



# Fragment

## Fragment이란 무엇인가

프래그먼트 | Android 개발자 | Android Developers

는 앱 UI의 재사용 가능한 부분을 나타냅니다. 프래그먼트는 자체 레이아웃을 정의 및 관리하고 자체 수명 주기를 보유하며 자체 입력 이벤트를 처리할 수 있습니다. 프래그먼트는 독립적으로 존재할 수 없고 활동이나 다른 프래그먼트에서 호스팅되어야 합니다. 프래그먼트의 뷰 계층 구조는 호

 <https://developer.android.com/guide/fragments?hl=ko>

Developers 

fragment란, 어플리케이션에서 화면에 직접 보이는 공간인 Activity내에서 분할시키고 다른 화면으로 전환할 수 있는 화면 공간의 단위입니다.


여러개의 fragment를 결합해서 하나의 액티비티에 보이게 할 수도 있으며 하나의 fragment를 여러 액티비티에서 재사용할 수 있다.

Fragment를 이용하려면, 그 상위에 있는 Activity에서 출력할 layout을 제어해줘야 합니다.

## Fragment Manager

Handling Lifecycles with Lifecycle-Aware Components | Android Developers

Lifecycle-aware components perform actions in response to a change in the lifecycle status of another component, such as activities and fragments. These components help you produce better-organized, and often lighter-weight code, that is easier to maintain. A

 <https://developer.android.com/topic/libraries/architecture/lifecycle>

Developers 

Fragment Manager는 Activity안에 있는 여러 fragment들을 관리하는 클래스로 fragment를 추가하거나 제거하거나 교체하거나 back stack에 추가하는 일들을 한다.

## Fragment LifeCycle

 [LifeCycle](#)

## fragment와 연결

▷xml파일과 fragment 클래스 연결

```
import androidx.fragment.app.Fragment
```

```
Fragment1 extends Fragment
```

onCreateView

```
return inflater.inflate(R.layout.fragment, container, false)
```

▷fragment.xml의 widget과 fragment.java과 연결

```
ViewGroup rootView = (ViewGroup) inflater.inflate(R.layout.fragment_main, container, false);
```

```
Button button = rootView.findViewById(R.id.button);
```

▷activity와 fragment연결

두가지 방법이 있다. activity의 xml파일에 fragmentview를 할당해주거나 fragment를 담은 container을 담은 공간을 만들어 준 다음 activity 클래스에서 연결해줄 수 있다.

두 상황 모두에서 fragmentcontainerview를 사용하자

fragmentcontainerview 초기화 방법

fragmentcontainerview에는 android:name="com.example.ExampleFragmen"를 넣어서 초기화 할 수 있다. The android:name attribute specifies the class name of the Fragment to instantiate. When the activity's layout is inflated, the specified fragment is instantiated,

FragmentManager을 이용 할 수도 있다.

```
fragmentManager.beginTransaction().replace(R.id.container, mainfragment).commit();
```

## fragment의 전환

▷activity 밖에서 fragment를 바꾸고 싶을 때

fragment의 전환은 activity 그리고 그 안에 있는 fragmentmanager가 관장한다. 따라서 activity에 전환 메소드를 만들고 fragment의 버튼에서는 activity.메소드로 그 메소드를 불러와야 한다.

1. activity에 화면 전환 메서드 만들기 fragment manager는 getSupportFragmentManager를 이용해서 초기화해야 한다.

```
public void onFragmentChanged(int Index) {
    switch (Index) {
        case 0:
            //main
            fragmentManager.beginTransaction()
                .replace(R.id.container_frame,mainFragment).commit();
            break;
        case 1:
            //preview
            fragmentManager.beginTransaction()
                .replace(R.id.container_frame, previewFragment).commit();
            break;
        case 2:
            //time set
            fragmentManager.beginTransaction()
                .replace(R.id.container_frame, timeLockFragment).commit();
            break;
        case 3:
            // app setting
            fragmentManager.beginTransaction()
                .replace(R.id.container_frame, allowAppSettingFragment).commit();
            break;
    }
}
```

2. activity 객체를 만들어서 getActivity로 container을 포함하고 있는 activity를 가져오고 activity안에 있는 화면 전환 메시지를 불러와서 전환한다.

```
MainActivity activity = (MainActivity) getActivity();
activity.onFragmentChanged(0);
```

## activity에서 fragment 안에 있는 view, widget 제어하기

[안드로이드 Android] 프래그먼트(Fragment) 2. Fragment 제어하기-Activity에서 Fragment제어 이전 포스트(Fragment 만들기) 에서레이아웃 xml 파일안에 프래그먼트(Fragment)를 만들어서화면에 보여주는 작업을 소개했습니다. 이번에는이렇게 만들어진 Fragment를 보여주는 Activity에서 Fragment 안에 있는 View들을 제어하는 예제입니다. 간단하게 소개하기 위해activity\_main안에 Button을 하나 만들고이 버튼을 누

 <https://kitesoft.tistory.com/82>

activity에서 fragment view를 바꾸려면 두 가지 방법이 있다

1. fragment안의 view를 직접 가져오는 방법이 있고
2. fragment 클래스 내에 view를 바꾸는 메시지를 만들고 그 메시지를 가져오는 것이다.

두번째 방법이 좋다. fragment의 동작은 fragment내에 만들어 줘야 깔끔하다. 두번째 방법만 정리하겠다.

fragment안에 view를 바꾸는 메시지를 만든다

```
public void setTextView(String str){
    positive_text.setText(str);
}
```

fragment 객체를 가져와서 fragment 내의 메시지를 가져온다

```
mainFragment = new MainFragment();
mainFragment.setTextView();
```

## Interface를 이용해 fragment의 정보 교환

▷정보를 주는 fragment에서 listener interface를 만든다

```
public static interface DialogListener {
    public void textChange(String str);
}
```

▷activity가 fragment A 안의 interface를 implement하게 하고 interface의 메서드를 activity에서 구현한다. 그 메서드 안에 activity에서 할 동작을 적는다

Mainactivity extends AppCompatActivity implements  
CustomDialog.DialogListener

```
@Override
public void textChange(String str) {
//이 안에 activity에서 작동하는 코드를 입력한다
}
```

▷activity와 fragment가 연결되는 순간 fragment의 onAttach에서 activity context안에 담긴 listener를 가져온다. 그 안에 activity에서 만든 코드가 포함되어 들어온다

try catch문을 이용하면 좋다. 이러면 implement되지 않았으면 context안에 있는 callback을 가져오지 못하고 넘어간다

```
//val
DialogListener listener;

@Override
public void onAttach(@NonNull Context context) {
super.onAttach(context);
// ctrl+alt+t
try {
listener = (DialogListener) context;
} catch (ClassCastException e) {
throw new ClassCastException(context.toString() +
"must implement DialogListener")
}
}
```

▷onAttach를 통해 가져온 listener 객체로 activity의 메서드를 참조하고 그 메서드에 값을 넘겨줌으로서 정보를 전달한다

```
@Override
public void onClick(DialogInterface dialogInterface, int i) {
str_positive = positiveText.getText().toString();
listener.textChange(str_positive);
}
```

## ?VIEWMODEL

### Communicating with fragments | Android Developers

To reuse fragments, build each as a completely self-contained component that defines its own layout and behavior. Once you have defined these reusable fragments, you can associate them with an activity and connect them with the application logic to realize the

 <https://developer.android.com/guide/fragments/communicate>

Developers 

▷ To properly react to user events, or to share state information, you often need to have channels of communication between an activity and its fragments or between two or more fragments. To keep fragments self-contained, you should not have fragments communicate directly with other fragments or with its host activity.

쉽게 설명을 하면 두 fragment는 직접적으로 통신을 하면 안된다. 그래서 보통은 communicate 전용 interface를 만든 다음 이것을 activity에 implement하고 그 interface를 통해 소통한다.

▷ fragment 사이나 fragment와 activity 사이 user event, state information 공유 방법은 두가지가 있다.

ViewModel: 더 광범위한 상황에서 지속해야하는 data를 주고 받을 때

Fragment Result API: 일회성으로 넘겨주고 쓸나는 data이고 Bundle에 들어갈 크기면

#### ▶ ViewModel

▷ viewmodel을 사용해야 할 때

activity 안에 두개의 fragment가 서로 소통을 해야 할 때

fragment 안에 또 다른 fragment가 있고 서로 소통을 할 때: fragment에서 dialog나 picker을 띄워도 이들은 activity 위에서 생성되는 것이다.

navigation view와 fragment와 소통을 할 때

#### ☆ ViewModel 이용하기

viewmodel에 필요한 dependencies를 추가하고

viewmodel java class를 만들고 viewmodel을 상속한다

```
private MutableLiveData<원하는 type> text = new MutableLiveData<>();
```

mutable을 사용해야만 setText를 사용해서 추가할 수 있다.

data를 더러하고 빼는 메서드를 추가한다.

fragmentA에 들어가서 onActivityCreated override해서 activity와 observer설정한다

---

#### ◇ Fragment에서 Toast 사용하기

<https://horae.tistory.com/entry/안드로이드-Fragment-에서-Toast-사용하기>

toast를 사용하려면 toast가 표시되는 context가 필요하다.

Context 객체 만들기

fragment를 감싸고 있는 context 가져오기

```
makeText(context, "string", length)
```

#### ◇ FragmentManager

<https://developer.android.com/guide/fragments/fragmentmanager>

▷ FragmentManager is the class responsible for performing actions on your app's fragments, such as adding, removing, or replacing them, and adding them to the back stack.

▷ Every FragmentActivity and subclasses thereof, such as AppCompatActivity, have access to the FragmentManager through the getSupportFragmentManager() method.

▷ Fragment는 다른 Fragment를 안에 품을 수 있으며 getChildFragmentManager(), getParentFragmentManager() 메서드를 통해 각각의 fragmentmanager을 참조할 수 있다.

#### ◇ Fragment Transactions

<https://developer.android.com/guide/fragments/transactions>

fragmentmanager는 fragment의 추가, 제거, 교체를 관장하는데 fragmenttransaction class를 통해 fragment 변경이 가능하다. 이때 manager가 관리하는 back stack에 transaction 과정을 저장해서 뒤로 돌아갈 수 있다.

```
FragmentManager fragmentManager = getFragmentManager();
```

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

```
▷ .beginTransaction()
```

```
▷ .setReorderingAllowed()
```

```
▷ .add()
```

```
▷ .remove()
```

```
▷ .replace()
```

```
▷ .commit()
```

```
▷ .show() .hide()
```

```
▷ .detach() .attach()
```