3.8. Навигация в Android приложениях

Сайт: Samsung Innovation Campus
Курс: Мобильная разработка на Kotlin
Книга: 3.8. Навигация в Android приложениях

Напечатано:: Murad Rezvan
Дата: понедельник, 3 июня 2024, 17:50

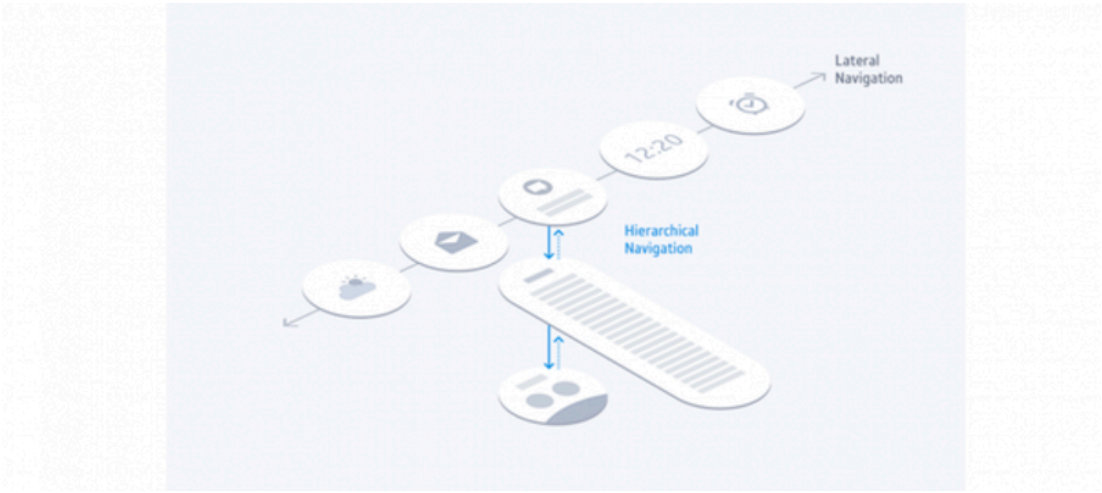
Оглавление

3.8.1. Виды навигации в Android

Современные принципы разработки интуитивно понятного мобильного интерфейса, фактически закрепленные в рекомендациях по материальному дизайну, также не обошли вниманием и вопросы навигации . Навигация - это процесс замены текущего интерфейса приложения на экране пользователя на другой в процессе взаимодействия пользователя с *[UI]: User Interface . Иначе говоря это переключение layout-ов в соответствии с действиями пользователя и логики работы программы. Например, при нажатии кнопки "помощь", у пользователя открывается активность помощи, которая занимает весь экран мобильного устройства. Или нажатии кнопки "настройки", в меню приложения - открываются настройки приложения, а при нажатии кнопки "назад" перед пользователем открывается активность, которая был перед переходом на текущую.

Есть разные классификации процессов навигации. В зависимости от точки зрения выделить следующие группы .

С точки зрения пользователя



Навигация между различными экранами и приложениями является основной частью взаимодействия с пользователя с системой. Соответственно пользователь может выполнять следующие управляющие действия :

- Иерархическая навигация. Зачастую это навигация "вниз-вверх". Т.е. из текущего экрана пользователь переходит на экран какой то подзадачи, подчиненной текущей. Нажав кнопку вверх, пользователь осуществляет передвижение к более верхнему в иерархии интерфейса окну. Такую иерархическую навигацию довольно сложно осуществить, поэтому часто иерархическая навигация заменяется навигацией по стеку.
- Боковая навигация используется для перемещения пользователя по экранам находящимся на одном уровне. Т.е. пользователь переходит влево или в право, перебирая экраны один за другим. Обычно переключение производится путем смахивания экрана влево или в право, либо прокручивания панели закладок (Bezel), либо выбора закладки на панели закладок. При этом на панели видно, какую часть информации просматривается в данный момент. Такой метод представления информации рекомендуется либо для представления структурированной одноуровневой информации, которой слишком много чтобы представить в одном окне, но легко можно разделить на последовательный блоки. Также используют для представления master-detail информации.
- Навигация по стеку возврата. При этом виде навигации, при нажатии кнопки назад пользователь последовательно переходит на экраны интерфейса, которые он открывал раньше, в порядке обратном хронологическому.
- Глубокие ссылки - это возможность переходить из экрана одного приложения прямо в конкретно указанный экран другого приложения.

С точки зрения пользователя нет никакой разницы, какими именно способами обеспечивается навигация. Ему важно чтобы это было интуитивно понятно, и поэтому важно, чтобы элементы поддерживали принципы Material Design, а также чтобы структура переходов была унифицирована. Например:

- кнопка "домой" - должна всегда приводить на один и тот же начальный экран,
- кнопка "назад" - делает строго обратный по хронологии (back stack) переход по экранам, вне зависимости от их иерархии и при достижении начала цепочки может позволять выход из приложения,
- кнопка "вверх", должна поднимать пользователя по иерархии, но не должна делать выход из приложения, когда достигнут верхний уровень,
- свайп используется либо для перелистывания Bezel (панель прокрутки вкладок) либо для перелистывания master-detail окон,
- использовать нестандартные управляющие элементы нежелательно, как например двойной или тройной клик.

С точки зрения разработчика, по цели навигации.

- переход на новый фрагмент,
- переход на новую активность,
- переход на другое приложение.

По элементам, обеспечивающих навигацию.

- кликабельные UI элементы (Button, ImageButton, пункты списков и т.д.),
- навигационные UI элементы (меню активности, боковая панель Drawer, TopAppBar, ToolBar, BottomNavigation),
- свайпы,

- управление стеком возврата.

3.8.2. Navigation API

На ежегодной конференции разработчиков, проводимой Google в Маунтин-Вью, штат Калифорния, в 2018г был представлен Android Jetpack. Android Jetpack - это следующее поколение компонентов Android, которые при сохранении обратной совместимости, позволяет быстро и легко создавать надежные, высококачественные приложения. В его состав в том числе вошли новое Navigation API а также существенно модернизированные стандартные навигационные UI компоненты.

В настоящее время можно пользоваться как старыми программными конструкциями для организации навигации так и возможностями нового API навигации. Если о классических приемах было сказано ранее в этом модуле то в этой главе рассмотрим использование именно нового API. Его использование дает множество преимуществ:

- создание единого графического представление всего графа переходов,
- контроль переходов в приложении на соответствии построенной схеме,
- унификация и упрощения перехода к новой активности или фрагменту, а также передаче им данных,
- модифицированные навигационные элементы UI теперь интегрированы с новым механизмом навигации, вследствие чего нужно писать значительно меньше кода при их реализации,
- упрощение описания deep link,
- засчет упрощения программирования переходов разрабатывать сложное приложение станет легче и быстрее, особенно это заметно в случае если нужно быстро переделать схему переходов.

Простой пример использования navigation API.

Рассматриваемые далее примеры входят в практическую работу, исходный код которой можно скачать по ссылке

Рассмотрим пример простого приложений, использующего Navigation API. Этот же пример разрабатывается далее в практической работе. Далее рассмотрим этапы и особенности его создания.

1. Для использования navigation API в проекте, в начале, необходимо добавить в **build.gradle(app)** следующие зависимости:

```
    def nav_version = "2.3.0-alpha05"
    // For Java language implementation
    //  implementation "androidx.navigation:navigation-fragment:$nav_version"
    //  implementation "androidx.navigation:navigation-ui:$nav_version"
    // For Kotlin language implementation
    implementation "androidx.navigation:navigation-fragment-ktx:$nav_version"
    implementation "androidx.navigation:navigation-ui-ktx:$nav_version"
    // Dynamic Feature Module Support
    implementation "androidx.navigation:navigation-dynamic-features-fragment:$nav_version"
```

2. В макет первой активности, например (**activity_main.xml**) необходимо добавить контейнер **NavHostFragment**. Именно в нем под управление **NavController** будут устанавливаться желаемые фрагменты. В простом случае, это может быть единственным тегом в макете.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <fragment
        android:id="@+id/nav_host_fragment"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:defaultNavHost="true"
        app:navGraph="@navigation/nav_graph" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

При этом сам код активности может быть очень минималистичен. Например таким:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

Open in Playground → Target: JVM Running on v.2.0.0

3. Необходимо создать нужное количество фрагментов, которые будут переключаться внутри **nav_host_fragment** в соответствии с правилами навигации. Например первый фрагмент

```
class BlankFragment : Fragment() {
// ...
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        var view:View=inflater.inflate(R.layout.fragment1, container, false)
        // ============= Button обработчик, способ записи 1
        var btn:Button=view.findViewById(R.id.button)
        btn.setOnClickListener(object : View.OnClickListener{
            override fun onClick(p0: View?) {
                if (p0 != null) {
                    p0.findNavController().navigate(R.id.action_blankFragment_to_blankFragment2)
                }
            }
        })
        // ============= Button3 обработчик, способ записи 2
        var btn3:Button=view.findViewById(R.id.button3)
        btn3.setOnClickListener(View.OnClickListener { view -> view.findNavController().navigate(R.id.action_blankFragment_to_blankFragment3) })
        // ============= Button5 обработчик, способ записи 3
        view.findViewById<Button>
(R.id.button5).setOnClickListener(Navigation.createNavigateOnClickListener(R.id.action_blankFragment_to_blankFragment4))
        return view
    }
// ...
}
```

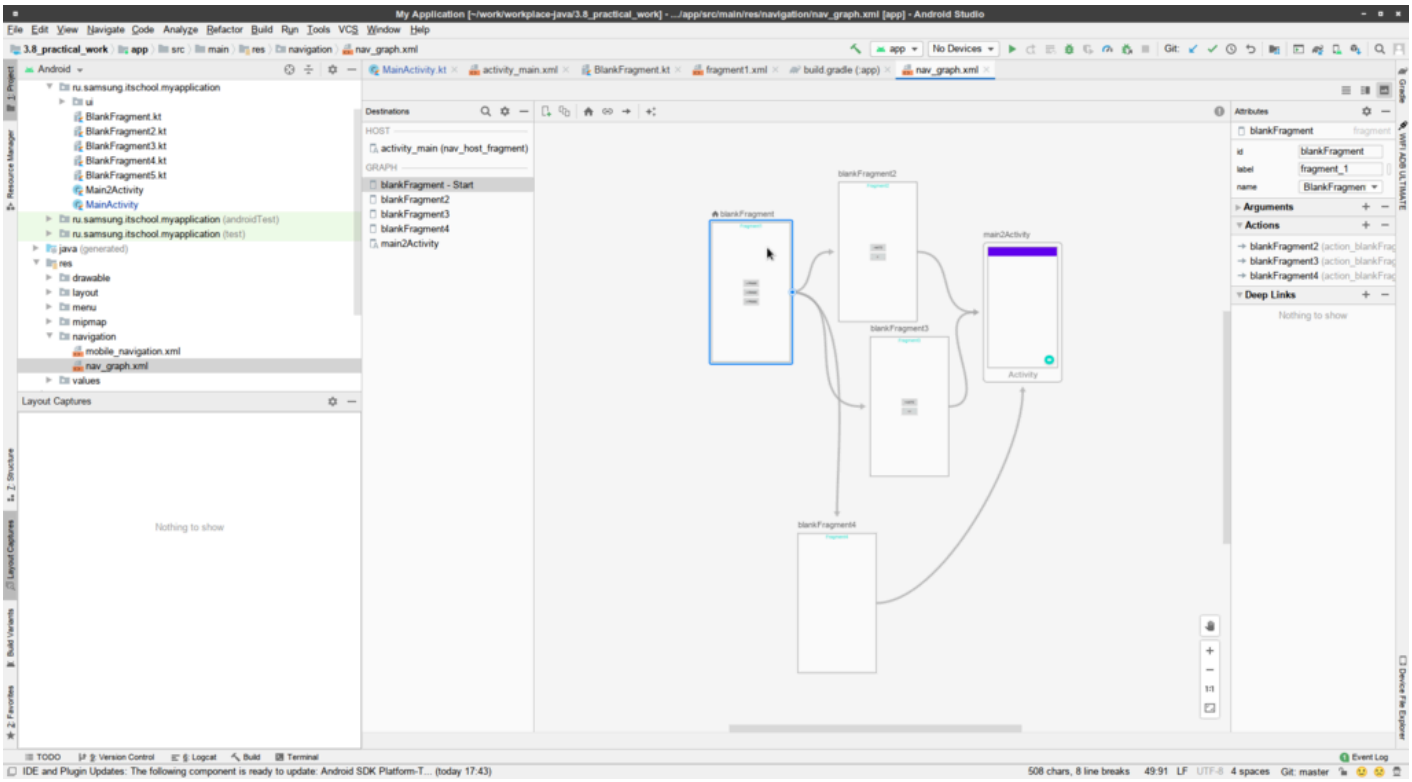
Open in Playground → Target: JVM Running on v.2.0.0

И его layout:

```
<FrameLayout 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=".BlankFragment">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:text="Fragment1"
        android:textColor="@color/colorAccent"
        android:textSize="24dp" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:orientation="vertical">

        <Button
            android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="-> frag2" />

        <Button
            android:id="@+id/button3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="-> frag3" />

        <Button
            android:id="@+id/button5"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="-> frag4" />

    </LinearLayout>
</FrameLayout>
```

4. И самая зрелищная часть. Для ранее описанного **nav_host_fragment** (см пп 1) нужно создать указанный в атрибуте **app:navGraph** файл графа навигации. Для этого нужно создать в **/res** паку **navigation**, а в ней создать файл навигации - *New -> Navigation Resource File* с именем

nav_graph.xml. После создания откроется пустой граф. Кнопкой "+" нужно добавить все фрагменты или активности, на которые будут осуществляться переходы, т.е. эти фрагменты или активности будут последовательно устанавливаться на экран устройства, по мере действий пользователя. Первый добавленный фрагмент будет отмечен как стартовый - то есть именно он будет автоматически установлен NavigationContrloller-ом в контейнер nav_host_fragment в макете main-activity.xml. При необходимости, впоследствии, при помощи кнопки "домик" можно сделать стартовым другой фрагмент (но только фрагмент!).



На этом этапе система навигации полностью готова. Теперь можно на элементах управления UI, в рассматриваемом примере кнопок, подключать обработчики нажатий с заданным переходом. Сам обработчик стандартный - OnClickListener. Однако сам процесс перехода теперь можно делать по новому. Если раньше требовалось писать перехода на другую активность что то вроде:

```
var intent: Intent = Intent( getContext(), Main2Activity::class.java)
startActivity(intent)
```

Open in Playground → Target: JVM Running on v.2.0.0

или для фрагмента:

```
val manager: FragmentManager? = fragmentManager
val transaction = manager?.beginTransaction();
if (transaction != null) {
  transaction.add(R.id.blankFragment2, BlankFragment2());
  transaction.commit()
};
```

Open in Playground → Target: JVM Running on v.2.0.0

то теперь, а независимости от того, что является целью фрагмент или активность:

```
p0.findNavController().navigate(R.id.action_blankFragment_to_blankFragment2)
```

Open in Playground → Target: JVM Running on v.2.0.0

Контролер сам определит цель перехода и предпримет соответствующие действия. Кроме того, в цели перехода вы можете указать как ID фрагмента/активности, так и ID связи из графа навигации. Более того, в JetPack-е добавлена возможность создания объекта OnClickListener-а при помощи createNavigateOnClickListener, более упрощенным образом:

```
view.findViewById<Button>(R.id.button5).setOnClickListener(Navigation.createNavigateOnClickListener(R.id.action_blankFragment_to_blankFragment4))
```

Open in Playground → Target: JVM Running on v.2.0.0

Обратите внимание.

Уточним устройство и принцип работы навигации:

1. Сначала стартует активность.
2. Внутри запущенной активности есть контейнер NavHostFragment с которым связан Navigation Controller и в котором указан файл с графом навигации.
3. Когда активность стартовала, Navigation Controller автоматически делает переход к узлу навигации, отмеченному как стартовый.
4. Указанный узел устанавливается на экран пользователя.

5. Если установленный узел является активностью, то внутри него может быть другой контейнер NavHostFragment со своим фалом навигации, и тогда текущий граф навигации заканчивается, и повторится сценарий с п.1 для нового графа навигации. Если же в нем контейнера навигации нет, то на этой активности заканчивается действие Navigation API. В любом случае действие текущего графа навигации заканчивается переходом на другую активность.
6. Если же установленный узел является фрагментом, то с него возможно передвижение дальше по графу навигации (метод navigate(ID) в коде), при условии, что разработчик указал на графе соответствующие дуги. Т.о. далее действия повторяются с п.4.

Распространенные ошибки.

Несмотря на то что Navigation API делает разработку быстрой и удобной, но у не опытного разработчика могут возникать некоторые типовые ошибки, которые иногда трудно обнаружить.

- NavHostFragment включен, есть файл навигации, но в нем не указан ни один фрагмент. Приложение будет компилироваться но при запуске будет сразу закрываться. Ошибка в LogCat -

```
 java.lang.RuntimeException: Unable to start activity
ComponentInfo{ru.samsung.itschool.myapplication/ru.samsung.itschool.myapplication.MainActivity}: android.view.InflateException: Binary
XML file line #18 in ru.samsung.itschool.myapplication:layout/activity_main: Binary XML file line #...
....
     Caused by: android.view.InflateException: Binary XML file line #18 in ru.samsung.itschool.myapplication:layout/activity_main: Binary
XML file line #18 in ru.samsung.itschool.myapplication:layout/activity_main: Error inflating class fragment
     Caused by: android.view.InflateException: Binary XML file line #18 in ru.samsung.itschool.myapplication:layout/activity_main: Error
inflating class fragment
     Caused by: java.lang.IllegalStateException: no start destination defined via app:startDestination for
ru.samsung.itschool.myapplication:id/nav_graph.xml
....
```

- NavHostFragment включен, есть файл навигации, и в нем в качестве стартового элемента указана сам же активность. Приложение будет компилироваться и запускаться но при запуске будет непрерывно и бесконечно открываться сама активность. Это приложение будет даже сложно закрыть. При этом ни LogCat ни debug ничего не дадут, так как приложение сделано технически корректно, но по смыслу запуск стартовой активности зациклен внутри Navigation API.
- Навигация в приложении сделана полностью корректно, но в коде в методе navigate() указывается ID action (дуги графа), не совпадающего со схемой навигации. Приложение будет компилироваться и запускаться, но при выполнении именно этого участка кода, приложение будет закрываться, а в LogCat будет ошибка -

```
 java.lang.IllegalArgumentException: navigation destination ru.samsung.itschool.myapplication:id/action_blankFragment2_to_blankFragment3
is unknown to this NavController
```

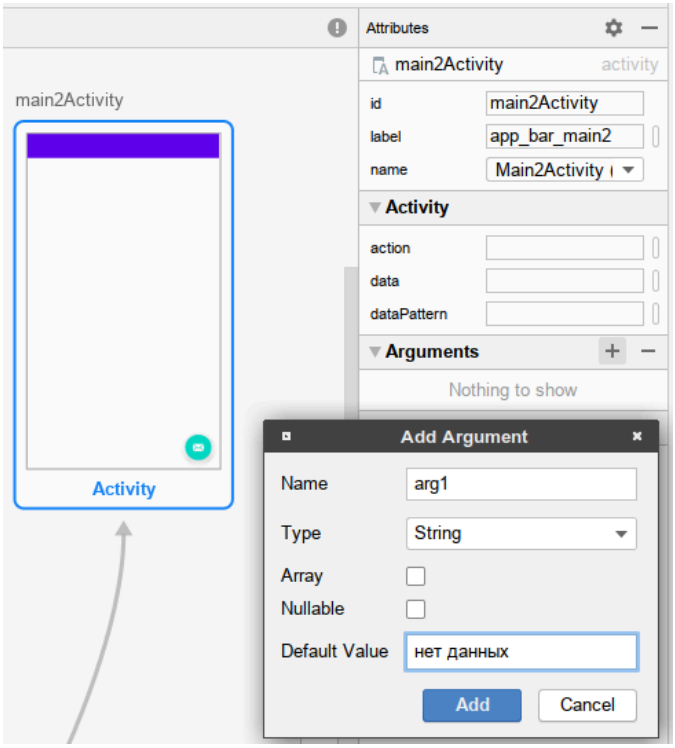
3.8.3. Дополнительные возможности навигации.

Передача параметров

До появления Navigation API, параметры при навигации передавались следующими способами:

- вносились в интент, при помощи метода **putExtra()** интента при запуске новой активности,
- бандл с данными вносился в фрагмент при помощи метода **setArguments()** при запуске фрагмента.

Этими же способами передачи данных можно пользоваться и сейчас, но JetPack предоставил унифицированный способ передачи параметров при переходе на любое окно UI , или иначе говоря на при переходе на новый узел графа навигации. Помимо единообразия кода передачи данных в UI пункта назначения, Navigation API, также контролирует количество и типы передаваемых параметров, а также может присваивать значения по умолчанию, в случае отсутствия передачи. Для использования нового способа передачи данных, нужно сначала на целевом узле графа навигации добавить аргументы:



После этого, в коде вызывающей стороны нужно добавить

```
btn6.setOnClickListener(View.OnClickListener { view ->
            // создаем бандл
            val bundle:Bundle=Bundle()
            // вносим в него значения
            bundle.putString("arg1",et.text.toString())
            // выполняем навигацию с передачей данных
            view.findNavController().navigate(R.id.action_blankFragment4_to_main2Activity,bundle)
        })
```

Open in Playground → Target: JVM Running on v.2.0.0

При этом уже не важно, является ли цель фрагментом или активностью, важно лишь, чтобы передаваемые аргументы были внесены в поле **arguments** узла графа навигации. На стороне получателя извлекаем данные, если это активность:

```
val str=intent.getStringExtra("arg1")
```

Open in Playground → Target: JVM Running on v.2.0.0

или, если фрагмент:

```
val str=arguments.getString("arg1")
```

Open in Playground → Target: JVM Running on v.2.0.0

С обратной передачей данных все несколько сложнее. До появления NavigationAPI стандартным способом было вызов функции **startActivityForResult()** с извлечением данных в методе **onActivityResult()** вызывающей активности. Начиная с версии 2.3.0 Navigation API можно передавать данные через **NavBackStackEntry**, который доступен через встроенный стек возвратов NavController-а и который представляет собой обертку над **Bundle**.

К примеру из **BlankFargment4** перейдем в **BlankFargment3**, а потом обратно с возвратом данных. Т.о. в **BlankFargment3.kt** Для возврата данных используется **previousBackStackEntry**

```
findNavController().previousBackStackEntry?.savedStateHandle?.set("result_from_activity", "ответные данные")
```

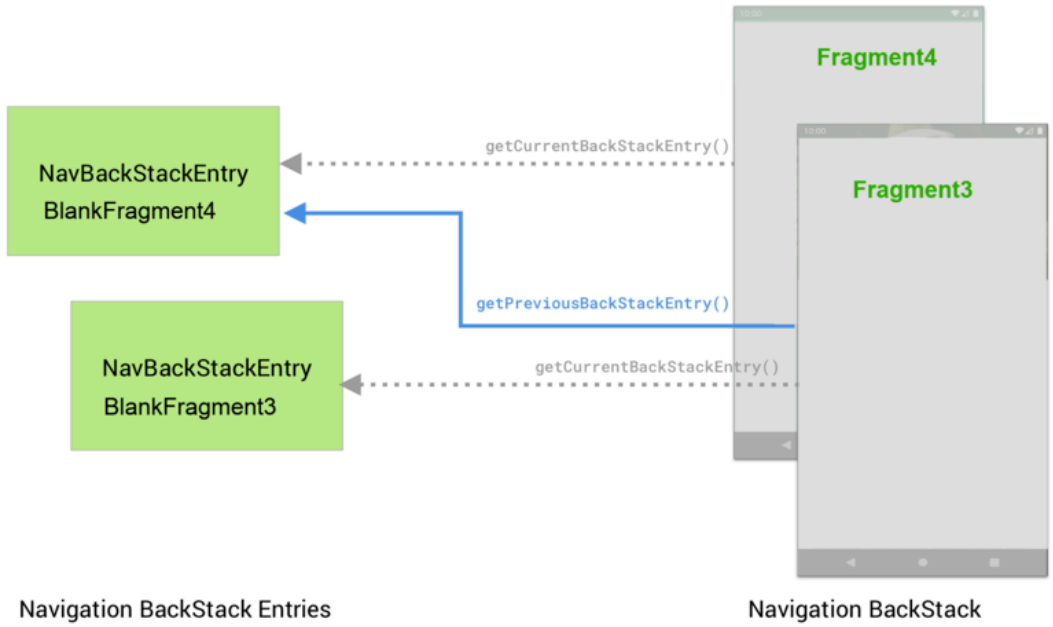
Open in Playground → Target: JVM Running on v.2.0.0

При возврате обратно в BlankFargment4.kt эти данные извлекаются из currentBackStackEntry соответсвенным обсервером:

```
val liveData = findNavController(this).currentBackStackEntry?.getSavedStateHandle()
            ?.getLiveData("result_from_activity")
        if (liveData != null) {
            liveData.observe(viewLifecycleOwner, object : Observer<String?> {
                override fun onChanged(s: String?) {
                    returnData = s
                }
            })
        }
```

Open in Playground → Target: JVM Running on v.2.0.0

Это процесс можно проиллюстрировать следующим образом:



К сожалению, подобным образом нельзя вернуть данные из активности во фрагмент, т.к. новая активность породит новый экземпляр NavigationController, со свои (с другим) BackStack. Все ньюансы передачи данных можно посмотреть в документации. Также при использовании плагина Safe Args, возможна поддержка TypeSafe передачи данных.

> Как правило, желательно передавать только минимальный объем данных между пунктами назначения. Например, нужно передать ключ для извлечения объекта, а не сам объект, так как общее пространство для всех сохраненных состояний ограничено в Android.

GlobalAction и DeepLink

DeepLink (глубокие ссылки) это альтернативный способ вызов приложения. При этом в приложении определенным образом регистрируется URL, при попытки показа которого в любом приложении на устройстве пользователя будет открыто приложение прямо на указанной активности или фрагменте.

Например онлайн-продавец пиццы на странице сайта с выбором пиццы для заказа указывает ссылку на URL заказа. Если у пользователя не установлено приложение по заказу пиццы от этого продавца - то у него браузер просто переходит по ссылке. Если же приложение установлено - то вместо веб страницы открывается андроид приложение, прямо на фрагменте с онлайн-заказом выбранного товара.

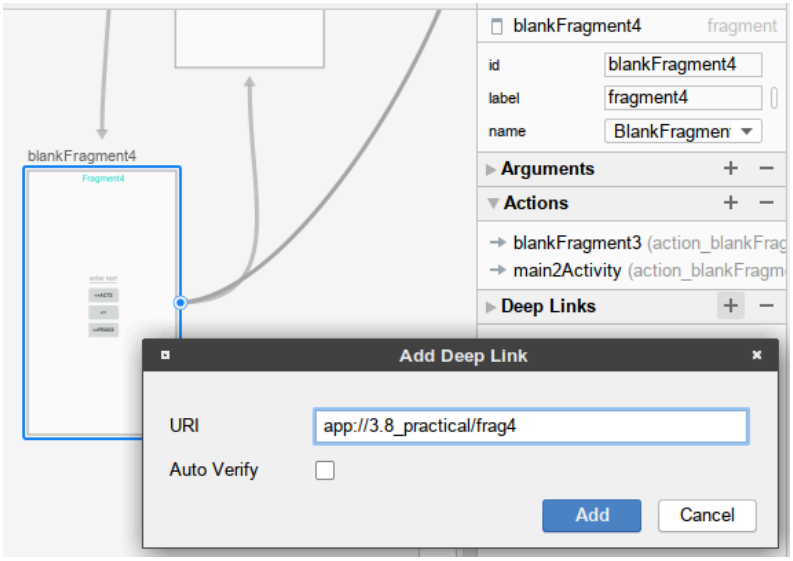
Появление NavigationAPI упростило процесс создания глубоких ссылок. Рассмотрим пример реализации. Для добавления глубоких ссылок на конкретную целевой UI необходимо сделать две вещи:

1. Для AndroidStudio версии выше 3.2, нужно добавить в манифест, в раздел активности добавить:

```
<nav-graph android:value="@navigation/nav_graph" />
```

   Для более ранней версии нужно вручную настроить Intent Filter , в соответствии с инструкцией.

2. выбрать в графе навигации целевой узел и в его свойствах добавить DeepLink:



Для примера добавим две следующие ссылки.

- app://3.8_practical/frag4
- https://myitacademy.ru/edu/login/index.php#reg

Если приложение из рассматриваемого примера установлено на мобильном устройстве, то при при просмотре в браузере этой главы учебника, при клике на вышеуказанные ссылки будет открыто рассматриваемое приложение - пример. Если же приложение не установлено, то при клике на второй ссылке будет переход на страницу регистрации.

Кроме того можно в задавать ссылки с передачей данных. Например при задании DeepLink в виде app://3.8_practical/frag4/{arg1}, реальный URL может выглядеть так: app://3.8_practical/frag4/100, а в коде можно извлечь:

```
val str=arguments.getString("arg1")
```

Open in Playground → Target: JVM Running on v.2.0.0

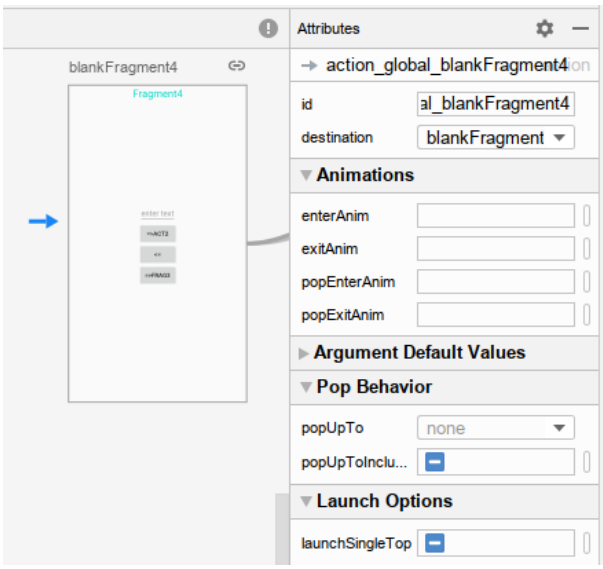
или

```
val str=intent.getStringExtra("arg1")
```

Open in Playground → Target: JVM Running on v.2.0.0

Дополнительную информацию и глубоких ссылках, можно посмотреть здесь.

GlobalAction - это по сути упрощение вида графа навигации. К примеру, если на цель слишком много ссылок, можно вместо них задать один GlobalAction и его ID указывать во всех нужных navigation(). Для этого нужно нажать ПКМ на узле графа и выбрать *Add Action->Global* . В окне свойств GlobalAction уможно увидеть ID, который можно использовать впоследствии:



Переход "назад" и "вверх"

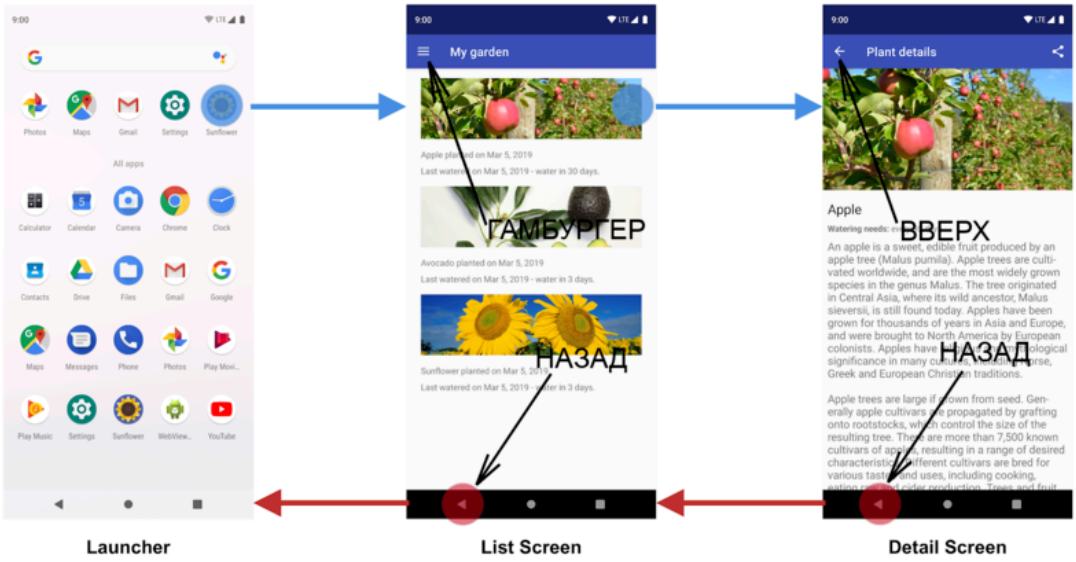
По умолчанию, в Android модулем исполнения является активность. Поэтому, в стеке возвратов, хранятся предыдущие, по отношению к текущей активности. Поскольку фрагмент встраивается в активность, то аппаратная кнопка back (возврат) переключает на экран предыдущую активность, в независимости от того какие переходы были между фрагментами.

В NavigationAPI напротив реализация стека возвратов в равной мере учитывает как переключение активностей, так и переключение фрагментов. Поэтому программный вызов функции **popBackStack()** NavigationAPI вызовет предыдущего посещенного пользователем узла графа навигации (или проще говоря экрана), также как и функции **navigateUp()** нажатие кнопки ⊲(Up) на TopAppBar. Если приложение реализуется с использованием NavigatioAPI, то можно изменить поведение по умолчанию аппаратной кнопки back. Для этого достаточно добавить атрибут **app:defaultNavHost** в описании фрагмента NavigationController-а в лайоуте активности следующим образом:

```
app:defaultNavHost="true"
```

Тогда, поведение аппаратной кнопки back будет соответствовать логике NavigationAPI, т.е. учитывать переходы по фрагментам, а не только по активностям как раньше.

Помимо навигации "назад", в NavigationAPI предусмотрена навигация "вверх". Если разработчик не меняет её поведение по умолчанию при помощи, например, указания **android:parentActivityName** или другими способами, то навигация "вверх" совпадает с навигацией назад за одним исключением - при достижении начала графа навигации кнопка вверх превращается в "гамбургер", т.е. по кнопке "вверх" нельзя выйти из приложения. А кнопка "назад" продолжает действовать и далее, т.е. выйти из приложения.



Начать тур для пользователя на этой странице