android

Laurent Provot <laurent.provot@uca.fr>

Février 2023

Android Jetpack : Navigation

Navigation Components

- Avec Jetpack : voir l'activité comme point d'entrée de l'application
- Les autres écrans sont gérés par des fragments
- On navigue grâce à des transactions entre les fragments
- `FragmentManager` OK, mais verbeux et il faut gérer la backstack
- Introduction du composant *Navigation*
- Gestion graphique sous Android Studio ($>$ v3.3)

Mise en place au sein du projet

- Dans le `build.gradle` du module de l'application

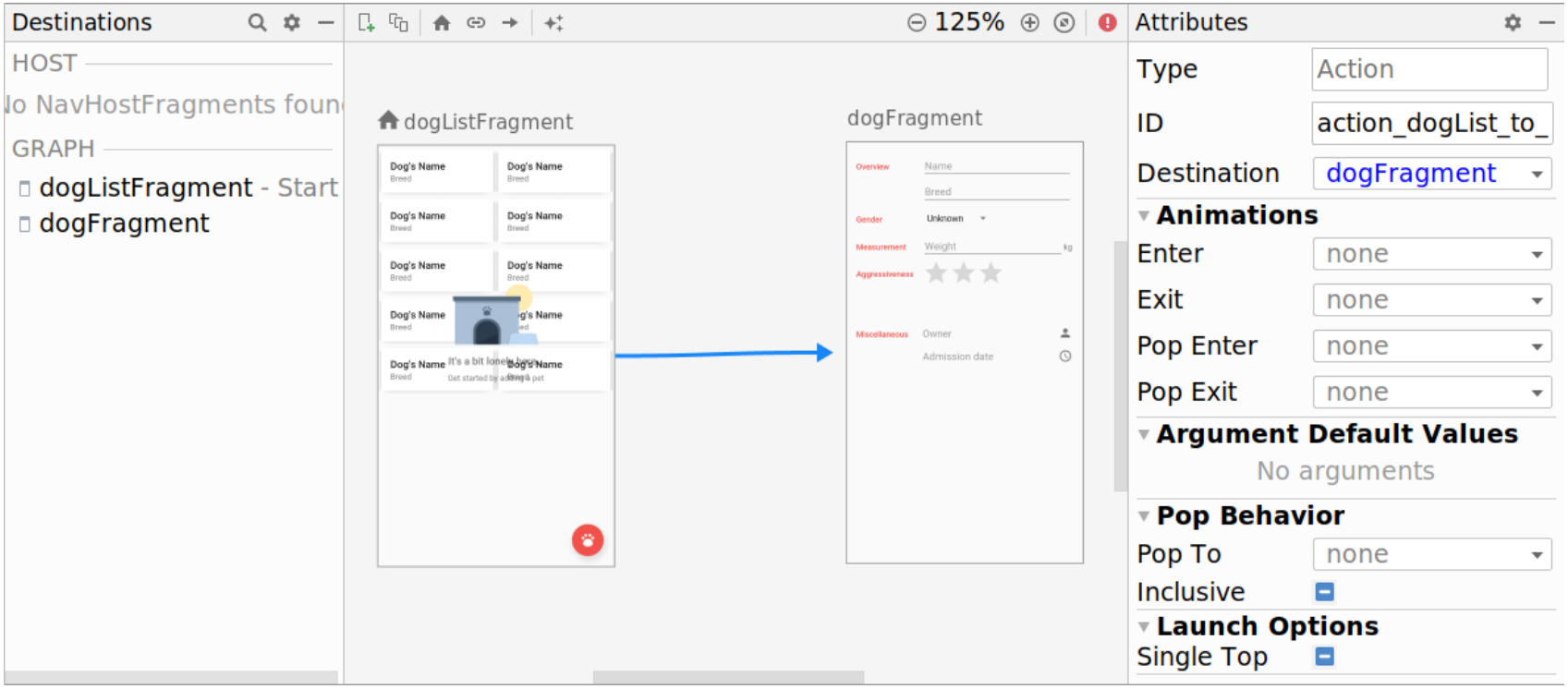
```
plugins {
    id 'androidx.navigation.safeargs.kotlin'
}
dependencies {
  implementation "androidx.navigation:navigation-fragment-ktx:$navVer"
  implementation "androidx.navigation:navigation-ui-ktx:$navVer"
}
```

- Pour utiliser le passage d'arguments sûr (Safe Args)
- Dans le `build.gradle` principal

```
buildscript {
  ext.navVer = '2.5.3'
  repositories {
    google()
  }
  dependencies {
    classpath "androidx.navigation:navigation-safe-args-gradle-plugin:$navVer"
  }
}
```

Définition du graphe de navigation

- Ajouter une nouvelle ressource de type `Navigation`
- Fichier XML dans le répertoire `navigation`
- Ajouter les fragments et définir les actions entre-eux



Au sein de la ressource `Navigation`

```
<navigation …
    android:id="@+id/nav_graph"
    app:startDestination="@id/dogListFragment">
    <fragment
        android:id="@+id/dogListFragment"
        android:name="fr.iut.ouafffaac.ui.fragment.DogListFragment"
        android:label="DogListFragment"
        tools:layout="@layout/fragment_list_dog">
    <fragment
        android:id="@+id/dogFragment" … />
    <dialog
        android:id="@+id/datePickerFragment" …/>
</navigation>
```

- Un fragment par destination (`app:startDestination` la première affichée)
- Balise `dialog` pour les `DialogFragment`
- `tools:layout` permet de spécifier le visuel à afficher dans l'éditeur
- Le reste est classique

Au sein de la ressource `Navigation`

- Navigation entre fragments = une action
- L'action est définie dans le fragment de départ
- On y spécifie le fragment de destination

```
<fragment
    android:id="@+id/dogListFragment"
    android:name="fr.iut.ouafffaac.ui.fragment.DogListFragment"
    android:label="DogListFragment"
    tools:layout="@layout/fragment_list_dog">
    <action
        android:id="@+id/action_dogList_to_dogDetail"
        app:destination="@id/dogFragment" />
</fragment>
```

Actions avec arguments

- Utilisation du plugin `Safe Args`
- Vérification des arguments à la compilation
- Pas oublier d'ajouter dans le `build.gradle` (section plugins) `id 'androidx.navigation.safeargs.kotlin'`
- Les arguments sont définis au niveau de la destination

```
<fragment
     android:id="@+id/dogFragment"
     android:name="fr.iut.ouafffaac.ui.fragment.DogFragment"
     android:label="DogFragment"
     tools:layout="@layout/fragment_dog" >
     <argument
         android:name="dogId"
         app:argType="long"
         android:defaultValue="0L" />
 </fragment>
```

Dans le layout de l'activité

- Ajouter le fragment de navigation : `NavHostFragment`

```
<androidx.fragment.app.FragmentContainerView …
    android:id="@+id/nav_fragment"
    android:name="androidx.navigation.fragment.NavHostFragment"
    …
    app:defaultNavHost="true"
    app:navGraph="@navigation/nav_graph" />
```

- Le lier au graphe de navigation grâce à `app:navGraph`
- `app:defaultNavHost` permettra de donner la gestion du Back Button à ce `NavHostFragment` (doit être unique)

Dans le code

- Récupérer le contrôleur de navigation
- Objet de type `NavController` … pour naviguer vers les destinations du graphe
- Plusieurs possibilités suivant l'endroit où on est :

```
Fragment.findNavController()
View.findNavController()
Activity.findNavController(viewId: Int)
```

1. Par exemple au niveau d'un des fragments du graphe

```
private lateinit var navController: NavController

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    navController = findNavController()
}
```

Dans le code

2. Au niveau de l'activité qui contient le `NavHostFragment`
   - Attention, si on utilise une `FragmentContainerView` ou qu'on injecte le `NavHostFragment` à la main par une `FragmentTransaction` le contrôleur n'est pas disponible dans le `onCreate()` en utilisant `findNavController(…)`

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    val navHostFragment =
        supportFragmentManager.findFragmentById(R.id.nav_host_fragment)
        as NavHostFragment
    val navController = navHostFragment.navController
    // Faire quelque chose avec navController
}
```

Naviguer vers une destination

- Il suffit d'appeler l'une des méthodes `navigate(…)` du `NavController`

1. Par exemple, navigation par id lors d'un clic sur un bouton

```
override fun onCreateView(…) {
    …
    val btn = view.findViewById<Button>(R.id.btn_nav_dog)
    btn.setOnClickListener(this::gotoDogFragment)
    …
}

fun gotoDogFragment(view: View) {
    navController.navigate(R.id.dog_fragment)
}
```

- Possibilité d'utiliser une méthode factory pour le `OnClickListener`

```
btn.setOnClickListener(
    Navigation.createNavigateOnClickListener(R.id.dog_fragment))
```

Utilisation du plugin Safe Args (simple)

- Si Safe Args est activé dans le projet, possibilité de naviguer grâce aux actions
- Pour un fragment `XXXFragment` une classe `XXXFragmentDirections` est auto-générée
- Elle contient une méthode par action déclarée dans le graphe
- Il suffit d'appeler `navigate()` avec en paramètre l'action créée par cette méthode pour naviguer vers la destination en question

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
  super.onViewCreated(view, savedInstanceState)

  view.findViewById<Button>(R.id.btn_nav_dog).setOnClickListener {
    val action = DogListFragmentDirections.actionDogListFragmentToDogFragment()
    navController.navigate(action)
  }
}
```

Utilisation du plugin Safe Args (avec passage d'arguments)

- De même pour les fragments acceptant des arguments
- Pour un fragment de destination `XXXFragment` une classe `XXXFragmentArgs` est auto-générée
- Elle contient un attribut (typé à la compilation) par argument déclaré dans le graphe
- L'instance est récupérable par la propriété déléguée `by navArgs<XXXFragmentArgs>()`

```
class DogFragment : Fragment () {
    private val args by navArgs<DogFragmentArgs>()
    …

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        …
        val dogId = args.dogId
        …
    }
}
```

Utilisation du plugin Safe Args (avec passage d'arguments)

- Les arguments sont envoyés à la destination grâce aux méthodes de `XXXFragmentDirections`
- Les méthodes d'actions de cette classe prennent alors des paramètres qui sont les valeurs à envoyer à la destination

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
  super.onViewCreated(view, savedInstanceState)

  view.findViewById<Button>(R.id.btn_nav_dog).setOnClickListener {
    val dogId = 42L
    val action =
        DogListFragmentDirections.actionDogListFragmentToDogFragment(dogId)
    navController.navigate(action)
  }
}
```

`NavigationUI`

- Lien entre navigation et widgets classiques (`AppBar` , `NavigationDrawer` , `BottomNavigation`) possible
- Grâce à `NavigationUI` = ensemble de méthodes `setupWithNavController()` qui font le travail pour vous
- Le lien est mis en place à l'aide d'une classe utilitaire : `AppBarConfiguration`
- Par exemple pour l'action bar (m.-à-j. du titre affiché) :

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    val navHostFragment =
        supportFragmentManager.findFragmentById(R.id.nav_host_fragment) as NavHostFragment
    navController = navHostFragment.navController

    val appBarConfiguration = AppBarConfiguration(navController.graph)
    setupActionBarWithNavController(navController, appBarConfiguration)
}
```

`NavigationUI`

- Pour les composants permettant une navigation (`NavigationDrawer`, `BottomNavigation`)
- L'association est faite grâce aux id des menu utilisés
- Ils doivent être les mêmes que les id des destinations et le tour est joué