Работа с фрагментами

Передача параметров

Для чего?

Иногда нам нужно передать во фрагмент какое-то значение или целый объект. Например, у нас есть список дел и по нажатию на элемент списка должен открываться фрагмент с подробным описанием дела.

Как использовать?

Есть множество разных способов передать данные во фрагмент, но мы с вами рассмотрим как это можно сделать с помощью метода **newInstance()**, т.к. он является рекомендуемым способом.

Допустим, мы хотим передать из активности в наш фрагмент текст, который мы будем отображать. Для этого нам нужно в метод **newInstance()** добавить входных аргументов тех данных, которые мы хотим передать. Внутри метода мы оборачиваем аргументы в бандл и вызываем метод fragment.setArguments().
Соответственно, после создания фрагмента, например, в onCreateView() можем вызвать getArguments() и извлечь записанный аргумент, что и продемонстрировано на листинге ниже.

```
public class ExampleFragment extends Fragment {

  public static final String ARG_TEXT = "TEXT_FOR_TEXT_VIEW_KEY";

  private TextView mExampleText;

  public static ExampleFragment newInstance(String textForTextView) {

    Bundle args = new Bundle();
    args.putString(ARG_TEXT, textForTextView);
    ExampleFragment fragment = new ExampleFragment();
    fragment.setArguments(args);
    return fragment;
  }

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fr_example, container, false);
    mExampleText = view.findViewById(R.id.tv_example);
```

```
        Bundle args = getArguments();
        String text = args.getString(ARG_TEXT);
        mExampleText.setText(text);
        return view;
    }
}
```

Динамическая смена

Для чего?

Фрагменты - очень мощный механизм, с помощью которого можно создавать гибкие пользовательские интерфейсы. Например, есть приложение с множеством различных экранов, выполненных через фрагменты, которые можно переключать. При этом хост активити не меняется.

Как использовать?

Например у нас есть активити с тремя кнопками, по которым можно переходить на новый экран.

```
private Button mButton1;
private Button mButton2;
private Button mButton3;

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

    mButton1 = findViewById(R.id.btn_1);
    mButton2 = findViewById(R.id.btn_2);
    mButton3 = findViewById(R.id.btn_3);
}
```

Теперь разберемся с фрагментами.
Создаем класс SampleFragment:

```
public class SampleFragment extends Fragment {
```

```
    private static final String ARG_NAME = "arg_name";
    private static final String ARG_COLOR = "arg_color";

    private String mName;
    private int mColor;
    private TextView mTextView;

    public static SampleFragment newInstance(String name, int color) {
        Bundle args = new Bundle();
        args.putString(ARG_NAME, name);
        args.putInt(ARG_COLOR, color);
        SampleFragment fragment = new SampleFragment();
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Bundle args = getArguments();
        if (args != null) {
            mName = args.getString(ARG_NAME);
            mColor = args.getInt(ARG_COLOR);
        }
    }

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container,
@Nullable Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fr_sample, container, false);
    }

    @Override
    public void onViewCreated(View view, @Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        mTextView = view.findViewById(R.id.tv_sample);
        mTextView.setText(mName);
        view.setBackgroundColor(mColor);
    }

}
```

В разметке fr_sample просто добавим TextView и укажем через атрибут gravity, чтобы

он располагался по середине экрана.

Чтобы различать фрагменты, в метод newInstance будем передавать строку текста для нашей TextView и цвет, которым будем красить корневой лейаут фрагмента.
В методе onCreate() достаем наши значения, а в методе onViewCreated() применяем.
Подготовка фрагмента завершена, теперь займемся вызывающей стороной, то есть активити.

Чтобы реагировать на нажатие кнопок, активити будет реализовывать интерфейс onClickListener.
В методе onClick через **switch** будем передавать в наши фрагменты цвета, в которые они будут окрашены после нажатия.

```
@Override
public void onClick(View view) {
  switch (view.getId()) {
     case R.id.btn_1:
        switchFragment("one", R.color.red);
        break;
     case R.id.btn_2:
        switchFragment("two", R.color.green);
        break;
     case R.id.btn_3:
        switchFragment("three", R.color.blue);
        break;
  }
}
```

Вся логика смены фрагментов происходит в методе switchFragment()

```
void switchFragment(String name, int color) {
  Fragment fragment;
  fragment = getSupportFragmentManager().findFragmentByTag(name);
  if (fragment == null) {
     fragment = SampleFragment.newInstance(name, ContextCompat.getColor(this, color));
     getSupportFragmentManager().beginTransaction()
           .replace(R.id.container, fragment, name)
           .commit();
  } else {
     Toast.makeText(this, "Не меняем", Toast.LENGTH_SHORT).show();
  }
}
```

Как уже рассказывалось ранее, мы используем фрагмент менеджер чтобы добавлять фрагменты в контейнер. Под контейнером мы понимаем layout в разметке, в котором будет храниться фрагмент. Мы указываем таг, с помощью которого проверяем, какой

фрагмент находится в контейнере на данный момент, и если он такой же, который мы хотим добавить, то нет смысла менять фрагмент, оставляем как есть. В противном случае мы создаем новый фрагмент с переданными параметрами и заменяем наш фрагмент. В тосте говорится, что новый фрагмент аналогичен старому.

Вот мы и рассмотрели как динамически менять фрагменты. Технически, в этом ничего сложного нет, и вся работа происходит в несколько строк.

Диалоги

Alert Dialog

Для чего?

Диалог - это окно, чаще всего не занимающее весь экран, в котором пользователю предлагается выполнить то или иное действие, к примеру, ввести логин-пароль, подтвердить какое либо действие, либо выбрать какой либо из предложенных вариантов.

Как использовать?

Давайте создадим какой нибудь диалог, в котором будет заголовок, текст и пара кнопок. Для этого создадим новое активити и добавим на него кнопку. При нажатии на кнопку и будет всплывать диалог.

Диалоги создавать напрямую нельзя, для этого используется билдер.
Билдер - это паттерн проектирования, в котором для целевого класса, который имеет много параметров, создается статический внутренний класс, методы которого и собирают целевой класс, а последний метод, чаще всего build() или create() возвращает готовый настроенный целевой класс.
Делается это для того чтобы не плодить кучу конструкторов с разными аргументами и их комбинациями.
Указываем название, сообщение, и две кнопки, позитивную и негативную, в каждую передаем текст и листенер с каким либо действием.
В конце вызываем метод show(), который создает и сразу же показывает нам диалог.

Метод showInfo() вызывается при нажатии на кнопку.

```
private void showInfo() {
  AlertDialog.Builder builder = new AlertDialog.Builder(this);
  builder.setTitle("Info")
        .setMessage("London is the capital of Great Britain")
        .setPositiveButton("true", new DialogInterface.OnClickListener() {
```

```
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                Toast.makeText(MainActivity.this, "You are absolutely right",
Toast.LENGTH_SHORT).show();
            }
        })
        .setNegativeButton("false", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                Toast.makeText(MainActivity.this, "Wrong", Toast.LENGTH_SHORT).show();
            }
        })
        .show();
}
```

DialogFragment

Для чего?

DialogFragment - это идейный наследник диалогов, но его главное преимущество в том, что можно в нем можно использовать кастомный дизайн.
DialogFragment - это фрагмент и одновременно диалог.
На практике DialogFrament применяется гораздо чаще и позволяет полностью отказаться от использования AlertDialog.

Как использовать?

Создадим новый класс, и укажем в качестве родителя - DialogFragment.
В нем вместо переопределения onCreateView переопределяем onCreateDialog()

```
 @NonNull
  @Override
  public Dialog onCreateDialog(Bundle savedInstanceState) {
     AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

     View inflate = getActivity().getLayoutInflater().inflate(R.layout.di_info, null);
     mEditText = inflate.findViewById(R.id.et_info);

     builder.setTitle("Info")
            .setView(inflate)
            .setPositiveButton("true", null)
            .setNegativeButton("false", null);
     return builder.create();
  }
```

Обратите внимание на строки
```
View inflate = getActivity().getLayoutInflater().inflate(R.layout.di_info, null);
mEditText = inflate.findViewById(R.id.et_info);
```
и
```
.setView(inflate)
```

В них мы создаем кастомную разметку для диалога с EditText'ом и добавляем ее в билдер.

Обычной практикой является использование статического метода show() для показа DialogFragment вместо newInstance(). В show() передают FragmentManager и, если необходимо, дополнительные параметры, так же как и в метод newInstance()

```
 public static final String TAG = InfoDialogFragment.class.getSimpleName();

   public static void show(FragmentManager fragmentManager) {
      InfoDialogFragment infoDialogFragment = new InfoDialogFragment();
      infoDialogFragment.show(fragmentManager, TAG);
   }
```

Метод вызова в активити будет выглядеть так:
```
void showEditDialog() {
   InfoDialogFragment.show(getSupportFragmentManager());
}
```

Его можно навесить на кнопку, например.

Теперь поговорим о том, как в активити получить результат работы фрагмента, на примере нашего DialogFragment.

Получение результатов работы с диалогом

Для чего?

Очень часто возникает необходимость передать результат работы диалога в активити для обработки. Например, пользователь ввел свои данные или выбрал один из предложенных вариантов. Необходимо сделать так, чтобы данные стали известны не только диалогу, но и активити, который запускал этот диалог.

Как использовать?

Колбек интерфейс - это самый распространенный способ передачи данных из фрагмента в активити, и фрагмент не обязательно должен быть диалогом. Но обычные фрагменты часто сами обрабатывают свою логику, а диалог - передает результат в активити.
Внутри класса диалога определяем интерфейс с нужным нам методом.

```
public interface DialogCallback {
    void setPositiveResult(String result);
}
```

Объявляем переменную этого типа

```
private DialogCallback mCallback;
```

В методе onAttach() проверяем и инициализируем

```
@Override
public void onAttach(Context context) {
    super.onAttach(context);
    if (context instanceof DialogCallback) {
        mCallback = (DialogCallback) context;
    }
}
```

Здесь мы проверяем, что контекст, по факту активити, реализует наш интерфейс. В противном случае можно например кинуть эксепшн.

В активити, само собой, реализуем наш интерфейс

```
@Override
public void setPositiveResult(String result) {
    Toast.makeText(this, result, Toast.LENGTH_SHORT).show();
}
```

Хранить ссылку на активити - не самая лучшая идея, поэтому не забываем также очищать ее в методе onDetach()

```
@Override
public void onDetach() {
    super.onDetach();
    mCallback = null;
}
```

Добавляем код, вызывающий метод колбека на позитивную кнопку:

```
// в onCreateDialog()
// ----
 .setPositiveButton("true", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialogInterface, int i) {
                    if (mEditText.getText().length() > 0) {
                        mCallback.setPositiveResult(mEditText.getText().toString());
                    }
                }
            })
// ---
```

Запускаем приложение, нажимаем на кнопку и запускаем диалог, вводим текст, нажимаем на позитивную кнопку и убеждаемся, что все работает.