragment imageViewerFragment =  ImageViewerFragment.*newInstance*();            getFragmentManager()              .beginTransaction()  **.add(R.id.*viewer\_fragment\_container*,**  **textViewerFragment, "TEXT\_VIEWER")**  **.add(R.id.*viewer\_fragment\_container*,**  **imageViewerFragment, "IMAGE\_VIEWER")**                  .hide( imageViewerFragment )              .commit();          // ====================================================================              ListMenuFragment listMenuFragment = (ListMenuFragment)                     getFragmentManager().findFragmentById( R.id.*menu\_fragment* );            listMenuFragment.setOnListItemClickListener(  **new** ListMenuFragment.OnListItemClickListener()          {              @Override  **public** **void** onItemClick( **int** itemType )              {                  // ② 특정 프래그먼트를 숨기거나 보이기 위해 각 프래그먼트 객체를                  //    찾고, 프래그먼트 객체를 반환받는다.                  // ============================================================  **TextViewerFragment textViewerFragment =**  **(TextViewerFragment) getFragmentManager().**  **findFragmentByTag( "TEXT\_VIEWER" );**    **ImageViewerFragment imageViewerFragment =**  **(ImageViewerFragment) getFragmentManager().**  **findFragmentByTag( "IMAGE\_VIEWER" );**                  // ============================================================    **if**( itemType == ListMenuFragment.*ITEM\_TYPE\_TEXT\_VIEWER*)                  {                      getFragmentManager()                          .beginTransaction()                          .hide( **imageViewerFragment** )                          .show( **textViewerFragment** )                      .commit();                  }  **else** **if**( itemType == ListMenuFragment.*ITEM\_TYPE\_IMAGE\_VIEWER* )                  {                      getFragmentManager()                      .beginTransaction()                          .hide( **textViewerFragment** )                          .show( **imageViewerFragment** )                      .commit();                  }              }          });      }  } |

①     텍스트뷰어, 이미지뷰어 프래그먼트를 생성하고 add 함수로 추가할 때, 해당 함수의 세 번째 인자로 태그명을 설정한다. 그런데 이전과는 달리 생성된 프래그먼트 객체를 멤버변수로 저장해두지 않았다. 그 이유는 그럴 필요가 없기 때문이다. 태그명을 설정했기 때문에 언제라도 태그명을 이용해서 프래그먼트매니저를 통해 참조할 수 있기 때문이다.

②     프래그먼트매니저의 findFragmentByTag 함수로 원하는 프래그먼트 객체를 참조한다. 해당 함수의 인자는 프래그먼트 태그를 전달하면 된다. 참조한 프래그먼트 객체를 통해 프래그먼트를 숨기거나 보이는 처리를 하면된다.

프래그먼트 ID와 태그는 비록 간단한 내용이나, 두 가지 방법이 왜 필요한지 이해해두지 않으면 어떤 것을 사용해야하는지 판단하기 어렵다. 따라서 정확히 이해하고 넘어가자.

<잠깐만>

|  |  |
| --- | --- |
| **프래그먼트 트랙잭션을 commit하면 바로 적용될까?**    만일 프래그먼트 트랙잭션 객체를 이용하여 특정 프래그먼트를 add 후 commit 했다면 즉시 프래그먼트매니저에 반영될까? 다음의 예제 소스를 살펴보자.   |  | | --- | | ...  // ① 프래그먼트를 프래그먼트매니저에 추가한다.  getFragmentManager()  .beginTransaction()  .add(R.id.*viewer\_fragment\_container*,  textViewerFragment, "TEXT\_VIEWER")  .commit();    // ② 프래그먼트매니저에 추가한 프래그먼트를 찾아 참조한다.  **TextViewerFragment textViewerFragment =**  **(TextViewerFragment) getFragmentManager().**  **findFragmentByTag( "TEXT\_VIEWER" );** |   ①     프래그먼트매니저에 새로운 프래그먼트를 추가했다.  ②     바로 다음 줄에서 추가한 프래그먼트를 찾아 참조한다. 여기서 추가한 프래그먼트를 찾을 수 있을까?  예제 ①에서 프래그먼트를 추가하더라도 프래그먼트매니저는 ②가 실행되는 시점에 추가된 프래그먼트를 가지지 않는다. 따라서 ②의 findFragmentByTag 함수의 반환값은 널이다.  그 이유는 프래그먼트매니저가 처리해야할 트랜잭션을 메인스레드의 메시지큐에 추가하기 때문이다. 큐에 유입된 메시지는 메인스레드의 루퍼가 유입된 순서대로 처리한다. 따라서 메인스레드의 루퍼가 프래그먼트트랜잭션을 처리해줄 때까지 기다려야한다.    *그렇다면 그 시점은 언제일까?*    바로 프래그먼트트랜잭션을 commit한 생명주기 함수 다음 단계다. 만일 액티비티 onCreate에서 프래그먼트 트랜잭션을 commit 했다면, 그 이후 호출되는 생명주기 함수인 0nStart부터란 의미다. 참고로 앞서 설명한 메인스레드, 메시지큐, 루퍼 등이 뭔지 잘 모르겠다면 14장 [14.2]절은 꼭 참고하자. 안드로이드 앱를 개발하면서 메인스레드, 메시지큐, 루퍼 등을 모르면 늘 한계에 직면하게 된다. |