|  |
| --- |
| **주의 : 소스 내용 중 "0nClick", "0nStart" 함수명 첫글자가 숫자 0인것은 오타가 아닙니다.**             다움 게시판은 o n C l i c k 라는 글을 입력할 수 없기 때문에 어쩔 수 없이             영문 소문자 o를 숫자 0으로 대체하였습니다. ^^; |

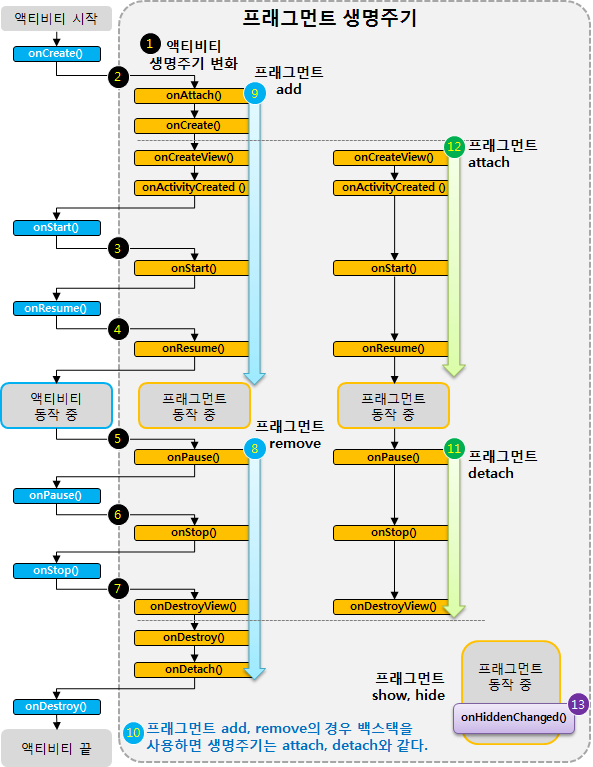
**22.3.6 프래그먼트의 생명주기**

프래그먼트도 액티비티와 같이 상태의 변화에 따른 생명주기 함수를 가진다. 그리고 그 생명주기는 액티비티의 생명주기 함수와 매우 유사하다. 그 이유는 액티비티와 프래그먼트 관계에 있는데, 프래그먼트가 액티비티 내에 포함되기 때문이다. 따라서 액티비티가 시작, 중단, 종료되면 프래그먼트 역시 시작, 중단, 종료될 수밖에 없다.

프래그먼트 생명주기를 설명하기 앞서 미리 말해둘 것이 있다. 프래그먼트 생명주기 함수는 액티비티 생명주기 함수보다 더 다양하다. 그래서 그 개수만 보면 복잡해 보이고 어렵게 느껴질 것이다. 하지만 절대 외우려 들지 말자. 각 생명주기 함수를 모두 활용하는 경우는 거의 없기 때문이다. 그리고 생명주기 함수는 앱의 다양한 상태 변화를 대처하기 위해 필요한 것이다. 따라서 개발자가 프래그먼트의 각 상태 변화에 대처하고 싶은 처리가 있다면, 정확히 그 시점에 호출되는 생명주기 함수만 그때그때 찾아 적용하면 된다.

[그림 22-36] 한 장으로 프래그먼트 생명주기를 이해해볼 것이다. 프래그먼트는 액티비티 생명주기에 영향을 받는다는 점을 기억하고 시작해보자.

그림 22-36 액티비티 생명주기와 프래그먼트 생명주기 함수



**①**     **프래그먼트 생명주기는 액티비티의 생명주기에 영향을 받는다. onCreate 액티비티 생명주기에서 프래그먼트를 생성 및 추가했을 때 프래그먼트 생명주기를 살펴보자.**

②     프래그먼트 생명주기 함수는 onAttach, onCreate, onCreateView, onActivityCreated가 순서대로 호출된다.

③     이어 액티비티는 0nStart 생명주기 함수가 호출된 후 프래그먼트 생명주기 함수 0nStart가 호출된다.

④     이어 액티비티는 onResume 생명주기 함수가 호출된 후 프래그먼트 생명주기 함수 0nResume이 호출된다. 이로써 액티비티와 프래그먼트는 화면에 표시되고, 사용자와 상호작용을 할 수 있다.

⑤     만일 액티비티를 종료하면, 이제 반대로 프래그먼트 생명주기 함수 0nPause가 먼저 호출된 후 액티비티는 생명주기 함수 onPause가 호출된다. 그 이유는 액티비티가 종료 과정을 진행하기 전에 프래그먼트가 미리 종료 준비를 할 수 있도록 하기 위함이다.

⑥     이어 프래그먼트 생명주기 함수 0nStop이 호출된 후 액티비티 생명주기 함수 0nStop가 호출된다.

⑦     이어 프래그먼트 생명주기 함수 onDestroyView, onDestroy, onDetach가 순서대로 호출된 후 액티비티 생명주기onDestroy가 호출된다.

⑧     **다음은 액티비티와 프래그먼트가 동작 중에 프래그먼트를 제거(remove), 추가(add)할 때 프래그먼트 생명주기 변화를 살펴보자.**먼저 프래그먼트를 제거(remove)한다. 이때 프래그먼트 생명주기 함수는 0nPause, 0nStop, onDestroyView, onDestroy, onDetach가 순서대로 호출된다. 즉 액티비티 생명주기의 변화없이 프래그먼트 생명주기만 변한다는 것을 기억하자.

⑨     이어 다시 프래그먼트를 추가(add)한다. 이때 프래그먼트 생명주기 함수는 onAttach, onCreate, onCreateView, onActivityCreated, 0nStart, 0nResume이 순서대로 호출된다. 이 경우도 액티비티 생명주기의 변화없이 프래그먼트 생명주기만 변한다.

⑩     그런데 주의할 점이 있다. 만일 프래그먼트를 add나 remove할 때 백스택에 추가했다면, 생명주기는 attach, detach와 동일해진다. 그렇다면 프래그먼트 attach, detach 생명주기를 살펴보자.

⑪     **다음은 액티비티와 프래그먼트가 동작 중에 프래그먼트를 떼어내고(detach) 붙일(attach) 때 프래그먼트 생명주기 변화를 살펴보자.**먼저 프래그먼트를 떼어(detach)낸다. 이때 프래그먼트 생명주기 함수는 0nPause, 0nStop, onDestroyView가 순서대로 호출된다. 여기서 프래그먼트를 remove 했을때 호출되었던 onDestroy, onDetach가 불리지 않는 이유는 뭘까? 바로 detach의 경우 프래그먼트 자체는 제거되지 않기 때문이다. 이는 [그림 22-24]에서 설명되었다. onDestroy, onDetach 함수가 호출되는 것은 프래그먼트 객체 자체가 제거됨을 의미한다. 그런데 ⑩에서 설명한 바와 같이 remove 시 백스택에 추가한 경우도 onDestroy, onDetach가 호출되지 않는다. 그 이유는 비록 프래그먼트를 remove 했더라도 이전키 등으로 복귀될 수 있기 때문에 프래그먼트 자체를 제거하지 않기 때문이다. 헷갈리는 부분이니 명확히 기억해두자.

⑫     이어 다시 프래그먼트를 붙인다(attach). 이때 프래그먼트 생명주기 함수는 onCreateView, onActivityCreated, 0nStart, 0nResume이 순서대로 호출된다. onAttach, onCreate가 호출되지 않는 이유는 프래그먼트 객체 자체가 제거되지 않았기 때문이다. onAttach, onCreate가 호출되는 것은 해당 프래그먼트가 새로 생성됨을 의미한다. 그런데 ⑩에서 설명한 바와 같이 add 시 백스택에 추가한 경우도 onAttach, onCreate가 호출되지 않는다. 그 이유는 백스택에 프래그먼트 객체가 존재하기 때문이다.

⑬     **다음은 액티비티와 프래그먼트가 동작 중에 프래그먼트를 숨기고 보이게 할 때 프래그먼트 생명주기 변화를 살펴보자.** 이 경우 프래그먼트 생명주기 변화는 없다. 다만 onHiddenChanged 함수가 호출된다. 따라서onHiddenChanged 함수를 통해 프래그먼트 레이아웃이 숨겨지거나 보일 때 원하는 처리를 할 수 있다. onHiddenChanged 함수 인자로 전달되는 불린 값이 true라면 숨겨진 상태고 false면 보이는 상태다.

[그림 22-36]은 필요할때마다 참조할 수 있도록 출력해서 책상 주변에 붙여두면 편하다. 해당 그림은 슈퍼드로이드 카페의 ‘책의 부록’ 게시판에서 내려받을 수 있다.

각각의 프래그먼트 생명주기 함수는 [표 22-1]을 통해 상세히 설명한다.

표 22-1 프래그먼트 생명주기 함수

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 생명주기 함수 | | | | | 설명 |
| onAttach | | | | | 프래그먼트가 액티비티 레이아웃에 추가될 때 호출된다. 그리고 이 함수가 호출되었다는 것은 프래그먼트매니저가 해당 프래그먼트의 관리를 시작한다는 의미다. |
|  | onCreate | | | | 프래그먼트 객체가 최초 생성될 때 호출된다. 따라서 이 함수에서 프래그먼트 내부에 필요한 각종 클래스들을 생성 및 초기화하면 된다. 그리고 이 함수가 호출되었다는 것은 프래그먼트 객체가 새로 생성됨을 의미한다. 반대로 호출되지 않았다면 기존에 프래그먼트 객체가 존재하여 재사용된 것이다. 일반적으로 프래그먼트를 add하면 호출 되어야 하지만 기존 백스택에 해당 프래그먼트가 존재하면 호출되지 않는다. |
|  | | onCreateView | | | 액티비티에서 프래그먼트의 레이아웃을 요청할 때 호출된다. 따라서 해당 함수에서 프래그먼트 레이아웃을 생성하여 반환해야 한다.  그리고 이 함수가 호출되었다는 것은 프래그먼트의 레이아웃이 새로 생성됨을 의미한다. |
|  | | onActivityCreated | | | 프래그먼트를 포함하고 있는 액티비티 onCreate 생명주기 함수가 호출된 후 호출된다. 따라서 액티비티의 onCreate 과정 이후에 처리해야할 일이 있다면 이 함수에서 처리하면 된다. |
|  | | | 0nStart | | 프래그먼트 레이아웃이 화면에 표시될 때 호출된다. 단 아직 사용자와 상호작용은 할 수 없는 상태다. |
|  | | | | onResume | 사용자와 상호작용도 할 수 있는 상태다. |
|  | | | | onPause | 사용자와 상호작용을 할 수 없는 상태다. |
|  | | | onStop | | 프래그먼트 레이아웃이 화면에서 사라질 때 호출된다. |
|  | | onDestroyView | | | 액티비티에서 프래그먼트의 레이아웃을 제거할 때 호출된다. 따라서 해당 함수가 호출되면 화면에는 프래그먼트 레이아웃이 존재하지 않는다. |
|  | onDestory | | | | 프래그먼트 객체가 소멸될 때 호출되는 함수다. |
| onDetach | | | | | 프래그먼트가 액티비티 레이아웃에서 제거될 때 호출된다. 그리고 이 함수가 호출되었다는 것은 프래그먼트매니저가 더 이상 해당 프래그먼트를 관리하지 않는다는 의미다. |
|  |  |  |  |  |  |

재차 말하지만 프래그먼트 생명주기 함수는 그 개수가 많고 복잡해보인다. 그러므로 [그림 22-36]을 정확히 이해만 하고 넘어가자. 개발하면서 프래그먼트를 다룰 때, 특정 프래그먼트 상태에서 처리해야 할 일이 생긴다면 정확히 그 시점에 필요한 생명주기 함수만 찾을 수 있으면 그만이다. 그리고 프래그먼트를 사용하다보면 자연스럽게 외워진다.

다음은 프래그먼트 상태 변화에 따라 생명주기 함수가 호출되는 것을 예제로 직접 확인해보자.

**액티비티의 실행과 종료에 따라 프래그먼트의 상태변화와 액티비티의 실행 중 프래그먼트 show, hide 상태변화**

액티비티를 실행과 종료에 따라 프래그먼트 생명주기 상태변화를 확인하기 위해, 액티비티와 텍스트뷰어 프래그먼트 각각 모든 생명주기 함수를 추가한다. 그리고 각 생명주기 함수가 호출되는 것을 직접 확인하기 위해 로그를 추가한다.

먼저 액티비티의 생명주기 함수를 추가한다.

예제 22-34 액티비티 생명주기 함수 추가

|  |
| --- |
| **src/MainActivity.java**    **public** **class** MainActivity **extends** Activity **implements** OnBackStackChangedListener {      ...      @Override  **protected void onCreate(Bundle savedInstanceState)** {  **super**.onCreate(savedInstanceState);  Log.*d*("superdroid", "Activity onCreate()");          ... }        @Override  **protected void 0nStart() {**          Log.*d*("superdroid", "Activity 0nStart()");  **super**.0nStart(); }        @Override  **protected void onRestart() {**          Log.*d*("superdroid", "Activity onRestart()");  **super**.onRestart(); }        @Override  **protected void 0nResume(){**          Log.*d*("superdroid", "Activity 0nResume()");  **super**.onResume(); }        @Override  **protected void 0nPause() {**          Log.*d*("superdroid", "Activity 0nPause()");  **super**.onPause(); }        @Override  **protected void 0nStop() {**          Log.*d*("superdroid", "Activity 0nStop()");  **super**.onStop(); }        @Override  **protected void onDestroy() {**          Log.*d*("superdroid", "Activity onDestroy()");          ...  **super**.onDestroy(); }  } |

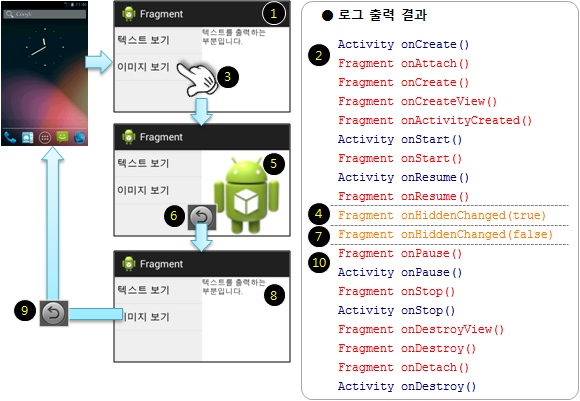
다음은 텍스트뷰어 프래그먼트에 생명주기 함수를 추가한다. 참고로 이미지뷰어 프래그먼트까지 생명주기 함수를 추가하면 출력되는 로그가 복잡해보인다. 따라서 이미지뷰어 프래그먼트 소스는 수정하지 않는다.

예제 22-35 텍스트뷰어 프래그먼트 생명주기 함수 추가

|  |
| --- |
| **src/TextViewerFragment.java**    **public** **class** TextViewerFragment **extends** Fragment {        ...      @Override  **public void onAttach( Activity activity ) {**          Log.*e*("superdroid", "Fragment onAttach()");  **super**.onAttach( activity ); }        @Override  **public void onCreate( Bundle savedInstanceState ) {**          Log.*e*("superdroid", "Fragment onCreate()");  **super**.onCreate( savedInstanceState ); }        @Override  **public View onCreateView( LayoutInflater inflater,**  **ViewGroup container,**  **Bundle savedInstanceState ) {**          Log.*e*("superdroid", "Fragment onCreateView()");  **return** inflater.inflate( R.layout.*fragment\_text\_viewer*,                                      container,  **false**); }        @Override  **public void onActivityCreated( Bundle savedInstanceState ) {**          Log.*e*("superdroid", "Fragment onActivityCreated()");  **super**.onActivityCreated( savedInstanceState ); }        @Override  **public void 0nStart() {**          Log.*e*("superdroid", "Fragment 0nStart()");  **super**.0nStart(); }        @Override  **public void 0nResume() {**          Log.*e*("superdroid", "Fragment 0nResume()");  **super**.onResume(); }        @Override  **public void 0nPause() {**          Log.*e*("superdroid", "Fragment 0nPause()");  **super**.onPause(); }        @Override  **public void 0nStop() {**          Log.*e*("superdroid", "Fragment 0nStop()");  **super**.onStop(); }        @Override  **public void onDestroyView() {**          Log.*e*("superdroid", "Fragment onDestroyView()");  **super**.onDestroyView(); }        @Override  **public void onDestroy() {**          Log.*e*("superdroid", "Fragment onDestroy()");  **super**.onDestroy(); }        @Override  **public void onDetach() {**          Log.*e*("superdroid", "Fragment onDetach()");  **super**.onDetach(); }        @Override  **public void onHiddenChanged( boolean hidden ) {**          Log.*w*("superdroid", "Fragment onHiddenChanged(" + hidden + ")");  **super**.onHiddenChanged( hidden ); }  } |

액티비티와 프래그먼트 모두 생명주기가 추가되었다. 다음은 텍스트뷰어 프래그먼트를 숨기거나 보여줄 때 생명주기 함수가 호출되는 것을 확인해야 한다. 하지만 이미 [예제 22-29]에서 프래그먼트를 보이거나 감추도록 구현되어 있기 때문에 추가 구현할 내용이 없다. 바로 실행하여 결과를 확인해보자. 참고로 예제의 실행 결과는 반드시 [그림 22-36]과 비교하며 확인하도록 하자.

그림 22-37 액티비티 실행과 종료에 따른 프래그먼트 생명주기 변화와 프래그먼트 show, hide에 따른 생명주기 변화



①     예제 앱을 실행하면 화면 우측에 텍스트뷰어 프래그먼트가 화면에 나타난다.

②     액티비티 생명주기와 프래그먼트 생명주기를 로그로 살펴보면, [그림 22-36] ②부터 ④ 과정까지 생명주기 함수가 정상적으로 호출된 것을 확인할 수 있다.

③     다음은 이미지 보기 리스트 아이템을 선택한다.

④     이때 이미지뷰어 프래그먼트는 show하고, 텍스트뷰어 프래그먼트는 hide한다. 따라서 텍스트뷰어 프래그먼트 생명주기를 보면 onHiddenChanged가 호출되고, 그 인자로 true 값이 전달되었다. true의 의미는 텍스트뷰어 프래그먼트가 숨겨졌다는 것이다.

⑤     이제 화면 우측에는 이미지뷰어 프래그먼트가 보인다.

⑥     이 상태에서 이전키를 누른다.

⑦     이때 최근 추가된 백스택 트랜잭션이 반대로 실행될 것이다. 여기서 최근 실행된 트랜잭션은 이미지뷰어를 보이고, 텍스트뷰어는 감추는 것이다. 따라서 그 반대로 텍스트뷰어는 다시 보이고, 이미지뷰어는 사라진다. 로그를 확인해보면 텍스트뷰어 프래그먼트에서 onHiddenChanged가 false 인자로 호출되었다.

⑧     이제 화면 우측에는 다시 텍스트뷰어 프래그먼트가 보인다.

⑨     다시 이전키를 누른다. 그런데 백스택에는 더 이상 쌓인 트랜잭션이 없다. 따라서 이 경우 액티비티 자체가 종료된다.

⑩     로그를 확인해보면 [그림 22-36] ⑤부터 ⑦ 과정까지 생명주기 함수가 정상적으로 호출된 것을 확인할 수 있다.

[참고 예제 소스]

[[첨부파일](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D8%26fileid%3D5%26regdt%3D20140926214933&url=http%3A%2F%2Fcfile286.uf.daum.net%2Fattach%2F244F1A41542D15B5170A3B')) 22-10. 프래그먼트 라이프 사이클(액티비티 변화와, 프래그먼트 show hide 변화).zip](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D8%26fileid%3D5%26regdt%3D20140926214933&url=http%3A%2F%2Fcfile286.uf.daum.net%2Fattach%2F244F1A41542D15B5170A3B'))

**액티비티의 실행 중 프래그먼트 attach, detach 상태변화**

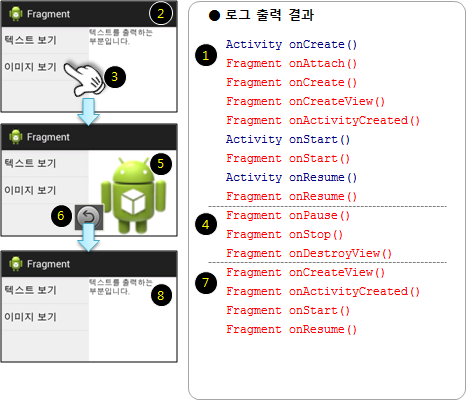
 다음은 액티비티 실행 중 프래그먼트를 detach, attach할 때 프래그먼트 생명주기 변화를 살펴보자. 이를 위해 예제에 프래그먼트를 detach, attach하도록 수정한다.

예제 22-36 프래그먼트 detach, attach 상태 변화

|  |
| --- |
| **src/ MainActivity.java**    **public** **class** MainActivity **extends** Activity **implements** OnBackStackChangedListener {        @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);          ...            getFragmentManager()           .beginTransaction()              .add(R.id.*viewer\_fragment\_container*,  textViewerFragment,"TEXT\_VIEWER")              .add(R.id.*viewer\_fragment\_container*,  imageViewerFragment, "IMAGE\_VIEWER")  **.detach( imageViewerFragment )**  .commit();            ListMenuFragment listMenuFragment = (ListMenuFragment)                     getFragmentManager().findFragmentById( R.id.*menu\_fragment* );            listMenuFragment.setOnListItemClickListener(  **new** ListMenuFragment.OnListItemClickListener()          {              @Override  **public** **void** onItemClick( **int** itemType )              {                  ...    **if**( itemType == ListMenuFragment.*ITEM\_TYPE\_TEXT\_VIEWER* &&                      textViewerFragment.isVisible() == **false** )                  {                      getFragmentManager()                      .beginTransaction()  **.detach( imageViewerFragment )**  **.attach( textViewerFragment )**                          .addToBackStack( "TEXT\_VIEWER\_BACKSTACK" )                      .commit();                  }  **else** **if**( itemType == ListMenuFragment.*ITEM\_TYPE\_IMAGE\_VIEWER* &&                           imageViewerFragment.isVisible() == **false** )                  {                      getFragmentManager()                      .beginTransaction()  **.detach( textViewerFragment )**  **.attach( imageViewerFragment )**                          .addToBackStack( "IMAGE\_VIEWER\_BACKSTACK" )                      .commit();                  }              }          });    ... |

 기존 예제는 텍스트뷰어와 이미지뷰어 프래그먼트를 show, hide하며 화면 우측 영역을 변경해주었다. 수정된 예제는attach, detach를 이용하여 화면 우측 영역을 변경한다. 실행하여 결과를 확인해보자. 참고로 예제의 실행 결과는 반드시[그림 22-36]과 비교하며 확인하도록 하자.

그림 22-38 프래그먼트 attach, detach 상태 변화



①     최초 예제앱을 실행하면, [그림 22-36] ②부터 ④ 과정까지 생명주기 함수가 정상적으로 호출된 것을 확인할 수 있다.

②     화면 우측은 텍스트뷰어 프래그먼트가 보이는 상태다.

③     이미지 보기 리스트 아이템을 누른다.

④     이때 이미지뷰어 프래그먼트는 attach하고, 텍스트뷰어 프래그먼트는 detach한다. 따라서 텍스트뷰어 프래그먼트 생명주기를 보면 [그림 22-36] ⑪ 과정의 생명주기 함수가 정상적으로 호출되었다. 그런데 detach의 경우, 프래그먼트 생명주기 onDestroyView 함수까지만 호출되고 onDestory, onDetach 함수는 호출되지 않았다. 그 이유는 앞서 설명했듯 프래그먼트 객체가 제거되지 않았기 때문이다. detach는 프래그먼트의 레이아웃만 제거한다.그러므로 레이아웃이 제거됨을 의미하는 onDestroyView 함수까지 호출된다.

⑤     이제 화면 우측에는 이미지뷰어 프래그먼트가 보인다.

⑥     이 상태에서 이전키를 누른다.

⑦     이때 최근 추가된 백스택 트랜잭션이 반대로 실행될 것이다. 따라서 텍스트뷰어는 다시 attach된다. 프래그먼트 생명주기 로그를 살펴보면 [그림 22-36] ⑫ 과정의 함수가 정상적으로 호출되었다. attach의 경우 기존 프래그먼트 객체 자체를 재사용하기 때문에 onAttach, onCreate 함수는 호출되지 않았다. 하지만 프래그먼트의 레이아웃은 dettach시 제거되므로 다시 생성되어야 한다. 따라서 onCreateView 생명주기 함수가 호출되고, 해당 함수의 인자로 프래그먼트 레이아웃을 다시 생성하고 반환한다.

 [참고 예제 소스]

[[첨부파일](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D8%26fileid%3D6%26regdt%3D20140926214933&url=http%3A%2F%2Fcfile294.uf.daum.net%2Fattach%2F21649B44542D15C913B3AF')) 22-11. 프래그먼트 라이프 사이클(attach,detach).zip](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D8%26fileid%3D6%26regdt%3D20140926214933&url=http%3A%2F%2Fcfile294.uf.daum.net%2Fattach%2F21649B44542D15C913B3AF'))

**액티비티의 실행 중 프래그먼트 add, remove 상태변화**

 마지막으로 액티비티 실행 중 프래그먼트를 replace(혹은 remove + add)할 때 프래그먼트 생명주기 변화를 살펴보자. 이를 위해 예제에 프래그먼트를 replace(혹은 remove + add)하도록 수정한다.

예제 22-37 프래그먼트 replace(혹은 remove + add) 상태 변화

|  |
| --- |
| **src/ MainActivity.java**    **public** **class** MainActivity **extends** Activity **implements** OnBackStackChangedListener {        @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);          ...            getFragmentManager()           .beginTransaction()              .add(R.id.*viewer\_fragment\_container*,  textViewerFragment,"TEXT\_VIEWER")              .add(R.id.*viewer\_fragment\_container*,  imageViewerFragment, "IMAGE\_VIEWER")  **.remove( imageViewerFragment )**            .commit();            ListMenuFragment listMenuFragment = (ListMenuFragment)                     getFragmentManager().findFragmentById( R.id.*menu\_fragment* );            listMenuFragment.setOnListItemClickListener(  **new** ListMenuFragment.OnListItemClickListener()          {              @Override  **public** **void** onItemClick( **int** itemType )              {                  ...    **if**( itemType == ListMenuFragment.*ITEM\_TYPE\_TEXT\_VIEWER* &&                      textViewerFragment.isVisible() == **false** )                  {                      getFragmentManager()                      .beginTransaction()  **.replace( R.id.*viewer\_fragment\_container*,**  **textViewerFragment )**  .addToBackStack( "TEXT\_VIEWER\_BACKSTACK" )                      .commit();                  }  **else** **if**( itemType == ListMenuFragment.*ITEM\_TYPE\_IMAGE\_VIEWER* &&                           imageViewerFragment.isVisible() == **false** )                  {                      getFragmentManager()                      .beginTransaction()  **.replace( R.id.*viewer\_fragment\_container*,**  **imageViewerFragment)**                          .addToBackStack( "IMAGE\_VIEWER\_BACKSTACK" )                      .commit();                  }              }          });    ... |

 바로 이전 예제는 텍스트뷰어와 이미지뷰어 프래그먼트를 attach, detach하며 화면 우측 영역을 변경해주었다. 수정된 예제는 replace(혹은 remove + add)를 이용하여 화면 우측 영역을 변경한다. 그런데 예제를 실행하면 [그림 22-38] attach, detach의 결과와 같을 것이다. 즉 add 시 onAttach, onCreate 생명주기 함수가 호출되지 않고, remove 시 onDestory, onDetach 함수가 호출되지 않는다는 의미다. 그 이유는 바로 백스택 때문인데 [그림 22-36] ⑩부터 ⑫까지 과정에서 설명되었다. 그렇다면 replace(혹은 remove + add) 시 백스택에 트랜잭션을 추가하지 않으면 호출되지 않은 각각의 생명주기 함수들이 호출될까? 직접 예제로 확인해보자.

 [참고 예제 소스]

[[첨부파일](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D8%26fileid%3D7%26regdt%3D20140926214933&url=http%3A%2F%2Fcfile296.uf.daum.net%2Fattach%2F2358D344542D15DC2F5D58')) 22-12. 프래그먼트 라이프 사이클(add,remove).zip](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D8%26fileid%3D7%26regdt%3D20140926214933&url=http%3A%2F%2Fcfile296.uf.daum.net%2Fattach%2F2358D344542D15DC2F5D58'))

예제 22-38 백스택을 적용하지 않은 프래그먼트 remove, add (replace) 상태 변화

|  |
| --- |
| **src/ MainActivity.java**    **public** **class** MainActivity **extends** Activity **implements** OnBackStackChangedListener {  TextViewerFragment mTextViewerFragment = **null**;      ImageViewerFragment mImageViewerFragment = **null**;        @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);          ...    // ① 프래그먼트를 remove 해버리면 findFragmentById, findFragmentByTag 함수          //    등으로 찾을 수 없다. 따라서 한번 생성한 프래그먼트를 참조할 수 있도록          //    멤버 변수로 저장해둬야한다.  **mTextViewerFragment = TextViewerFragment.*newInstance*();**  **mImageViewerFragment = ImageViewerFragment.*newInstance*();**            getFragmentManager()           .beginTransaction()  // ② 최초 보여질 텍스트뷰어 프래그먼트만 추가해둔다.  **.add(R.id.*viewer\_fragment\_container*,**  **mTextViewerFragment,"TEXT\_VIEWER")**  .commit();            ListMenuFragment listMenuFragment = (ListMenuFragment)                     getFragmentManager().findFragmentById( R.id.*menu\_fragment* );            listMenuFragment.setOnListItemClickListener(  **new** ListMenuFragment.OnListItemClickListener()          {  @Override  **public** **void** onItemClick( **int** itemType )              {  // ③ remove한 프래그먼트는 찾을 수 없다. 따라서 주석처리한다.                  /\*TextViewerFragment textViewerFragment = (TextViewerFragment)                         getFragmentManager().findFragmentByTag( "TEXT\_VIEWER" );                  ImageViewerFragment imageViewerFragment = (ImageViewerFragment)                        getFragmentManager().findFragmentByTag( "IMAGE\_VIEWER" );\*/    **if**( itemType == ListMenuFragment.*ITEM\_TYPE\_TEXT\_VIEWER* &&                      mTextViewerFragment.isVisible() == **false** )                  {                      getFragmentManager()                      .beginTransaction()                          .replace( R.id.*viewer\_fragment\_container*,**mTextViewerFragment** )  // ④ 백스택에 추가하지 않도록 주석처리한다.                          //**.addToBackStack( "TEXT\_VIEWER\_BACKSTACK" )**                      .commit();                  }  **else** **if**( itemType == ListMenuFragment.*ITEM\_TYPE\_IMAGE\_VIEWER* &&                           mImageViewerFragment.isVisible() == **false** )                  {                      getFragmentManager()                      .beginTransaction()                          .replace( R.id.*viewer\_fragment\_container*,**mImageViewerFragment** )  // ⑤ 백스택에 추가하지 않도록 주석처리한다.                          //**.addToBackStack( "IMAGE\_VIEWER\_BACKSTACK" )**                      .commit();                  }              }          });    ... |

①     이 예제에서는 각 뷰어 프래그먼트를 replace(혹은 remove + add)한다. 그런데 한번 remove한 프래그먼트는 프래그먼트매니저를 통해 findFragmentById, findFragmentByTag 함수로 찾을 수 없다. 따라서 한번 생성된 프래그먼트를 계속 참조하기 위해 멤버 변수로 저장해둔다.

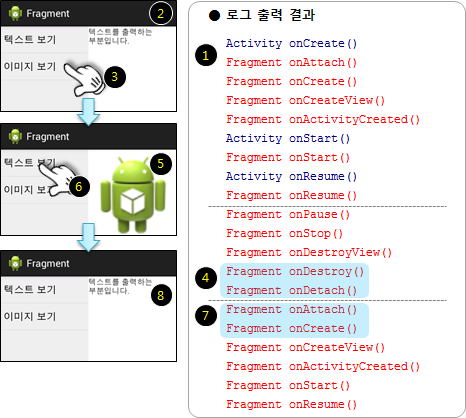
②     화면 좌측에 최초 보여질 텍스트뷰어 프래그먼트만 추가한다.

③     ①에서 설명했듯 remove된 프래그먼트는 프래그먼트매니저에 존재하지 않는다. 따라서 findFragmentById, findFragmentByTag 함수로 찾을 수 없다. 만일 해당 함수를 사용하면 null을 반환할 것이다.

④     화면에 보여질 뷰어 프래그먼트로 replace(혹은 remove + add)하는 트랜잭션은 백스택에 저장해두지 않도록 주석처리한다. ⑤도 동일한 이유로 주석 처리한다.

예제를 실행하여 프래그먼트 add 시 onAttach, onCreate 생명주기 함수가 호출되고, remove 시 onDestory, onDetach 함수가 호출되는지 확인해보자.

그림 22-39 백스택을 적용하지 않은 프래그먼트 replace(혹은 remove + add) 상태 변화



①     최초 예제앱을 실행하면, [그림 22-36] ②부터 ④ 과정까지 생명주기 함수가 정상적으로 호출된 것을 확인할 수 있다.

②     화면 우측은 텍스트뷰어 프래그먼트가 보이는 상태다.

③     이미지 보기 리스트 아이템을 누른다.

④     이때 텍스트뷰어를 이미지뷰어 프래그먼트로 replace(혹은 remove + add) 한다. 따라서 로그를 보면 텍스트뷰어는 remove 되므로 [그림 22-36] ⑧과정의 생명주기 함수가 모두 호출되었다. 여기서 중요한 점은 백스택에 트랜잭션을 추가하지 않았기 때문에 onDestroy, onDetach 생명주기 함수도 호출된 것이다. 즉 텍스트뷰어 프래그먼트 객체도 제거되었다.

⑤     이제 화면 우측에는 이미지뷰어 프래그먼트가 보인다.

⑥     이 상태에서 텍스트 보기 리스트 아이템을 누른다. 참고로 이전 트랜잭션을 백스택에 추가하지 않았기 때문에 이전키를 사용하지 않았다.

⑦     이때 이미지뷰어를 텍스트뷰어 프래그먼트로 replace(혹은 remove + add) 한다. 따라서 로그를 보면 텍스트뷰어는 add 되므로 [그림 22-36] ⑨과정의 생명주기 함수가 모두 호출되었다. 여기서 중요한 점은 이전 트랜잭션을 백스택에 추가하지 않았기 때문에 onAttach, onCreate 생명주기 함수도 호출된 것이다. 즉 텍스트뷰어 프래그먼트는 remove 되었을 때 제거되어 재사용할 수 없다. 그러므로 add 시점에 텍스트뷰어 프래그먼트를 다시 생성한다.

[참고 예제 소스]

[[첨부파일](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D8%26fileid%3D8%26regdt%3D20140926214933&url=http%3A%2F%2Fcfile265.uf.daum.net%2Fattach%2F2552D943542D15EC1B1BCC')) 22-13. 프래그먼트 라이프 사이클(백스택에 추가하지 않고 add,remove).zip](javascript:checkVirus('grpid%3D1MWA2%26fldid%3DpgqT%26dataid%3D8%26fileid%3D8%26regdt%3D20140926214933&url=http%3A%2F%2Fcfile265.uf.daum.net%2Fattach%2F2552D943542D15EC1B1BCC'))