Navigation 用法详解

`Navigation`主要运用于管理`fragment`之间的跳转，通过可视化的操作，使得开发者能够更好的操作`fragment`之间的跳转的关系，`navigation`将`fragment`好处在于：

1. 处理`fragment`之间的转场动画
2. 在一系列有序的`fragment`，让它们的顺序变得更加可靠'
3. 将一系列有序的`fragment`，产生相同的堆栈。

Navigation 使用配置

1. 项目构建配置

```
// Java language implementation
implementation "androidx.navigation:navigation-fragment:2.3.0"
implementation "androidx.navigation:navigation-ui:2.3.0"

// Kotlin
implementation "androidx.navigation:navigation-fragment-ktx:2.3.0"
implementation "androidx.navigation:navigation-ui-ktx:2.3.0"

// Feature module Support
implementation "androidx.navigation:navigation-dynamic-features-
fragment:2.3.0"

// Testing Navigation
androidTestImplementation "androidx.navigation:navigation-testing:2.3.0"
```

2. 相关配置的解释

关键攻略	解析
NavHostFragment	jetpack 实现好的导航fragment 其内部以封装好 navigation 跳转处理逻辑
navGraph	代表需要引用哪一个 navigation 文件
navigation 文件夹	负责存放 navigation配置文件
startDestination	最开始的展示的 fragment 通过配置文件内的id进行索引
action	表示当前fragment 一些操作动作 例如 转场动画、下一个需要展示的 fragment

`Navigation`最为关键的一步在于，在`Android`项目工程在`navigation`文件目录中创建相应的`navigation`文件，来设置相应一系列有序的`fragment`相互间的关系。

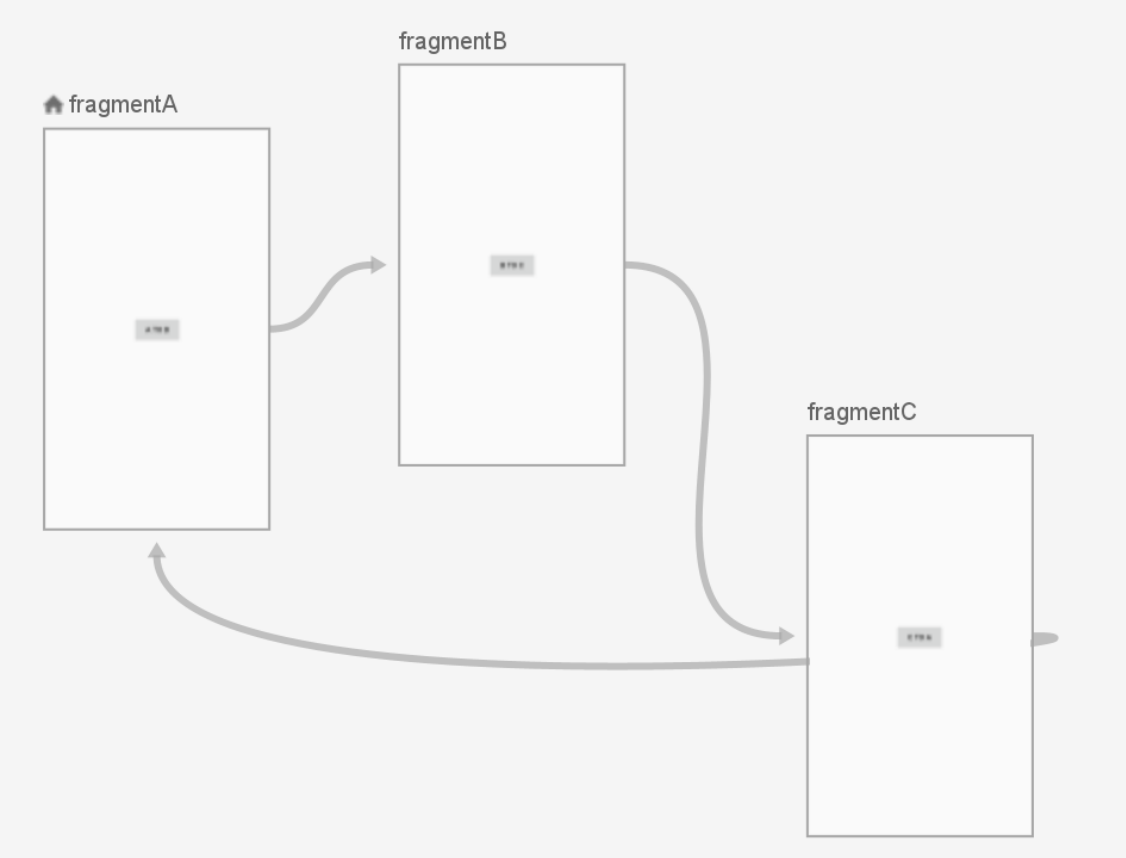
Navigation 基本使用

在上一节中我们简单的了解Navigation的基本配置，以下内容将配合上一节的概念简单的去使用navigation。

1. 以下以`lib_navigation_demo_nav`示例代码进行演示如何配置好一个简单的导航文件

```
<?xml version="1.0" encoding="utf-8"?>
<navigation 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:id="@+id/lib_navigation_demo_nav"
 app:startDestination="@id/fragmentA"
 tools:ignore="UnusedNavigation">

 <fragment
     android:id="@+id/fragmentA"
     android:name="com.killeTom.navigation.fragment.NavDemoFragmentA"
     android:label="fragmentA"
     tools:layout="@layout/nav_demo_fragment_a">

     <action
         android:id="@+id/action_fragmentA_to_fragmentB"
         app:destination="@id/fragmentB"
         app:exitAnim="@android:anim/slide_out_right" />
 </fragment>

 <fragment
     android:id="@+id/fragmentB"
     android:name="com.killeTom.navigation.fragment.NavDemoFragmentB"
     android:label="fragmentB"
     tools:layout="@layout/nav_demo_fragment_b">

     <action
         android:id="@+id/action_fragmentB_to_fragmentC"
         app:destination="@id/fragmentC"
         app:exitAnim="@android:anim/slide_out_right" />
 </fragment>

 <fragment
     android:id="@+id/fragmentC"
     android:name="com.killeTom.navigation.fragment.NavDemoFragmentC"
     android:label="fragmentC"
     tools:layout="@layout/nav_demo_fragment_c">
     <action
         android:id="@+id/action_fragmentC_to_fragmentA"
         app:destination="@id/fragmentA"
         app:exitAnim="@android:anim/slide_out_right" />
 </fragment>
</navigation>
```

其视图预览如下图：



2. 在`Activity`对应的xml文件引用`navigation`

```
<?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=".NavigationDemoActivity">

    <fragment
        android:id="@+id/nav_demo"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:navGraph="@navigation/lib_navigation_demo_nav"
        app:defaultNavHost="true"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

通过这样的引用就实现出 A -> B -> C -> A 这样一个fragment切换显示的顺序。

3. 利用navigation对fragment进行跳转

下面将以NavDemoFragmentA跳转到NavDemoFragmentB为示例

```
class NavDemoFragmentA : Fragment() {

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val view = inflater.inflate(R.layout.nav_demo_fragment_a, container,
false)

        view.nav_action.setOnClickListener {
            //通过利用findNavController获取导航控制器，然后指向配置文件中的actionId
进行控制跳转
            findNavController().navigate(R.id.action_fragmentA_to_fragmentB)
        }

        return view
    }


}
```

4. 利用navigation对fragment进行值传递跳转

   navigation值传递分为`Bundle`、`safeargs`两种以下针对这两种方式分别讲解

   1. 利用`Bundle`进行值传递

      Bundle传值在于构建出一个Bundle对象，在里面存放数据，然后通过调用navigate时，将Bundle传入

      譬如：NavDemoFragmentA跳转到NavDemoFragmentB时将一个 hello 字符串传递

      ```
      class NavDemoFragmentA : Fragment() {

          override fun onCreateView(
              inflater: LayoutInflater,
              container: ViewGroup?,
              savedInstanceState: Bundle?
          ): View? {
              val view = inflater.inflate(R.layout.nav_demo_fragment_a,
      container, false)

              view.nav_action.setOnClickListener {

                  var bundle: Bundle = bundleOf("value" to "hello")

                  findNavController()
                  .navigate(R.id.action_fragmentA_to_fragmentB,bundle)

              }

              return view
          }

      }
      ```

```
class NavDemoFragmentB :Fragment() {

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val view =  inflater.inflate(R.layout.nav_demo_fragment_b,
container, false)

        //取值
        var message = arguments?.getString("value")?:
view.nav_action.text

        view.nav_action.text = message

        return view
    }
}
```

2. 利用 `safeargs` 进行类型安全值传递

   利用 `safeargs` 首先得在gralde文件中配置:

```
buildscript {
    repositories {
        google()
    }
    dependencies {
        def nav_version = "2.3.0"
        classpath "androidx.navigation:navigation-safe-args-gradle-
plugin:$nav_version"
    }
}
```

然后在需要使用 `navigation` 的模块的gradle文件中添加如下配置

注意当项目完全为Java时请使用这一配置

```
apply plugin: "androidx.navigation.safeargs"
```

如果支持kotlin时请使用这一配置

```
apply plugin: "androidx.navigation.safeargs.kotlin"
```

到此 `safeargs` 环境配置则已经完成紧接着我们以 NavDemoFragmentC 到 NavDemoFragmentA 为例子演示应该如何使用 `safeargs` 进行安全的值传递,

首先将为 `lib_navigation_demo_nav` C 到 A 的配置代码修改为

```
<?xml version="1.0" encoding="utf-8"?>
<navigation 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:id="@+id/lib_navigation_demo_nav"
    app:startDestination="@id/fragmentA"
    tools:ignore="UnusedNavigation">

   <--!-->......</--!-->

     <--!-->......</--!-->

    <fragment
        android:id="@+id/fragmentC"
        android:name="com.killeTom.navigation.fragment.NavDemoFragmentC"
        android:label="fragmentC"
        tools:layout="@layout/nav_demo_fragment_c">

        <action
            android:id="@+id/action_fragmentC_to_fragmentA"
            app:destination="@id/fragmentA"
            app:exitAnim="@android:anim/slide_out_right" />

        <argument android:name="message"
            android:defaultValue=""
            app:argType="string"/>

        <argument android:name="result"
            app:argType="boolean"
            android:defaultValue="false"/>


    </fragment>
</navigation>
```

然后查看下是否能够动态生成相应的Args代码，譬如这里示例代码对应为`NavDemoFragmentCArgs`文件。注意如果没有及时动态生成，重新 build 构建当前模块尝试生成即可。

```
data class NavDemoFragmentCArgs(
  val message: String = "",
  val result: Boolean = false
) : NavArgs {
  fun toBundle(): Bundle {
    val result = Bundle()
    result.putString("message", this.message)
    result.putBoolean("result", this.result)
    return result
  }

  companion object {
    @JvmStatic
    fun fromBundle(bundle: Bundle): NavDemoFragmentCArgs {

 bundle.setClassLoader(NavDemoFragmentCArgs::class.java.classLoader)
      val __message : String?
      if (bundle.containsKey("message")) {
        __message = bundle.getString("message")
        if (__message == null) {
          throw IllegalArgumentException("Argument \"message\" is marked
as non-null but was passed a null value.")
```

```
        }
      } else {
        __message = ""
      }
      val __result : Boolean
      if (bundle.containsKey("result")) {
        __result = bundle.getBoolean("result")
      } else {
        __result = false
      }
      return NavDemoFragmentCArgs(__message, __result)
    }
  }
}
```

当对应的文件生成后，我们可以这样使用达到一个值类型安全传递的效果：

```
class NavDemoFragmentC :Fragment() {

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        //省略部分不相干代码

        //构建NavArgs
        val args = NavDemoFragmentCArgs("ok",true)

        //利用args 获取结果bundle 并作为值传递过去
            NavHostFragment.findNavController(this)

.navigate(R.id.action_fragmentC_to_fragmentA,args.toBundle())

        //省略部分不相干代码
    }

}

class NavDemoFragmentA : Fragment() {


    override fun onResume() {
        super.onResume()

        resultAction()
    }

    //取值
    private fun resultAction(){
        val bundle = arguments?:return

        val args = NavDemoFragmentCArgs.fromBundle(bundle)

        Log.i(this::class.java.simpleName,args.toString())

    }
```

```
}
```

5. 常见的一些错误信息

   针对一些场景操作导致的常见错误，以下将会进行一些讲解分析。

   1. 当前Activity静态配置，但是启动当前Activity 出现了 `android.view.InflateException` 可能存在以下几个因素
      - navigation配置文件没有设置`startDestination`,因为静态配置必须配置`startDestination`属性否则将无法感知到默认显示的`fragment`是哪一个
      - 当前Activity对应的布局中并没有使用`name`属性导致无法正常引用布局文件
   2. 静态配置中无法正常导航出现了`does not have a NavController`相关提示错误信息出现场景的错误代码示例如下:

```
   <?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=".NavigationDemoActivity">

       <fragment
           android:id="@+id/nav_demo"
           android:name="com.killeTom.navigation.fragment.NavDemoFragmentA"
           android:layout_width="0dp"
           android:layout_height="0dp"
           app:navGraph="@navigation/lib_navigation_demo_nav"
           app:defaultNavHost="true"
           app:layout_constraintStart_toStartOf="parent"
           app:layout_constraintEnd_toEndOf="parent"
           app:layout_constraintTop_toTopOf="parent"
           app:layout_constraintBottom_toBottomOf="parent"/>

   </androidx.constraintlayout.widget.ConstraintLayout>
```

   针对上诉代码，第一次启动往往是能够正常启动的，并且能够显示出首个fragment但是首个fragment可能与配置文件中的默认显示不一样，具体的第一次显示fragment取`name`属性对应的fragment。

   为什么会出现`does not have a NavController`错误提示呢？原因简单在于`name`属性对应的fragment并没有实现对应的`NavController`逻辑，导致无法找到这个对象，无法实现出导航逻辑。

   因此没有特殊需求，其实我们直接将`name`属性对应`androidx.navigation.fragment.NavHostFragment`

Navigation 原理解析

针对前一节的基本使用，以下将会讲解一些对应的原理分析：

`NavHostFragment`基于`fragment`以及`NavHost`接口的实现。

1. 为什么静态配置能够启动默认fragment?

   在`NavHostFragment`部分源码中，通过这样的一个方式启动默认的fragment

```
  @CallSuper
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        final Context context = requireContext();

        mNavController = new NavHostController(context);


        Bundle navState = null;
        //通过判断设置相关的导航配置文件
        if (mGraphId != 0) {
            // Set from onInflate()
            mNavController.setGraph(mGraphId);
        } else {
            // See if it was set by NavHostFragment.create()
            final Bundle args = getArguments();
            final int graphId = args != null ? args.getInt(KEY_GRAPH_ID) :
0;
            //当设别到存在默认的fragment的时候通过navController去显示默认的fragment
            final Bundle startDestinationArgs = args != null
                    ? args.getBundle(KEY_START_DESTINATION_ARGS)
                    : null;
            if (graphId != 0) {
                mNavController.setGraph(graphId, startDestinationArgs);
            }
        }
    }
```

2. NavController的作用?

   NavController负责管理App的Navigation，其内部实现了对fragment的堆栈以及生命周期的感知管理，并且监听手机后退键的事件触发等一系列操作。

   例如：

   - 通过`LifecycleOwner`以及`LifecycleObserver`实现生命周期的感知以及观察
   - 通过`OnBackPressedCallback`分发后退键触发事件
   - `NavigatorProvider`代理管理导航操作

总结

navigation 静态配置的使用时需要注意，name属性应该指向哪个`fragment`、以及navigation配置文件中`startDestination`的是否漏写。
理解为什么`NavHostFragment`为什么能够达到导航功能，因为其内部持有了`NavController`以及`NavController`实现了一些对生命周期感知以及监测一些事件等操作。

navigation可以通过`Bundle`、`safeargs`这两种方式的传值。其中`safeargs`方式还需要对项目进行额外的配置。