Фрагменты



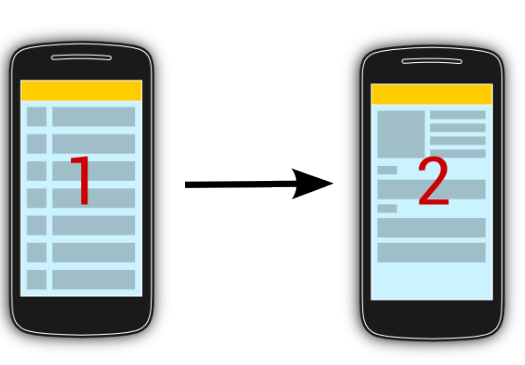
Существует два основных подхода в использовании фрагментов.

Первый способ основан на замещении родительского контейнера. Создаётся стандартная разметка и в том месте, где будут использоваться фрагменты, размещается контейнер, например, **FrameLayout**. В коде контейнер замещается фрагментом. При использовании подобного сценария в разметке не используется тег **fragment**, так как его нельзя менять динамически. Также вам придётся обновлять ActionBar, если он зависит от фрагмента. Здесь показан [такой пример](http://developer.alexanderklimov.ru/android/theory/fragment-add.php).

Второй вариант - используются отдельные разметки для телефонов и планшетов, которые можно разместить в разных папках ресурсов. Например, если в планшете используется двухпанельная разметка с двумя фрагментами на одной активности, мы используем эту же активность для телефона, но подключаем другую разметку, которая содержит один фрагмент. Когда нам нужно переключиться на второй фрагмент, то запускаем вторую активность.

Второй подход является наиболее гибким и в целом предпочтительным способом использования фрагментов. Активность проверяет в каком режиме (свои размеры) он запущен и использует разную разметку из ресурсов. Графически это выглядит следующим образом.





Основные классы

Сами фрагменты наследуются от **android.app.Fragment**. Существует подклассы фрагментов: **ListFragment**, **DialogFragment**, **PreferenceFragment**, **WebViewFragment** и др. Не исключено, что число классов будет увеличиваться, например, появился ещё один класс **MapFragment**.

Для взаимодействия между фрагментами используется класс **android.app.FragmentManager** - специальный менеджер по фрагментам.

Как в любом офисе, спецманагер не делает работу своими руками, а использует помощников. Например, для транзакций (добавление, удаление, замена) используется класс-помощник **android.app.FragmentTransaction**.

Для сравнения приведу названия классов из библиотеки совместимости:

* android.support.v4.app.FragmentActivity
* android.support.v4.app.Fragment
* android.support.v4.app.FragmentManager
* android.support.v4.app.FragmentTransaction

Как видите, разница в одном классе, который я привёл первым. Он используется вместо стандартного **Activity**, чтобы система поняла, что придётся работать с фрагментами. На данный момент студия создаёт проект на основе **ActionBarActivity**, который является подклассом **FragmentActivity**.

В одном приложении нельзя использовать новые фрагменты и фрагменты из библиотеки совместимости.

В 2018 году Гугл объявила фрагменты из пакета **androd.app** устаревшими. Заменяйте везде на версию из библиотеки совместимости.

Общий алгоритм работы с фрагментами будет следующим:

У каждого фрагмента должен быть свой класс. Класс наследуется от класса **Fragment** или схожих классов, о которых говорилось выше. Это похоже на создание новой активности или нового компонента.

Также, как в активности, вы создаёте различные методы типа **onCreate()** и т.д. Если фрагмент имеет разметку, то используется метод **onCreateView()** - считайте его аналогом метода **setContentView()**, в котором вы подключали разметку активности. При этом метод **onCreateView()** возвращает объект **View**, который является корневым элементом разметки фрагмента.

Разметку для фрагмента можно создать программно или декларативно через XML.

Создание разметки для фрагмента ничем не отличается от создания разметки для активности. Вот отрывок кода из метода **onCreateView()**:

public class FirstFragment extends Fragment implements OnClickListener {

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

View view = inflater.inflate(R.layout.first\_fragment,

container, false);

Button nextButton = (Button) view.findViewById(R.id.button\_first);

nextButton.setOnClickListener(this);

return view;

}

// ...

}

Глядя на этот код, вы должные понять, что фрагмент использует разметку из файла **res/layout/first\_fragment.xml**, которая содержит кнопку с идентификатором **android:id="@+id/button\_first"**. Здесь также прослеживается сходство с подключением компонентов в активности. Обратите внимание, что перед методом **findViewById()**используется **view**, так как этот метод относится к компоненту, а не к активности, как мы обычно делали в программах, когда просто опускали имя активности. Т.е. в нашем случае мы ищем ссылку на кнопку не среди разметки активности, а внутри разметки самого фрагмента.

Нужно помнить, что в методе **inflate()** последний параметр должен иметь значение **false** в большинстве случаев.

FragmentManager

Класс **FragmentManager** имеет два метода, позволяющих найти фрагмент, который связан с активностью:

**findFragmentById(int id)**

Находит фрагмент по идентификатору

**findFragmentByTag(String tag)**

Находит фрагмент по заданному тегу

Методы транзакции

Мы уже использовали некоторые методы класса **FragmentTransaction**. Познакомимся с ними поближе

**add()**

Добавляет фрагмент к активности

**remove()**

Удаляет фрагмент из активности

**replace()**

Заменяет один фрагмент на другой

**hide()**

Прячет фрагмент (делает невидимым на экране)

**show()**

Выводит скрытый фрагмент на экран

**detach() (API 13)**

Отсоединяет фрагмент от графического интерфейса, но экземпляр класса сохраняется

**attach() (API 13)**

Присоединяет фрагмент, который был отсоединён методом **detach()**

Методы **remove()**, **replace()**, **detach()**, **attach()** не применимы к статичным фрагментам.

Перед началом транзакции нужно получить экземпляр **FragmentTransaction** через метод **FragmentManager.beginTransaction()**. Далее вызываются различные методы для управления фрагментами.

В конце любой транзакции, которая может состоять из цепочки вышеперечисленных методов, следует вызвать метод **commit()**.

FragmentManager fragmentManager = getFragmentManager()

fragmentManager.beginTransaction()

.remove(fragment1)

.add(R.id.fragment\_container, fragment2)

.show(fragment3)

.hide(fragment4)

.commit();

Аргументы фрагмента

Фрагменты должны сохранять свою модульность и не должны общаться друг с другом напрямую. Если один фрагмент хочет докопаться до другого, он должен сообщить об этом своему менеджеру активности, а он уже передаст просьбу другому фрагменту. И наоборот. Это сделано специально для того, чтобы было понятно, что менеджер тут главный и он не зря зарплату получает. Есть три основных способа общения фрагмента с активностью.

* Активность может создать фрагмент и установить аргументы для него
* Активность может вызвать методы экземпляра фрагмента
* Фрагмент может реализовать интерфейс, который будет использован в активности в виде слушателя

Фрагмент должен иметь только один пустой конструктор без аргументов. Но можно создать статический **newInstance** с аргументами через метод **setArguments()**.

public class CatFragment extends Fragment {

public static CatFragment newInstance(int someInt, String someString) {

CatFragment catFragment = new CatFragment();

Bundle args = new Bundle();

args.putInt("someInt", someInt);

args.putString("SomeString", someString);

catFragment.setArguments(args);

return catFragment;

}

}

Доступ к аргументам можно получить в методе **onCreate()** фрагмента:

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

// Get back arguments

int someInt = getArguments().getInt("someInt", 0);

String someString = getArguments().getString("someString", "");

}

Динамически загружаем фрагмент в активность.

FragmentTransaction ft = getSupportFragmentManager().beginTransaction();

CatFragment catFragment = CatFragment.newInstance(5, "Васька");

ft.replace(R.id.your\_placeholder, catFragment);

ft.commit();

Если активность должна выполнить какую-то операцию в фрагменте, то самый простой способ - задать нужный метод в фрагменте и вызвать данный метод через экземпляр фрагмента.

public class CatFragment extends Fragment {

public void sayMeow(String word) {

// do something in fragment

}

}

Вызываем метод в активности:

public class MainActivity extends ActionBarActivity {

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

CatFragment catFragment = (CatFragment)

getSupportFragmentManager().findFragmentById(R.id.fragmentCat);

catFragment.sayMeow("Жрать!");

}

}

Если фрагмент должен сообщить о своих действиях активности, то следует реализовать интерфейс.

import android.content.Context;

import android.support.v4.app.Fragment;

import android.view.View;

public class LinkListFragment extends Fragment {

// Define the listener of the interface type

// listener is the activity itself

private OnLinkItemSelectedListener mListener;

// Define the events that the fragment will use to communicate

public interface OnLinkItemSelectedListener {

public void onLinkItemSelected(String link);

}

// Store the listener that will have events fired once the fragment is attached

@Override

public void onAttach(Context context) {

super.onAttach(context);

if (context instanceof OnLinkItemSelectedListener) {

mListener = (OnLinkItemSelectedListener) context;

} else {

throw new ClassCastException(context.toString()

+ " must implement MyListFragment.OnItemSelectedListener");

}

}

// Now we can fire the event when the user selects something in the fragment

public void onSomeClick(View v) {

mListener.onLinkItemSelected("some link");

}

}

В активности:

public class MainActivity extends ActionBarActivity **implements**

**LinkListFragment.OnLinkItemSelectedListener** {

DetailFragment fragment;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

fragment = (DetailFragment) getSupportFragmentManager()

.findFragmentById(R.id.detailFragment);

}

// Now we can define the action to take in the activity when the fragment event fires

@Override

public void onLinkItemSelected(String link) {

if (fragment != null && fragment.isInLayout()) {

fragment.setText(link);

}

}

}

Управление стеком фрагментов

Фрагменты, как и активности, могут управляться кнопкой **Back**. Вы можете добавить несколько фрагментов, а потом через кнопку **Back** вернуться к первому фрагменту. Если в стеке не останется ни одного фрагмента, то следующее нажатие кнопки закроет активность.

Чтобы добавить транзакцию в стек, вызовите метод **FragmentTransaction.addToBackStack(String)** перед завершением транзакции (commit). Строковый аргумент - опциональное имя для идентификации стека или **null**. Класс **FragmentManager** имеет метод **popBackStack()**, возвращающий предыдущее состояние стека по этому имени.

Если вы вызовете метод **addToBackStack()** при удалении или замещении фрагмента, то будут вызваны методы фрагмента **onPause(), onStop(), onDestroyView()**.

Когда пользователь нажимает на кнопку возврата, то вызываются методы фрагмента **onCreateView()**, **onActivityCreated()**, **onStart()** и **onResume()**.

Рассмотрим пример реагирования на кнопку **Back** в фрагменте без использования стека. Активность имеет метод **onBackPressed()**, который реагирует на нажатие кнопки. Мы можем в этом методе сослаться на нужный фрагмент и вызвать метод фрагмента.

@Override

public void onBackPressed() {

super.onBackPressed();

// где-то ранее мы объявили фрагмент

fragment1.backButtonWasPressed();

}

Теперь в классе фрагмента прописываем метод с нужным кодом.

public void backButtonWasPressed() {

Toast.makeText(getActivity(), "Back button pressed", Toast.LENGTH\_LONG)

.show();

}

Более желательным вариантом является использование интерфейсов. В некоторых примерах с фрагментами такой приём используется.

Интеграция Action Bar/Options Menu

Фрагменты могут добавлять свои элементы в панель действий или меню активности. Сначала вы должны вызвать метод **Fragment.setHasOptionsMenu()** в методе фрагмента **onCreate()**. Затем нужно задать настройки для методов фрагмента **onCreateOptionsMenu()** и **onOptionsItemSelected()**, а также при необходимости для методов **onPrepareOptionsMenu()**, **onOptionsMenuClosed()**, **onDestroyOptionsMenu()**. Работа методов фрагмента ничем не отличается от аналогичных методов для активности.

В активности, которая содержит фрагмент, данные методы автоматически сработают.

Если активность содержит собственные элементы панели действий или меню, то следует позаботиться, чтобы они не мешали вызовам методам фрагментов.

Код для активности:

public class MyActivity extends Activity {

// ...

@Override

public boolean onCreateOptionsMenu(Menu menu) {

super.onCreateOptionsMenu(menu);

getMenuInflater().inflate(R.menu.activity\_options, menu);

return true;

}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

switch (item.getItemId()) {

case R.id.menu\_activity\_info:

// Handle activity menu item

return true;

default:

// Handle fragment menu items

return super.onOptionsItemSelected(item);

}

}

// ...

}

Код для фрагмента:

public class MyFragment extends Fragment {

// ...

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setHasOptionsMenu(true);

}

@Override

public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {

inflater.inflate(R.menu.myfragment\_options, menu);

}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

switch (item.getItemId()) {

case R.id.menu\_first\_info:

// Handle fragment menu item

return true;

default:

// Not one of ours. Perform default menu processing

return super.onOptionsItemSelected(item);

}

}

// ...

}

Связь между фрагментом и активностью

Экземпляр фрагмента связан с активностью. Активность может вызывать методы фрагмента через ссылку на объект фрагмента. Доступ к фрагменту можно получить через методы **findFragmentById()** или **findFragmentByTag()**.

Фрагмент в свою очередь может получить доступ к своей активности через метод **Fragment.getActivity()**.

View listView = getActivity().findViewById(R.id.list);

# Fragment

Компонент **Fragment** находится в папке **Layouts**, что позволяет рассматривать его как контейнер. В то же время фрагмент обладает функциональностью активности (жизненный цикл, схожие методы и т.д.).

Сами фрагменты появились в API 11, но в целях совместимости была написана специальная библиотека **Android Support library** для старых устройств, которая по умолчанию готова к использованию в новых проектах.

Фрагментам будет посвящен отдельный цикл статей. Начать можно [отсюда](http://developer.alexanderklimov.ru/android/theory/fragments.php).

Здесь я покажу базовый пример использования фрагментов.

## Базовый пример с фрагментом

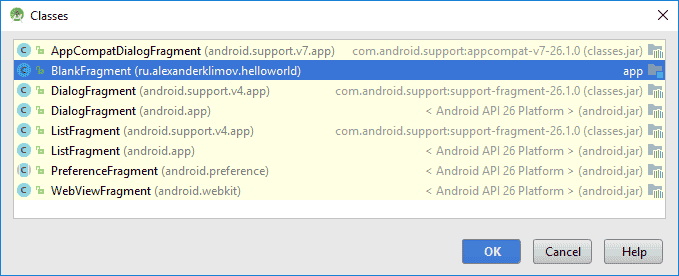
Рассмотрим простейший пример с использованием фрагмента, чтобы понять основные принципы работы.

Сначала нам нужно создать класс фрагмента. Воспользуемся услугами студии и создадим фрагмент при помощи мастера **File | New | Fragment | Fragment (Blank)**. В диалоговом окне оставляем всё без изменений.

Будет создать большой и страшный код для класса.

Оставим в покое этот класс и создадим макет. В палитре инструментов выберите раздел **Layouts** и перетащите на вашу заготовку экрана элемент **Fragment**.

Среда разработки предложит вам выбрать из списка готовый класс, который наследуется от класса **Fragment** или его потомков. Среди них будет созданный нами только что класс **BlankFragment**.



Код в классе фрагмента содержит интерфейс, который необходимо реализовать в активности, иначе мы получим ошибку.

package ru.alexanderklimov.helloworld;

import android.net.Uri;

import android.os.Bundle;

import android.support.v7.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity

**implements BlankFragment.OnFragmentInteractionListener** {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

**@Override**

**public void onFragmentInteraction(Uri uri) {**

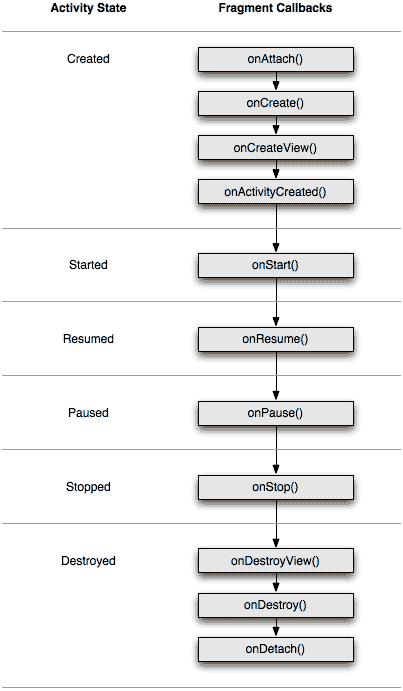
**}**

}

Запустим код и увидим, что внутри активности есть фрагмент (текст "Hello blank fragment").

# Жизненный цикл фрагментов

У фрагментов есть жизненный цикл, который во многом совпадает с жизненным циклом активности, внутри которой они находятся.



Список дополнительных методов жизненного цикла фрагментов, которых нет у активности:

**onAttach(Activity)**

Вызывается, когда фрагмент связывается с активностью. С этого момента мы можем получить ссылку на активность через метод **getActivity()**

**onCreate()**

В этом методе можно сделать работу, не связанную с интерфейсом. Например, подготовить адаптер.

**onCreateView(LayoutInflater, ViewGroup, Bundle)**

Вызывается для создания компонентов внутри фрагмента

**onActivityCreated(Bundle)**

Вызывается, когда отработает метод активности **onCreate()**, а значит фрагмент может обратиться к компонентам активности

**onDestroyView()**

Вызывается, когда набор компонентов удаляется из фрагмента

**onDetach()**

Вызывается, когда фрагмент отвязывается от активности

Одноимённые с методами активности методы фрагментов выполняют аналогичные функции. К примеру, метод **onResume()**вызывается, когда фрагмент вновь становится видимым.

Метод **onStart()** вызывается, когда фрагмент становится видимым после запуска такого же метода в родительской активности.

Фрагмент всегда связан с активностью. Отдельно фрагмент от активности существовать не может.

Если активность останавливается, то её фрагменты также останавливаются. Если активность уничтожается, то её фрагменты также уничтожаются.

Метод **onCreateView()** вызывается один раз, когда фрагмент должен загрузить на экран свой интерфейс. В этом методе вы можете "надуть" (inflate) разметку фрагмента через метод **inflate()** объекта **Inflater**, который задан в параметре метода. В фрагментах без интерфейса вы можете пропустить надувание.

Метод **onActivityCreated()** вызывается после метода **onCreateView()**, когда создаётся активность-хозяйка для фрагмента. Здесь можно объявить объекты, необходимые для **Context**.

Фрагменты не являются подклассами **Context**, вам следует использовать метод **getActivity()**, чтобы получить родительскую активность.

Создадим несколько фрагментов и с помощью всплывающих сообщений и логов посмотрим на срабатывание методов.

### FirstFragment

package ru.alexanderklimov.fragmentdemo;

import android.app.Activity;

import android.net.Uri;

import android.os.Bundle;

import android.app.Fragment;

import android.util.Log;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.Toast;

public class FirstFragment extends Fragment {

public static FirstFragment newInstance(String param1, String param2) {

FirstFragment fragment = new FirstFragment();

// Bundle args = new Bundle();

return fragment;

}

public FirstFragment() {

// Required empty public constructor

}

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

Toast.makeText(getActivity(), "FirstFragment.onCreate()",

Toast.LENGTH\_LONG).show();

Log.d("Fragment 1", "onCreate");

}

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

View rootView = inflater.inflate(R.layout.fragment\_first, container, false);

Toast.makeText(getActivity(), "FirstFragment.onCreateView()",

Toast.LENGTH\_LONG).show();

Log.d("Fragment 1", "onCreateView");

return rootView;

}

@Override

public void onAttach(Activity activity) {

super.onAttach(activity);

Toast.makeText(getActivity(), "FirstFragment.onAttach()",

Toast.LENGTH\_LONG).show();

Log.d("Fragment 1", "onAttach");

}

@Override

public void onActivityCreated(Bundle savedInstanceState) {

super.onActivityCreated(savedInstanceState);

Toast.makeText(getActivity(), "FirstFragment.onActivityCreated()",

Toast.LENGTH\_LONG).show();

Log.d("Fragment 1", "onActivityCreated");

}

@Override

public void onStart() {

super.onStart();

Toast.makeText(getActivity(), "FirstFragment.onStart()",

Toast.LENGTH\_LONG).show();

Log.d("Fragment 1", "onStart");

}

@Override

public void onResume() {

super.onResume();

Toast.makeText(getActivity(), "FirstFragment.onResume()",

Toast.LENGTH\_LONG).show();

Log.d("Fragment 1", "onResume");

}

@Override

public void onPause() {

super.onPause();

Toast.makeText(getActivity(), "FirstFragment.onPause()",

Toast.LENGTH\_LONG).show();

Log.d("Fragment 1", "onPause");

}

@Override

public void onStop() {

super.onStop();

Toast.makeText(getActivity(), "FirstFragment.onStop()",

Toast.LENGTH\_LONG).show();

Log.d("Fragment 1", "onStop");

}

@Override

public void onDestroyView() {

super.onDestroyView();

Toast.makeText(getActivity(), "FirstFragment.onDestroyView()",

Toast.LENGTH\_LONG).show();

Log.d("Fragment 1", "onDestroyView");

}

@Override

public void onDestroy() {

super.onDestroy();

Toast.makeText(getActivity(), "FirstFragment.onDestroy()",

Toast.LENGTH\_LONG).show();

Log.d("Fragment 1", "onDestroy");

}

@Override

public void onDetach() {

super.onDetach();

//mListener = null;

Toast.makeText(getActivity(), "FirstFragment.onDetach()",

Toast.LENGTH\_LONG).show();

Log.d("Fragment 1", "onDetach");

}

}

Разметка для первого фрагмента

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical" >

<TextView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Первый фрагмент" />

<ImageView

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:src="@drawable/ic\_launcher"/>

</LinearLayout>

По аналогии создайте второй фрагмент **SecondFragment**, заменив тексты и названия идентификаторов там, где это нужно.

Создадим разметку для главной активности с двумя кнопками для переключения между фрагментами.

### activity\_main.xml

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:orientation="horizontal">

<LinearLayout

android:layout\_width="0px"

android:layout\_weight="1"

android:layout\_height="match\_parent"

android:orientation="vertical">

<TextView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Выберите фрагмент:" />

<Button

android:id="@+id/buttonFragment1"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Fragment 1" />

<Button

android:id="@+id/buttonFragment2"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Fragment 2" />

</LinearLayout>

<FrameLayout

android:id="@+id/container"

android:layout\_width="0px"

android:layout\_weight="3"

android:layout\_height="match\_parent" />

</LinearLayout>

### MainActivity.java

В главной активности можно переключаться между фрагментами при нажатии на кнопку, а также через кнопку BACK, чтобы увидеть транзакции фрагментов.

package ru.alexanderklimov.fragmentdemo;

import android.app.Fragment;

import android.app.FragmentManager;

import android.app.FragmentTransaction;

import android.support.v7.app.ActionBarActivity;

import android.os.Bundle;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.widget.Button;

public class MainActivity extends ActionBarActivity {

Button fragment1Button, fragment2Button;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

fragment1Button = (Button) findViewById(R.id.buttonFragment1);

fragment2Button = (Button) findViewById(R.id.buttonFragment2);

// get an instance of FragmentTransaction from your Activity

FragmentManager fragmentManager = getFragmentManager();

FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();

//add a fragment

FirstFragment firstFragment = new FirstFragment();

fragmentTransaction.add(R.id.container, firstFragment);

fragmentTransaction.commit();

fragment1Button.setOnClickListener(onButtonClickListener);

fragment2Button.setOnClickListener(onButtonClickListener);

}

Button.OnClickListener onButtonClickListener = new Button.OnClickListener() {

@Override

public void onClick(View v) {

// TODO Auto-generated method stub

Fragment newFragment = null;

// Create new fragment

if (v == fragment1Button) {

newFragment = new FirstFragment();

} else {

newFragment = new SecondFragment();

}

// Create new transaction

FragmentTransaction transaction = getFragmentManager().beginTransaction();

// Replace whatever is in the fragment\_container view with this fragment,

// and add the transaction to the back stack

transaction.replace(R.id.container, newFragment);

transaction.addToBackStack(null);

// Commit the transaction

transaction.commit();

}

};

}

Cмотрим сообщения. При запуске приложения запускается первый фрагмент, который загружается в контейнер. Последовательность следующая.

onAttach  
onCreate  
onCreateView  
onActivityCreated  
onStart  
onResume

Нажимаем на кнопку Home:

onPause  
onStop

Запустим из списка недавно запущенных программ:

onStart  
onResume

Выходим из программы через кнопку Back:

onPause  
onStop  
onDestroyView  
onDestroy  
onDetach

При замещении первого фрагмента вторым почти как при закрытии, только нет методов **onDestroy** и **onDetach**, а затем повторяются те же методы при старте фрагмента:

// Первый фрагмент onPause  
onStop  
onDestroyView  
// Второй фрагмент onAttach  
onCreate  
onCreateView  
onActivityCreated  
onStart  
onResume

# Программное добавление фрагментов

Рассмотрим программное добавление фрагмента в активность. Суть состоит в следующем. В разметке активности нельзя прописывать тег **fragment**, так как в этом случае Android не сможет менять фрагменты динамически. Воспользуемся **LinearLayout**как контейнером и будем его заменять программно на нужный фрагмент.

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:orientation="horizontal" >

<TextView

android:layout\_width="0px"

android:layout\_height="match\_parent"

android:layout\_weight="1"

android:background="#555555"

android:text="TextView в главной Activity" />

<LinearLayout

android:orientation="vertical"

android:id="@+id/container"

android:layout\_width="0px"

android:layout\_height="match\_parent"

android:layout\_weight="4" />

</LinearLayout>

**LinearLayout** с идентификатором **@+id/container** является нашим контейнером. Подготовим класс фрагмента. Разметку для фрагмента использовать не будем, всё сделаем программно.

package ru.alexanderklimov.test;

import android.app.Fragment;

import android.content.Context;

import android.graphics.Color;

import android.os.Bundle;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.LinearLayout;

import android.widget.TextView;

public class MyFragment extends Fragment {

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

Context context = getActivity().getApplicationContext();

LinearLayout layout = new LinearLayout(context);

layout.setBackgroundColor(Color.BLUE);

TextView text = new TextView(context);

text.setText("Это область фрагмента");

layout.addView(text);

return layout;

}

}

Мы закрасили фрагмент синим цветом, добавили текстовую метку и вывели текст.

Осталось добавить код в метод **onCreate()** основной активности. Нам нужно получить экземпляр класса **FragmentTransaction** и добавить фрагмент в контейнерный **LinearLayout**.

package ru.alexanderklimov.fragment;

import android.app.Activity;

import android.app.Fragment;

import android.app.FragmentManager;

import android.app.FragmentTransaction;

import android.os.Bundle;

public class FragmentDemoActivity extends Activity {

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

// получаем экземпляр FragmentTransaction

FragmentManager fragmentManager = getFragmentManager();

FragmentTransaction fragmentTransaction = fragmentManager

.beginTransaction();

// добавляем фрагмент

MyFragment myFragment = new MyFragment();

fragmentTransaction.add(R.id.container, myFragment);

fragmentTransaction.commit();

}

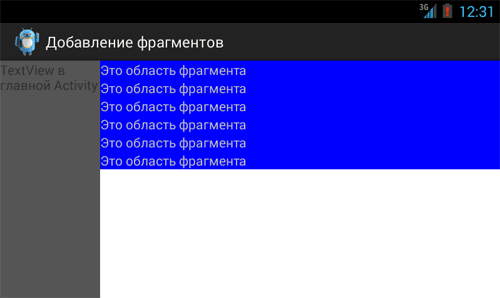
}

В примере мы добавляли фрагмент в контейнер. Если у вас фрагментов несколько и их нужно выводить в одном месте, то их нужно не добавлять в контейнер, а замещать. Подробнее в статье [Замещение фрагментов](http://developer.alexanderklimov.ru/android/theory/fragment-replace.php).

Также следует обращать внимание на поведение фрагмента при поворотах экрана. Для статичных фрагментов это не является проблемой, так как активность сама позаботится о восстановлении своих элементов разметки.

Для динамических фрагментов возможно ситуация, что при восстановлении активности будет создаваться второй экземпляр фрагмента. Если вы ещё не закрыли предыдущий пример, то поверните экран в другую ориентацию. При повороте будет создана новая активность, которая создаст новый фрагмент. Верните устройство в обратное положение и снова активность создаст новый фрагмент.

После шести поворотов приложение будет выглядеть следующим образом:



Тут нам пригодится параметр **Bundle** в методе активности **onCreate()**. Если его значение равно **null**, значит мы впервые запускаем активность и можем спокойно создавать новый экземпляр фрагмента. Помещаем наш код в блок **if**:

public void onCreate(Bundle savedInstanceState) {

// ...

if (savedInstanceState == null) {

FragmentManager fragmentManager = getFragmentManager();

FragmentTransaction fragmentTransaction = fragmentManager

.beginTransaction();

// добавляем фрагмент

MyFragment myFragment = new MyFragment();

fragmentTransaction.add(R.id.container, myFragment);

fragmentTransaction.commit();

}

}

В нашем примере фрагмент не использует разметку и метод **onCreateView()** для него необязателен. Вы можете добавить фрагмент в активность также через метод **FragmentTransaction.add(Fragment, String)**, передав методу уникальную строку для идентификации нужного фрагмента.

FragmentManager fragmentManager = getFragmentManager()

FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();

BackgroundFragment fragment = new BackgroundFragment();

fragmentTransaction.add(fragment, "thread\_manager");

fragmentTransaction.commit();

В первом примере создание новых фрагментов носило неуправляемый характер. Рассмотрим пример, когда мы сознательно хотим добавлять фрагменты в активность.

Изменим немного разметку, добавив две кнопки:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical" >

<LinearLayout

android:id="@+id/linearlayout"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" >

<Button

android:id="@+id/button1"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:layout\_weight="1"

android:onClick="onClick"

android:text="Добавить синий фрагмент" />

<Button

android:id="@+id/button2"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:layout\_weight="1"

android:onClick="onClick"

android:text="Добавить красный фрагмент" />

</LinearLayout>

<LinearLayout

android:id="@+id/container"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical" />

</LinearLayout>

Мы будем динамически добавлять два разных фрагмента. Переименуйте класс **MyFragment** в **BlueFragment** и создайте его копию под именем **RedFragment**. Измените код во втором фрагменте, чтобы область заливалась красным цветом.

Логика приложения теперь будет сосредоточена в методе **onClick()**, который отвечает за нажатия кнопок:

package ru.alexanderklimov.test;

import ...

public class TestActivity extends Activity {

private FragmentManager mFragmentManager;

private FragmentTransaction mFragmentTransaction;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_test);

mFragmentManager = getFragmentManager();

}

public void onClick(View v) {

mFragmentTransaction = mFragmentManager.beginTransaction();

switch (v.getId()) {

case R.id.button1:

BlueFragment bluefragment = new BlueFragment();

mFragmentTransaction.add(R.id.container, bluefragment);

break;

case R.id.button2:

RedFragment redfragment = new RedFragment();

mFragmentTransaction.add(R.id.container, redfragment);

break;

}

mFragmentTransaction.commit();

}

}

Запускаем проект и щёлкаем по кнопкам в произвольном порядке.



# Замещение фрагментов

Мы рассматривали [пример программного добавления фрагмента](http://developer.alexanderklimov.ru/android/theory/fragment-add.php) при помощи метода **add()**, который является только одним из способом замещения контейнера фрагментом.

В более сложных программах, состоящих из нескольких фрагментов, используется метод **replace()**, который замещает фрагмент в контейнере другим фрагментом.

Рассмотрим на примере. Создадим активность с тремя кнопками и контейнером **FrameLayout**:

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:paddingBottom="@dimen/activity\_vertical\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

tools:context=".TestActivity" >

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="horizontal" >

<Button

android:id="@+id/button1"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:layout\_weight="1"

android:text="Фрагмент 1" />

<Button

android:id="@+id/button2"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:layout\_weight="1"

android:text="Фрагмент 2" />

<Button

android:id="@+id/button3"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:layout\_weight="1"

android:text="Фрагмент 3" />

</LinearLayout>

<FrameLayout

android:id="@+id/container"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" >

</FrameLayout>

</LinearLayout>

Создадим разметку для трёх фрагментов. Я приведу код только для первой разметки. В других можете использовать тот же код, только заменить текст в TextView, чтобы было понятно, о каком фрагменте идёт речь. По желанию можете также использовать разные картинки в ImageView:

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/fragment1"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical" >

<TextView

android:id="@+id/tvMessage"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Фрагмент 1" />

<ImageView

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:src="@drawable/ic\_launcher" />

</LinearLayout>

Теперь напишем код для замещения фрагментов. Обратите внимание, что классы фрагментов мы используем в основном классе активности. Хотя в реальных проектах лучше выделить каждому классу фрагмента отдельный файл.

package ru.alexanderklimov.fragments;

import ...

public class TestActivity extends Activity {

FrameLayout container;

FragmentManager myFragmentManager;

MyFragment1 myFragment1;

MyFragment2 myFragment2;

MyFragment3 myFragment3;

final static String TAG\_1 = "FRAGMENT\_1";

final static String TAG\_2 = "FRAGMENT\_2";

final static String TAG\_3 = "FRAGMENT\_3";

final static String KEY\_MSG\_1 = "FRAGMENT1\_MSG";

final static String KEY\_MSG\_2 = "FRAGMENT2\_MSG";

final static String KEY\_MSG\_3 = "FRAGMENT3\_MSG";

// класс для первого фрагмента

public static class MyFragment1 extends Fragment {

TextView textMsg;

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

View view = inflater.inflate(R.layout.fragment1, null);

textMsg = (TextView) view.findViewById(R.id.tvMessage);

Bundle bundle = getArguments();

if (bundle != null) {

String msg = bundle.getString(KEY\_MSG\_1);

if (msg != null) {

textMsg.setText(msg);

}

}

return view;

}

public void setMsg(String msg) {

textMsg.setText(msg);

}

}

// класс для второго фрагмента

public static class MyFragment2 extends Fragment {

TextView textMsg;

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

View view = inflater.inflate(R.layout.fragment2, null);

textMsg = (TextView) view.findViewById(R.id.tvMessage);

Bundle bundle = getArguments();

if (bundle != null) {

String msg = bundle.getString(KEY\_MSG\_2);

if (msg != null) {

textMsg.setText(msg);

}

}

return view;

}

public void setMsg(String msg) {

textMsg.setText(msg);

}

}

// класс для третьего фрагмента

public static class MyFragment3 extends Fragment {

TextView textMsg;

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

View view = inflater.inflate(R.layout.fragment3, null);

textMsg = (TextView) view.findViewById(R.id.tvMessage);

Bundle bundle = getArguments();

if (bundle != null) {

String msg = bundle.getString(KEY\_MSG\_3);

if (msg != null) {

textMsg.setText(msg);

}

}

return view;

}

public void setMsg(String msg) {

textMsg.setText(msg);

}

}

// метод основной активности

/\*\* Called when the activity is first created. \*/

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_test);

container = (FrameLayout) findViewById(R.id.container);

Button button1 = (Button) findViewById(R.id.button1);

Button button2 = (Button) findViewById(R.id.button2);

Button button3 = (Button) findViewById(R.id.button3);

button1.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View arg0) {

MyFragment1 fragment = (MyFragment1) myFragmentManager

.findFragmentByTag(TAG\_1);

if (fragment == null) {

Bundle bundle = new Bundle();

bundle.putString(KEY\_MSG\_1, "Заменили на первый фрагмент");

myFragment1.setArguments(bundle);

FragmentTransaction fragmentTransaction = myFragmentManager

.beginTransaction();

fragmentTransaction.replace(R.id.container, myFragment1,

TAG\_1);

fragmentTransaction.commit();

} else {

fragment.setMsg("Первый фрагмент уже загружен");

}

}

});

button2.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View arg0) {

MyFragment2 fragment = (MyFragment2) myFragmentManager

.findFragmentByTag(TAG\_2);

if (fragment == null) {

Bundle bundle = new Bundle();

bundle.putString(KEY\_MSG\_2, "Заменили на второй фрагмент");

myFragment2.setArguments(bundle);

FragmentTransaction fragmentTransaction = myFragmentManager

.beginTransaction();

fragmentTransaction.replace(R.id.container, myFragment2,

TAG\_2);

fragmentTransaction.commit();

} else {

fragment.setMsg("Второй фрагмент уже загружен");

}

}

});

button3.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View arg0) {

MyFragment3 fragment = (MyFragment3) myFragmentManager

.findFragmentByTag(TAG\_3);

if (fragment == null) {

Bundle bundle = new Bundle();

bundle.putString(KEY\_MSG\_3, "Заменили на третий фрагмент");

myFragment3.setArguments(bundle);

FragmentTransaction fragmentTransaction = myFragmentManager

.beginTransaction();

fragmentTransaction.replace(R.id.container, myFragment3,

TAG\_3);

fragmentTransaction.commit();

} else {

fragment.setMsg("Третий фрагмент уже загружен");

}

}

});

myFragmentManager = getFragmentManager();

myFragment1 = new MyFragment1();

myFragment2 = new MyFragment2();

myFragment3 = new MyFragment3();

if (savedInstanceState == null) {

// при первом запуске программы

FragmentTransaction fragmentTransaction = myFragmentManager

.beginTransaction();

// добавляем в контейнер при помощи метода add()

fragmentTransaction.add(R.id.container, myFragment1, TAG\_1);

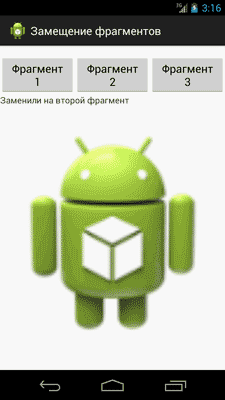
fragmentTransaction.commit();

}

}

}

При первом запуске приложения мы добавляем фрагмент в контейнер через метод **add()**. Далее при нажатии на одну из трёх кнопок мы заменяем фрагмент другим фрагментов. Если мы хотим заменить фрагмент на этот же фрагмент, то выводим сообщение, что данный фрагмент уже загружен.



# Доступ к View внутри фрагмента

[Доступ к View внутри динамического фрагмента из активности](http://developer.alexanderklimov.ru/android/theory/fragment-view.php#dynamic)  
[Передача данных между фрагментами](http://developer.alexanderklimov.ru/android/theory/fragment-view.php#getactivity)

Так как существует два способа использования фрагментов в активности, то взаимодействие между компонентами, которые находятся внутри фрагментов немного отличается.

## Доступ к View внутри динамического фрагмента из активности

Рассмотрим пример доступа к компоненту, который находится внутри динамического фрагмента из активности.

Фрагмент может иметь собственную разметку и содержать различные компоненты View: TextView, EditText и т.д. Напрямую из активности обратиться к нужному компоненту и поменять, например, текст в TextView не получится. А как же достучаться до нужного компонента? Рассмотрим простой пример.

Создадим класс фрагмента **MyFragment.java**:

package ru.alexanderklimov.fortest;

import android.app.Fragment;

import android.os.Bundle;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

public class MyFragment extends Fragment {

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

// TODO Auto-generated method stub

View myFragmentView = inflater.inflate(R.layout.fragmentlayout,

container, false);

return myFragmentView;

}

}

В методе **onCreateView()** мы указали ресурс разметки **R.layout.fragmentlayout**. Давайте создадим разметку для фрагмента.

### res/layout/fragmentlayout.xml

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:orientation="vertical" >

<TextView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Fragment:" />

<TextView

android:id="@+id/fragmenttext"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" />

</LinearLayout>

В данной разметке нас интересует вторая текстовая метка с идентификатором **fragmenttext**.

Создадим разметку для основной активности:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:orientation="horizontal" >

<LinearLayout

android:layout\_width="0px"

android:layout\_height="match\_parent"

android:layout\_weight="1"

android:orientation="vertical" >

<TextView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Activity:" />

<EditText

android:id="@+id/activitytext"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" />

<Button

android:id="@+id/sendtofragment"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Send Text to Fragment" />

</LinearLayout>

<LinearLayout

android:id="@+id/myfragment"

android:layout\_width="0px"

android:layout\_height="match\_parent"

android:layout\_weight="3" />

</LinearLayout>

Последняя компоновка **LinearLayout** с идентификатором **myfragment** является контейнером для фрагмента, который будет его замещать. Напишем код для главной активности:

public class TestActivity extends Activity {

EditText textActivity;

Button buttonSendToFragment;

MyFragment myFragment;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_test);

textActivity = (EditText) findViewById(R.id.activitytext);

buttonSendToFragment = (Button) findViewById(R.id.sendtofragment);

// получим экземпляр FragmentTransaction из нашей Activity

FragmentManager fragmentManager = getFragmentManager();

FragmentTransaction fragmentTransaction = fragmentManager

.beginTransaction();

// добавляем фрагмент

myFragment = new MyFragment();

fragmentTransaction.add(R.id.myfragment, myFragment);

fragmentTransaction.commit();

buttonSendToFragment.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View arg0) {

// TODO Auto-generated method stub

String text = textActivity.getText().toString();

TextView textFragment = (TextView) findViewById(R.id.fragmenttext);

textFragment.setText(text);

}

});

}

}

При запуске программы мы получаем экземпляр класса **FragmentTransaction** и добавляем фрагмент на экран вместо LinearLayout. Теперь текстовая метка фрагмента доступна для изменения - получаем ссылку на нужный компонент и устанавливаем требуемый текст.

## Передача данных между фрагментами

Когда мы заменяем контейнер своим фрагментом, то он становится частью активности и получаем доступ к компонентам стандартным способом, как в примере выше. Если мы используем фрагменты как самостоятельные элементы, то доступ к компонентам происходит немного по-другому. Так как фрагменты не существуют сами по себе, а только внутри активности, то сначала нужно получить доступ к родительской активности через метод **getActivity()**, а затем уже можно получить доступ к нужному компоненту из фрагмента:

TextView tvInfo = (TextView)getActivity().findViewById(R.id.textView);

Создадим разметки для двух фрагментов. В одном разместим текстовое поле, а во втором кнопку и текстовую метку, в которой будем выводить текст из текстового поля первого фрагмента:

### res/layout/redfragment.xml

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/red\_fragment"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:background="@android:color/holo\_red\_light"

android:orientation="vertical" >

<TextView

android:id="@+id/tvMessage"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Красный фрагмент" />

<EditText

android:id="@+id/editText"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:ems="10" >

<requestFocus />

</EditText>

</LinearLayout>

### res/layout/bluefragment.xml

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/blue\_fragment"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:background="@android:color/holo\_blue\_light"

android:orientation="vertical" >

<TextView

android:id="@+id/tvGetMessage"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Синий фрагмент" />

<Button

android:id="@+id/butGetText"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Прочитать данные" />

</LinearLayout>

В код первого фрагмента добавим только наполнение из разметки:

package ru.alexanderklimov.test;

import ...

public class RedFragment extends Fragment {

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

// TODO Auto-generated method stub

// Возвращаем объект View из разметки фрагмента

return inflater.inflate(R.layout.redfragment, container, false);

}

}

В класс второго фрагмента добавим код для чтения данных из первого фрагмента. Сделаем это в методе **onStart()**:

package ru.alexanderklimov.test;

import ...

public class BlueFragment extends Fragment implements OnClickListener {

private TextView tvGetMsg;

private EditText edit;

private Button butGetMsg;

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

// TODO Auto-generated method stub

// Возвращаем объект View из разметки фрагмента

return inflater.inflate(R.layout.bluefragment, container, false);

}

@Override

public void onStart() {

// TODO Auto-generated method stub

super.onStart();

tvGetMsg = (TextView)getActivity().findViewById(R.id.tvGetMessage);

edit = (EditText)getActivity().findViewById(R.id.editText);

butGetMsg = (Button)getActivity().findViewById(R.id.butGetText);

butGetMsg.setOnClickListener(this);

}

@Override

public void onClick(View v) {

// TODO Auto-generated method stub

tvGetMsg.setText(edit.getText());

}

}

Мы получаем ссылки на все компоненты и обрабатываем щелчок мыши. В разметку активности добавьте два созданных фрагмента и запустите пример. Введите какой-нибудь текст в текстовом поле (напоминаю, что он относится к первому фрагменту). Нажмите на кнопку, которая относится ко второму фрагменту. В текстовой метке второго фрагмента появится введённым вами текст.

Fragment (Фрагменты). Часть шестая



Удержание состояния фрагментов

В предыдущей статье мы рассмотрели, как сохранять данные при изменении конфигурации и других операциях.

Но есть другая проблема - нам нужно сохранить не просто данные, а работающий объект. К примеру, наш объект воспроизводит музыку или видео. Мы можем сохранить название песни, время проигрывания и т.д, но при повороте музыка прервётся, так как объект будет уничтожен вместе с фрагментом. Нужен другой подход.

У фрагмента есть свойство **retainInstance**, которое по умолчанию содержит значение **false**. Это означает, что при поворотах фрагмент не сохраняется, а уничтожается и создается заново вместе с активностью-хостом. Вызов **setRetainInstance(true)** сохраняет фрагмент, который не уничтожается вместе с активностью, а передается новой активности в неизменном виде.

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setRetainInstance(true);

}

Представление фрагмента может уничтожаться и создаваться заново без необходимости уничтожать сам фрагмент. При изменении конфигурации (повороте) **FragmentManager** сначала уничтожает макет фрагмента в своем списке. Макеты фрагментов всегда уничтожаются и создаются заново по тем же причинам, по которым уничтожаются и создаются заново разметки активности: в новой конфигурации могут потребоваться новые ресурсы. Затем **FragmentManager** проверяет свойство **retainInstance** каждого фрагмента. Если оно равно **false** (по умолчанию), **FragmentManager** уничтожает экземпляр фрагмента. Фрагмент и его макет будут созданы заново новым экземпляром **FragmentManager** новой активности.

Если значение **retainInstance** равно *true*, макет фрагмента уничтожается, но сам фрагмент остаётся. При создании новой активности новый экземпляр **FragmentManager** находит сохранённый фрагмент и воссоздаёт его макет.

Сохранённый фрагмент не уничтожается, а отсоединяется (detached) от «умирающей» активности. В сохранённом состоянии фрагмент всё ещё существует, но не имеет активности-хоста.

Переход в сохранённое состояние происходит только при выполнении двух условий:

* для фрагмента был вызван метод **setRetainInstance(true)**
* активность-хост уничтожается для изменения конфигурации (обычно поворот). Фрагмент находится в сохранённом состоянии очень недолго — от момента отсоединения от старой активности до повторного присоединения к новой, немедленно создаваемой активности.

Сохранённые фрагменты продолжают существовать только при уничтожении активности при изменения конфигурации. Если активность уничтожается из-за того, что системе потребовалось освободить память, то все сохранённые фрагменты также будут уничтожены.

# Анимация фрагментов

Android позволяет переключаться между фрагментами с помощью анимационных эффектов. Рассмотрим примеры с применением готовых эффектов, а также создадим свою собственную анимацию.

Создадим новый проект с главной активностью. В активность динамически будут подключаться фрагменты, между которыми будем переключаться с анимационными эффектами при помощи кнопки. Компонент **FrameLayout**послужит контейнером для подключаемых фрагментов.

Разметка для активности:

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".MainActivity" >

<FrameLayout

android:id="@+id/fragment"

android:layout\_width="wrap\_content"

android:layout\_height="match\_parent" />

<Button

android:id="@+id/button"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_toRightOf="@id/fragment"

android:text="Переключиться" />

</RelativeLayout>

Создадим разметки для фрагментов.

### fragment1.xml

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:background="#FF0000">

<TextView

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:text="Fragment 1" />

</LinearLayout>

### fragment2.xml

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:background="#00FF00">

<TextView

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:text="Fragment 2" />

</LinearLayout>

Отличия между фрагментами - фоновый цвет (красный и зелёный) и текст, по которым будет проще различать фрагменты.

Фрагмент должен иметь свой класс. Для примера достаточно простейшего кода, где нужно указать разметку фрагмента.

### Fragment1.java

package ru.alexanderklimov.fragmentsanimation;

import android.app.Fragment;

import android.os.Bundle;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

public class Fragment1 extends Fragment {

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

return inflater.inflate(R.layout.fragment1, null);

}

}

По такому же принципу создайте класс для второго фрагмента самостоятельно.

Теперь перейдём к коду для главной активности:

package ru.alexanderklimov.fragmentsanimation;

import android.os.Bundle;

import android.app.Activity;

import android.app.Fragment;

import android.app.FragmentTransaction;

import android.view.Menu;

import android.view.View;

public class MainActivity extends Activity {

private Fragment fragment1;

private Fragment fragment2;

private FragmentTransaction transaction;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

fragment1 = new Fragment1();

fragment2 = new Fragment2();

transaction = getFragmentManager().beginTransaction();

// transaction.setCustomAnimations(R.animator.slide\_in\_left, R.animator.slide\_in\_right);

transaction.replace(R.id.fragment, fragment1);

transaction.addToBackStack(null);

transaction.commit();

}

public void onClick(View v){

transaction = getFragmentManager().beginTransaction();

// transaction.setTransition(FragmentTransaction.TRANSIT\_FRAGMENT\_OPEN);

// transaction.setCustomAnimations(R.animator.slide\_in\_left, R.animator.slide\_in\_right);

if(fragment1.isVisible()){

transaction.replace(R.id.fragment, fragment2);

}else{

transaction.replace(R.id.fragment, fragment1);

}

transaction.commit();

}

}

В коде я закомментировал две строчки для анимации. В таком виде переключение между фрагментами происходит стандартным способом без анимации. Теперь раскомментируйте первую строчку **transaction.setTransition(FragmentTransaction.TRANSIT\_FRAGMENT\_OPEN);**. В классе **FragmentTransaction** есть несколько готовых анимаций. С помощью метода **setTransition(int transit)** мы можем указать нужную анимацию и увидеть её в действии.

Список анимаций

* **TRANSIT\_FRAGMENT\_CLOSE**
* **TRANSIT\_FRAGMENT\_OPEN**
* **TRANSIT\_FRAGMENT\_FADE**
* **TRANSIT\_NONE**

Снова закомментируйте строку и раскомментируйте вторую строку **transaction.setCustomAnimations(R.animator.slide\_in\_left, R.animator.slide\_in\_right);**. Метод **setCustomAnimations()** позволяет указать собственную анимацию. Методу передаются два параметра. Первый параметр описывает анимацию для фрагмента, который появляется, а второй — описывает анимацию для фрагмента, который убирается с экрана устройства. Метод следует вызывать до появления фрагментов, иначе анимация не будет применена.

С методом **setCustomAnimations()** нужно быть осторожным при работе с фрагментами из support-библиотеки. В одной из версий библиотеки разработчики из Гугла всё поломали и код перестал работать. Неизвестно когда починят. Поэтому используйте стандартные фрагменты.

Для анимации нужно создать XML-файлы в папке **res/animator** (её тоже нужно создать вручную).

### slide\_in\_left.xml

<?xml version="1.0" encoding="utf-8"?>

<set xmlns:android="http://schemas.android.com/apk/res/android" >

<objectAnimator

xmlns:android="http://schemas.android.com/apk/res/android"

android:duration="1500"

android:interpolator="@android:anim/accelerate\_decelerate\_interpolator"

android:propertyName="y"

android:valueFrom="-1280"

android:valueTo="0"

android:valueType="floatType" />

</set>

### slide\_in\_right.xml

<?xml version="1.0" encoding="utf-8"?>

<set xmlns:android="http://schemas.android.com/apk/res/android"

android:ordering="together">

<objectAnimator

android:interpolator="@android:anim/accelerate\_interpolator"

android:propertyName="alpha"

android:valueType="floatType"

android:valueTo="0"

android:duration="300"/>

<objectAnimator xmlns:android="http://schemas.android.com/apk/res/android"

android:interpolator="@android:anim/accelerate\_decelerate\_interpolator"

android:propertyName="x"

android:valueType="floatType"

android:valueTo="1280"

android:valueFrom="0"

android:duration="1500"/>

</set>

Элементы визуальных эффектов задаются в теге **objectAnimator**. У атрибута **propertyName** указывается свойство фрагмента, которое мы будем изменять при анимации, **valueType** указывает тип изменяемого параметра. Атрибуты **valueFrom** и **valueTo** указывают диапазон изменения параметра, указанного в **propertyName**. Если параметр **valueFrom** не указан, то значение берётся равное текущему. В нашем случае **valueFrom** равен -1280, это означает, что движение фрагмента по оси **y** будет начинаться со значения -1280 и перемещение будет происходить пока значение **y** не станет равным 0 для верхнего левого угла нашего фрагмента в течении 1500 миллисекунд (атрибут **duration**).

Тег **set** служит для объединения эффектов либо их разделения. В файле **slide\_in\_right.xml** используется атрибут **ordering** со значением **together**, что означает проигрывать эффекты одновременно, в противовес ему существует значение **sequentially**, которое требует последовательного отображения эффектов в анимации.

## Переворачиваем карту

Напишем ещё один пример для закрепления материала.

Найдите в интернете две картинки, например, изображения карты и её рубашки. Подготовим четыре файла в папке **res/animator**.

### card\_flip\_left\_enter.xml

<?xml version="1.0" encoding="utf-8"?>

<set xmlns:android="http://schemas.android.com/apk/res/android">

<objectAnimator

android:valueFrom="1.0"

android:valueTo="0.0"

android:propertyName="alpha"

android:duration="0"/>

<objectAnimator

android:valueFrom="-180"

android:valueTo="0"

android:propertyName="rotationY"

android:interpolator="@android:interpolator/accelerate\_decelerate"

android:duration="@integer/card\_flip\_duration\_full"/>

<objectAnimator

android:valueFrom="0.0"

android:valueTo="1.0"

android:propertyName="alpha"

android:startOffset="@integer/card\_flip\_duration\_half"

android:duration="1"/>

</set>

### card\_flip\_left\_exit.xml

<?xml version="1.0" encoding="utf-8"?>

<set xmlns:android="http://schemas.android.com/apk/res/android">

<objectAnimator

android:duration="@integer/card\_flip\_duration\_full"

android:interpolator="@android:interpolator/accelerate\_decelerate"

android:propertyName="rotationY"

android:valueFrom="0"

android:valueTo="180"/>

<objectAnimator

android:duration="1"

android:propertyName="alpha"

android:startOffset="@integer/card\_flip\_duration\_half"

android:valueFrom="1.0"

android:valueTo="0.0"/>

</set>

### card\_flip\_right\_enter.xml

<?xml version="1.0" encoding="utf-8"?>

<set xmlns:android="http://schemas.android.com/apk/res/android">

<objectAnimator

android:duration="0"

android:propertyName="alpha"

android:valueFrom="1.0"

android:valueTo="0.0"/>

<objectAnimator

android:duration="@integer/card\_flip\_duration\_full"

android:interpolator="@android:interpolator/accelerate\_decelerate"

android:propertyName="rotationY"

android:valueFrom="180"

android:valueTo="0"/>

<objectAnimator

android:duration="1"

android:propertyName="alpha"

android:startOffset="@integer/card\_flip\_duration\_half"

android:valueFrom="0.0"

android:valueTo="1.0"/>

</set>

### card\_flip\_right\_exit.xml

<?xml version="1.0" encoding="utf-8"?>

<set xmlns:android="http://schemas.android.com/apk/res/android">

<objectAnimator

android:duration="@integer/card\_flip\_duration\_full"

android:interpolator="@android:interpolator/accelerate\_decelerate"

android:propertyName="rotationY"

android:valueFrom="0"

android:valueTo="-180"/>

<objectAnimator

android:duration="1"

android:propertyName="alpha"

android:startOffset="@integer/card\_flip\_duration\_half"

android:valueFrom="1.0"

android:valueTo="0.0"/>

</set>

В файле **res/values/strings.xml** добавьте пару новых ресурсов.

<integer name="card\_flip\_duration\_full">1000</integer>

<integer name="card\_flip\_duration\_half">500</integer>

Теперь создадим два макета для двух фрагментов в папке **res/layout**. Фрагменты будут содержать по одной картинке.

### fragment\_card\_front.xml

<?xml version="1.0" encoding="utf-8"?>

<ImageView xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:scaleType="centerCrop"

android:src="@drawable/card\_front"/>

### fragment\_card\_back.xml

<?xml version="1.0" encoding="utf-8"?>

<ImageView xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:scaleType="centerCrop"

android:src="@drawable/card\_back"/>

Создаём два класса для фрагментов, которые загружают свои макеты.

package ru.alexanderklimov.as21;

import android.app.Fragment;

import android.os.Bundle;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

public class CardFrontFragment extends Fragment {

@Override

public View onCreateView(LayoutInflater inflater,

ViewGroup container, Bundle savedInstanceState) {

return inflater.inflate(

R.layout.fragment\_card\_front, container, false);

}

}

package ru.alexanderklimov.as21;

import android.app.Fragment;

import android.os.Bundle;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

public class CardBackFragment extends Fragment {

@Override

public View onCreateView(LayoutInflater inflater,

ViewGroup container, Bundle savedInstanceState) {

return inflater.inflate(

R.layout.fragment\_card\_back, container, false);

}

}

Упростим макет активности.

<?xml version="1.0" encoding="utf-8"?>

<FrameLayout android:id="@+id/container"

xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"/>

Осталось написать код. Ещё раз напомню, что используйте стандартные фрагменты, а не из библиотеки совместимости.

package ru.alexanderklimov.as21;

import android.os.Bundle;

import android.support.v7.app.AppCompatActivity;

import android.view.View;

import android.widget.FrameLayout;

public class MainActivity extends AppCompatActivity {

private boolean mShowingBack = false;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

FrameLayout frameLayout = (FrameLayout) findViewById(R.id.container);

assert frameLayout != null;

frameLayout.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

flipCard();

}

});

if (savedInstanceState == null) {

getFragmentManager()

.beginTransaction()

.add(R.id.container, new CardFrontFragment())

.commit();

}

}

private void flipCard() {

if (mShowingBack) {

mShowingBack = false;

getFragmentManager().popBackStack();

} else {

mShowingBack = true;

getFragmentManager()

.beginTransaction()

.setCustomAnimations(

R.animator.card\_flip\_right\_enter,

R.animator.card\_flip\_right\_exit,

R.animator.card\_flip\_left\_enter,

R.animator.card\_flip\_left\_exit)

.replace(R.id.container, new

CardBackFragment())

.addToBackStack(null)

.commit();

}

}

}

Контейнер обрабатывает касание пальца как щелчок и переворачивает фрагмент на другой.

# Создание контекстного меню внутри фрагмента

Создание контекстного меню внутри фрагмента ничем не отличается от создания контекстного меню в активности. Попробуем?

Создадим разметку для фрагмента. Достаточно будет текстовой метки и кнопки.

### fragment\_1.xml

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical"

android:padding="8dp" >

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:text="Нажмите и удерживайте лапу на кнопке"

android:textAppearance="?android:attr/textAppearanceMedium" />

<Button

android:id="@+id/button\_long\_press"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:text="Нажми меня" >

</Button>

</LinearLayout>

Класс для фрагмента. В нём активируем разметку и зарегистрируем контекстное меню для кнопки. Меню создадим программно.

### MyFragment.java

package ru.alexanderklimov.test;

import android.app.Fragment;

import android.os.Bundle;

import android.view.ContextMenu;

import android.view.ContextMenu.ContextMenuInfo;

import android.view.LayoutInflater;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.view.View.OnClickListener;

import android.view.ViewGroup;

import android.widget.Button;

import android.widget.Toast;

public class MyFragment extends Fragment {

public static final int IDM\_A = 101;

public static final int IDM\_B = 102;

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

View root = inflater.inflate(R.layout.fragment\_1, container, false);

Button button = (Button)root.findViewById(R.id.button\_long\_press);

button.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View v) {

// TODO Auto-generated method stub

Toast.makeText(getActivity(), "Мяу!", Toast.LENGTH\_LONG)

.show();

}

});

registerForContextMenu(button);

return root;

}

@Override

public void onCreateContextMenu(ContextMenu menu, View v,

ContextMenuInfo menuInfo) {

super.onCreateContextMenu(menu, v, menuInfo);

menu.add(Menu.NONE, IDM\_A, Menu.NONE, "Menu A");

menu.add(Menu.NONE, IDM\_B, Menu.NONE, "Menu B");

}

@Override

public boolean onContextItemSelected(MenuItem item) {

switch (item.getItemId()) {

case IDM\_A:

Toast.makeText(getActivity(), "Выбран пункт А", Toast.LENGTH\_LONG)

.show();

return true;

case IDM\_B:

Toast.makeText(getActivity(), "Выбран пункт B", Toast.LENGTH\_LONG)

.show();

return true;

}

return super.onContextItemSelected(item);

}

}

Осталось подключить фрагмент к активности. В разметке активности разместите FrameLayout в качестве контейнера для фрагмента с идентификатором **@+id/containter**.

### MainActivity.java

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

MyFragment myfragment = new MyFragment();

getFragmentManager().beginTransaction().add(R.id.container, myfragment).commit();

}

[DesarrolloAntonio/FragmentTransactionExtended](https://github.com/DesarrolloAntonio/FragmentTransactionExtended) - библиотека анимаций между фрагментами.