

[22장] 프래그먼트 (1/8) - (20140815 완료) | 책에 담지 못한 장들

슈퍼성근 | 조회 326 | 2014/08/10 22:38:46

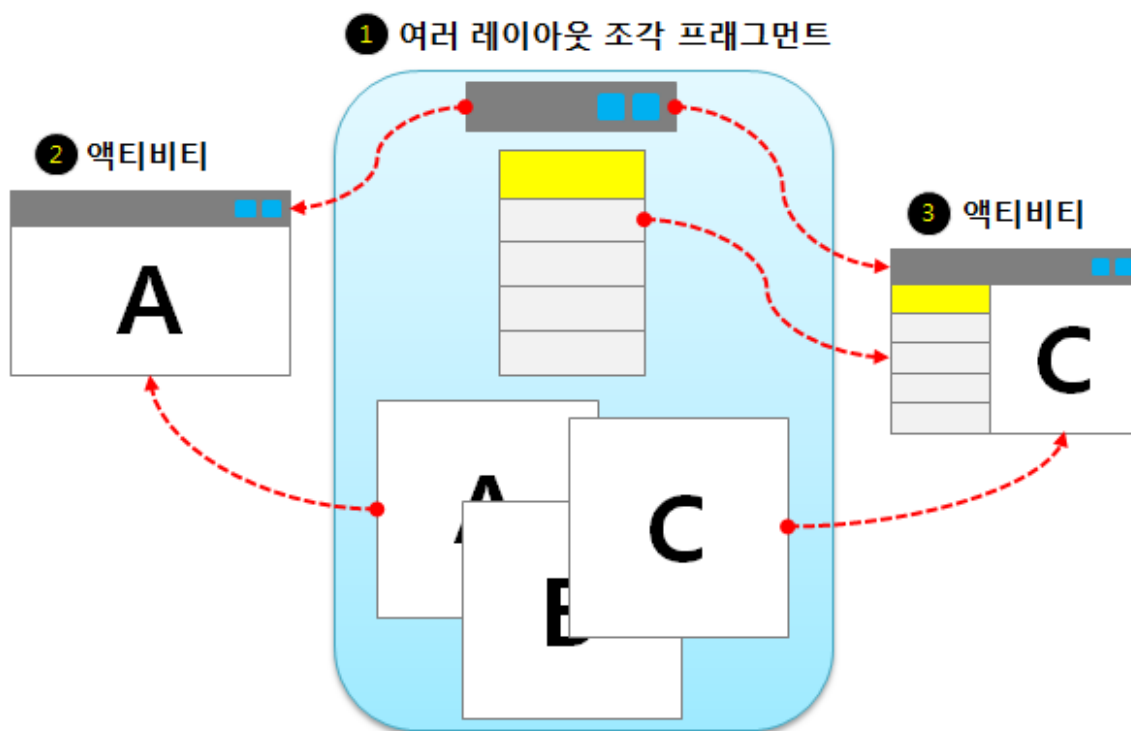
주의 : 소스 내용 중 "**OnClick**", "**OnStart**" 함수명 첫글자가 숫자 0인것은 오타가 아닙니다.

다음 게시판은 onClicK 라는 글을 입력할 수 없기 때문에 어쩔 수 없이  
영문 소문자 o를 숫자 0으로 대체하였습니다. ^^;

## 22장 프래그먼트

프래그먼트를 간단히 한 줄로 설명하자면, 액티비티 내의 작은 액티비티라 할 수 있다. 그 말은 프래그먼트는 UI와 라이프사이클을 가지고 있는 독립적인 모듈이라는 의미다. 그러나 독립적으로 실행될 수 없고 액티비티에 포함되어 사용된다는 점은 액티비티와 다른 점이다. [그림 22-1]을 살펴보자.

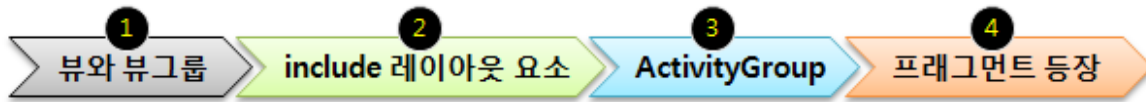
그림 22-1 프래그먼트



- ① 프래그먼트는 액티비티와 같이 자신만의 레이아웃을 포함하고, 별도의 생명주기 함수도 가진다. 이 단위는 매우 유연하고 독립적이어서 다른 액티비티에 포함될 수도 있다.
- ② , ③과 같이 특정 액티비티는 여러 프래그먼트를 조합하여 화면을 구성할 수 있다.

[그림 22-1]과 같이 프래그먼트 단위의 조합으로 만들어진 액티비티는 복잡도가 낮고 유지보수가 용이하다. 즉 프래그먼트 단위로 레이아웃과 자바 소스 파일이 분리되기 때문이다. 여기까지 보자면 프래그먼트는 안드로이드의 필수적인 요소라고 느껴질 수 있다. 하지만 프래그먼트는 안드로이드 API 11 허니콤부터 추가되었고, 그전까지 프래그먼트가 없어도 앱을 구현하는데 문제는 없었다. 아니 좀더 정확히 표현하자면 프래그먼트를 대신할 수 있는 다른 요소들을 이용했다고 하는 것이 맞다. 그래서 프래그먼트를 정확히 이해하려면 [그림 22-2]과 같이 프래그먼트 이전의 기술들을 차례대로 살펴보는 것이 중요하다.

그림 22-2 뷰와 뷰그룹에서 프래그먼트까지



- ① 레이아웃은 뷰와 뷰그룹 요소만으로도 원하는 화면과 그에 대한 처리를 모두 할 수 있다.
- ② 하지만 include 레이아웃 요소를 이용하면 좀더 화면을 유연하게 구성할 수 있도록 돕는다.
- ③ 하지만 ActivityGroup를 이용하면 여러 개의 액티비티를 포함할 수 있기 때문에 훨씬 강력한 방법으로 화면을 유연하고 독립적으로 구성할 수 있다.
- ④ 하지만 프래그먼트를 이용하면 ActivityGroup보다 완벽한 방법으로 화면을 유연하고 독립적으로 구성할 수 있다. 참고로 프래그먼트는 ActivityGroup의 문제점들을 해결하기 위해 등장했다. 즉 프래그먼트는 ActivityGroup를 대체하기 위해 탄생한 것이다.

[그림 22-2]에 나열된 기술의 흐름은 너무 자연스러워 그 필요성을 느끼는데 전혀 어렵지 않다. 가벼운 마음으로 시작해보자.

[이 장의 실습 내용]

- ① 뷰와 뷰그룹만을 이용하기에는 불편한 레이아웃 XML
- ② 레이아웃 XML 요소 include 활용
- ③ ActivityGroup 활용
- ④ 프래그먼트 활용

## 22.1 레이아웃 XML 요소 include

레이아웃 XML의 include 요소는 특정 레이아웃을 다른 레이아웃 안으로 포함시킬 수 있도록 한다. 따라서 여러 레이아웃에 중복적으로 들어가는 부분의 레이아웃을 따로 빼서 별도의 레이아웃 리소스 파일을 만들고, 필요한 레이아웃에 include 요소를 이용하여 추가하면 된다. 먼저 include 요소의 필요성을 예제를 통해 느껴보자.

### 22.1.1 뷰와 뷰그룹을 이용한 일반적인 레이아웃

일반적인 [그림 22-3]과 같은 레이아웃을 구성해보는 과정에서 불편한 요소를 찾아보자.

그림 22-3 일반적인 레이아웃 구조



- ① A 액티비티는 상단에 부제목 텍스트와 메뉴, 종료 버튼으로 구성된 부제목 영역이 존재하고 그 하단에는 액티비티의 콘텐츠 영역의 레이아웃이 배치된다. 그리고 가장 하단에는 B 액티비티를 실행할 수 있는 버튼이 존재한다.

- ② B 액티비티 역시 상단에 A 액티비티와 같은 부제목 영역이 존재하고 하단에는 액티비티의 콘텐츠 영역의 레이아웃이 배치된다.

새로운 예제 패키지를 생성하고 다음과 같이 AndroidManifest.xml을 작성한다.

#### 예제 22-1 일반적인 레이아웃 구조 앱의 AndroidManifest.xml

##### AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.superdroid.fragmentinclude"
    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=".AActivity" android:label="A Activity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".BActivity" android:label="B Activity" />

    </application>

</manifest>
```

AndroidManifest.xml에 A, B 액티비티 컴포넌트를 등록했다. 다음은 각 액티비티에서 사용될 레이아웃을 구현한다.

#### 예제 22-2 A 액티비티의 레이아웃 리소스

##### res/layout/a\_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="vertical">

    <!-- 부 타이틀 영역의 레이아웃 -->
    <LinearLayout android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="5dp"
        android:background="#CCC"
        android:orientation="horizontal">
```

```

<TextView android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="부 타이틀 영역 입니다." />

<Button android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="메뉴" />

<Button android:id="@+id/btn_finish"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="종료"
    android:OnClick="OnClick" />

</LinearLayout>

<!-- A 액티비티 Content 영역의 레이아웃 -->
<TextView
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:gravity="center"
    android:textSize="25dp"
    android:text="A Activity Content Layout" />

<Button android:id="@+id/btn_run_b_activity"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Run B Activity"
    android:OnClick="OnClick" />

</LinearLayout>

```

### 예제 22-3 B 액티비티의 레이아웃 리소스

#### res/layout/b\_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="vertical">

    <!-- 부 타이틀 영역의 레이아웃 -->
    <LinearLayout android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="5dp"
        android:background="#CCC"
        android:orientation="horizontal">

        <TextView android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="부 타이틀 영역 입니다." />
    
```

```

<Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="메뉴"/>

<Button android:id="@+id/btn_finish"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="종료"
        android:OnClick="OnClick"/>

</LinearLayout>

<!-- B 액티비티 Content 영역의 레이아웃 -->
<TextView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:textSize="25dp"
    android:text="B Activity Content Layout"/>

</LinearLayout>

```

다음은 A, B 액티비티 소스를 구현한다.

#### 예제 22-4 A 액티비티

##### src/AActivity.java

```

public class AActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.a_activity_main);
    }

    public void OnClick( View v )
    {
        switch( v.getId() )
        {
            case R.id.btn_finish:
            {
                finish();
                break;
            }

            case R.id.btn_run_b_activity:
            {
                Intent intent = new Intent(this, BActivity.class);
                startActivity(intent);
                break;
            }
        }
    }
}

```

## 예제 22-5 B 액티비티

## src/BActivity.java

```

public class BActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.b_activity_main);
    }

    public void onClick( View v )
    {
        switch( v.getId() )
        {
            case R.id.btn_finish:
            {
                finish();
                break;
            }
        }
    }
}

```

예제를 실행하여 결과를 확인해보자.

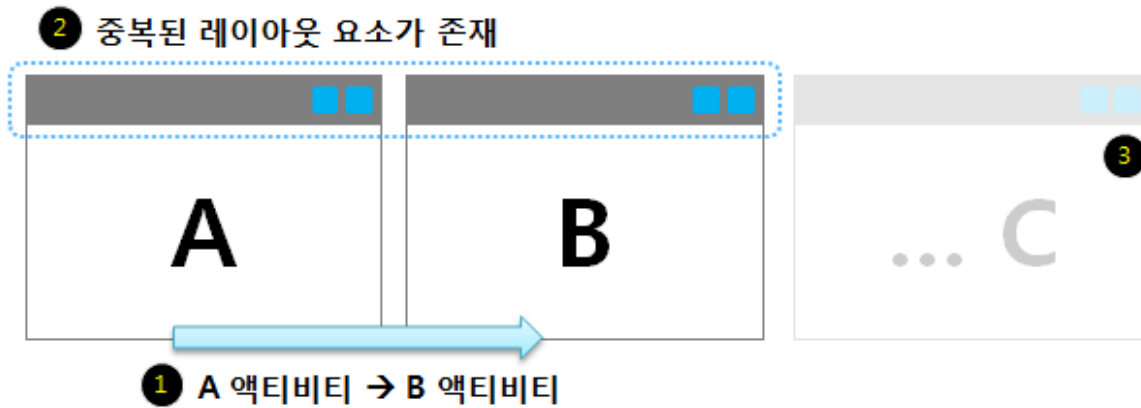
## 그림 22-4 일반적인 레이아웃 예제 앱 실행 결과



- ① A 액티비티가 실행되었다. 레이아웃을 살펴보면 상단에 부제목 영역과 하단에 A 액티비티의 콘텐츠 영역이 존재한다.
- ② Run B Activity 버튼을 눌러 B 액티비티를 실행한다.
- ③ B 액티비티가 실행되었다. 레이아웃을 살펴보면 상단에 부제목 영역과 하단에 B 액티비티의 콘텐츠 영역이 존재한다.
- ④ 종료 버튼을 누른다. B 액티비티는 종료되고 A 액티비티로 복귀된다.

지금까지 예제 소스에서 불편한 점 한 가지를 발견했는가! [그림 22-4]를 통해 살펴보자.

## 그림 22-5 중복 레이아웃 영역



- ① A는 B 액티비티를 실행하고 있다.
- ② 여기서 A, B 액티비티의 레이아웃을 살펴보면 상단 부제목 영역이 공통적으로 존재한다.
- ③ 만일 C, D, E... 등으로 더 많은 액티비티를 실행하게 된다면 중복된 부제목 영역이 계속 추가되어야 하고, 액티비티의 레이아웃 리소스에 [예제 22-2], [예제 22-3]과 같이 반복적으로 부제목 영역의 레이아웃 코드를 추가해줘야 할 것이다. 또한 부제목 영역의 레이아웃이 변경되기라도 한다면 모든 레이아웃 리소스들을 수정해줘야 하기 때문에 불편하다.

이렇게 중복된 레이아웃은 소스의 양을 증가시키고 유지보수를 어렵게 한다. 이 문제를 해결하기 위해 레이아웃에는 include 요소를 제공하고 있다. 그렇다면 include 요소를 직접 사용해보자.

## 22.1.2 include 요소 활용

먼저 중복되어 사용되는 레이아웃 영역을 별도의 레이아웃 XML 파일로 분리한다.

### 예제 22-6 중복 레이아웃 영역 분리

#### res/layout/subtitle\_layout.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="5dp"
    android:background="#CCC"
    android:orientation="horizontal">

    <TextView android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="부 타이틀 영역 입니다." />

    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="메뉴" />

    <Button android:id="@+id/btn_finish"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="종료"
        android:onClick="OnClick" />

</LinearLayout>
```

이제 분리된 레이아웃을 A, B 액티비티 레이아웃 리소스에 적용해보자.

#### 예제 22-7 A 액티비티 레이아웃 리소스

##### res/layout/a\_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="vertical">

    <!-- 부 타이틀 영역의 레이아웃 -->
    <include layout="@layout/subtitle_layout"/>

    <!-- A 액티비티 Content 영역의 레이아웃 -->
    <TextView android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="center"
        android:textSize="25dp"
        android:text="A Activity Content Layout" />

    <Button android:id="@+id/btn_run_b_activity"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Run B Activity"
        android:OnClick="OnClick"/>

</LinearLayout>
```

#### 예제 22-8 B 액티비티 레이아웃 리소스

##### res/layout/b\_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="vertical">

    <!-- 부 타이틀 영역의 레이아웃 -->
    <include layout="@layout/subtitle_layout"/>

    <!-- B 액티비티 Content 영역의 레이아웃 -->
    <TextView android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:textSize="25dp"
        android:text="B Activity Content Layout" />

</LinearLayout>
```

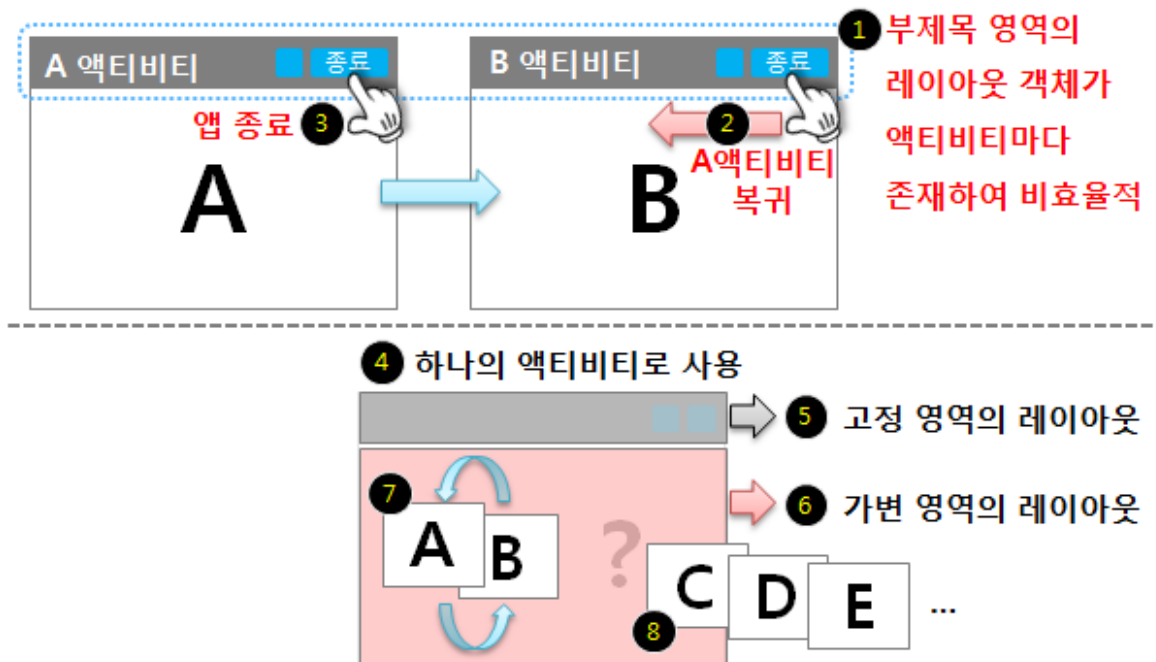
간단히 부제목 영역의 레이아웃을 제거하고, 그 부분에 include 요소를 추가한 다음 layout 속성값으로 분리된 레이아웃 리소스 ID를 적어주면 끝이다.



### 22.1.3 include 요소의 한계

공통적으로 사용될 레이아웃을 분리하여 필요한 레이아웃에 끼워 넣는 include 요소의 장점은 딱 거기 까지도. 즉 레이아웃 리소스의 소스 양을 줄여 유지보수 하기 쉽도록 돕는다는 점 이외에는 특별한 것이 없다는 것이다. 물론 이것 만으로도 충분한 가치가 있고 꼭 알고 있어야 할 기술임에는 분명하다. 어쨌든 include 요소의 한계는 무엇이고, 어떤 것이 더 필요한지 살펴보자.

그림 22-6 include 요소의 한계




- ① include 요소를 이용하여 부제목 영역의 레이아웃 코드는 재사용할 수 있었지만, 레이아웃 객체는 액티비티마다 각각 존재해서 비효율적이다. 뿐만 아니라 [예제 22-4], [예제 22-5]와 같이 액티비티마다 부제목 영역의 버튼 처리를 자바 소스로 각각 중복해서 구현해주어야 해서 유지보수가 어렵다. 물론 A, B 액티비티가 특정 액티비티를 상속받도록 구현하고, 그 상위 액티비티에서 부제목 영역의 처리를 담당한다면 이 문제는 해결될 수 있다. 하지만 다음의 상황에서 또 다른 문제점이 발생된다.
- ② B 액티비티에서 종료 버튼을 누르면 finish 함수를 호출하고 A 액티비티로 복귀한다. 하지만 분명 사용자는 앱 자체가 종료된다고 생각할 것이다.
- ③ 그리고 A 액티비티에서 종료 버튼을 누르면 A 액티비티가 종료되고 태스크 하위에 더 이상 액티비티가 존재하지 않기 때문에 홈으로 복귀한다. 즉 앱이 종료되었다고 볼 수 있다. 결국 A, B 액티비티의 부제목 영역의 처리는 동일한 결과를 처리할 수 없다. 이는 A, B 액티비티가 분리되었기 때문이다.
- ④ 이 문제를 해결하려면 A, B 액티비티를 하나의 액티비티로 합치면 된다.
- ⑤ 합쳐진 액티비티 레이아웃에는 상단에 고정 영역의 레이아웃이 존재하고
- ⑥ 하단에는 콘텐츠 내용이 변하는 가변 영역의 레이아웃이 존재하게 된다.
- ⑦ 가변 영역은 FrameLayout 영역으로 설정하고 A, B의 레이아웃을 서로 감쌌다 보여줬다 하면서 이동하면 된다.
- ⑧ 여러 액티비티에 들어갈 레이아웃을 하나의 액티비티에 포함하고 서로 전환하면서 보여주는 것은 매우 효율적이다. 하지만 만일 두 가지 레이아웃이 아니라 C, D, E... 등 전환할 레이아웃 개수가 늘어난다면 하나

의 액티비티에서 처리하기가 매우 복잡할 뿐만 아니라 관리하기도 쉽지 않을 것이다.

[그림 22-6]의 ⑧과 같은 문제를 해결하기 위해 바로 액티비티그룹 `ActivityGroup`이라는 것이 존재한다.

다음글에서 계속 됩니다.

[참고 예제 소스]

 22-1. Include 요소 적용전 일반적인 레이아웃.zip

 22-2. Include 요소 적용.zip

---

[인쇄하기](#)[취소](#)