Recycler View

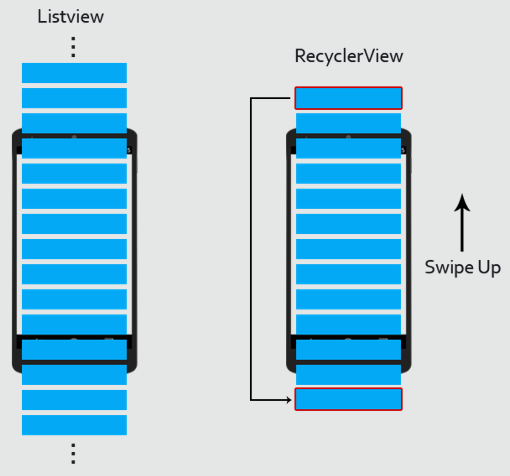
1. Objectif

* Savoir afficher une liste dans une application.
* Comprendre la plus-value des RecyclerView.
* Savoir mettre en place un RecyclerView.
* Savoir définir une action sur la sélection d’un item de RecyclerView.

1. Présentation

Les RecyclerView sont des listes optimisées.

S'il y a 10000 données dans une liste et que cette liste est fournie à un RecyclerView alors seules quelques données sont chargées à l'écran. Les données chargées à l'écran sont celles visibles et elles sont chargées dans des ViewHolders, le contenu des ViewHolders change lorsque l'utilisateur scroll sur la liste. Il y a donc qu'un petit nombre de ViewHolder. L'ancêtre du RecyclerView est la ListView qui chargeait la totalité des données dans l'IHM, cela avait pour effet de ralentir des applications avec beaucoup de données.



1. Mise en place

* Définir le ViewModel de la vue : Le ViewModel contient les données à afficher
* Définir un RecyclerView dans la vue​​​​​ : Elément graphique représentant la liste à afficher.
* Définir un fichier définissant le style pour une ligne de liste
* Définir un adapter : chargée de fournir des lignes de liste chargées de données à la liste
* Initialiser et lier les éléments

1. Démonstration : Définir le ViewModel de la vue

Le ViewModel apporte les données nécessaires pour la vue. Notre vue doit afficher une liste de personnes donc notre ViewModel ne contient qu'une liste de personne.

class ListePersonnesViewModel() : ViewModel()

{

//Liste de personne

val listePersonnes:List<Personne>=listOf(Personne(0,"Cos","joe"),Personne(0,"Son","Nick"),Personne(0,"Son","Leï"))

}

1. Démonstration : Définir un RecyclerView dans la vue

Le code suivant montre quelle balise utiliser pour définir un RecyclerView dans une IHM. C'est suffi tout simplement de mettre une balise RecyclerView dans le layout de l'IHM.

<layout 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">

<FrameLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".liste\_personnes.ListePersonnesFragment">

<androidx.recyclerview.widget.RecyclerView

android:id="@+id/rv\_personnes"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

!IMPORTANT! app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager" />

</FrameLayout>

</layout>

* Attention : l'attribut layoutManager avec la valeur indiquée est obligatoire !

1. Démonstration : Définir le style d'une ligne

Le code suivi montre comment définir le style des lignes affichées dans un RecyclerView.

Le style d'une ligne est défini dans un nouveau fichier XML positionné dans le dossier layout.

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

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

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical"

android:layout\_margin="16dp">

​

<TextView

android:id="@+id/tv\_nom"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_weight="1"

android:text="@{personne.nom}" />

​

<TextView

android:id="@+id/tv\_prenom"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_weight="1"

android:text="@{personne.prenom}" />

</LinearLayout>

</layout>

1. Démonstration : Définir un adapter

Le code suivant montre comment définir un adapter.

L'adapter va remplir votre liste, oui son code est imbuvable et peu intuitif, si vous ne le maîtrisez pas c'est normal ! Explications en commentaire !

class ListPersonnesAdapter : ListAdapter<Personne, ListPersonnesAdapter.ViewHolder>(PersonneDiffCallback())

{

//Fonction appelée automatiquement pour changer le contenu d'un conteneur de ligne

override fun onBindViewHolder(holder: ViewHolder, position: Int)

{

//On va chercher la données à afficher, c'est le sysème qui s'occupe de vous fournir la position

val item = getItem(position)

//holder représente un conteneur et on lui donne la donnée à charger

holder.bind(item)

}

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder

{

return ViewHolder.from(parent)

}

//Représente un conteneur de ligne

class ViewHolder private constructor(val binding: ItemPersonneBinding) : RecyclerView.ViewHolder(binding.root)

{

//Fait le travail de chargement des données dans la ligne

fun bind(item: Personne)

{

//On fournit à la ligne les données, le dataBinding de l'IHM fait le reste

binding.personne = item

binding.executePendingBindings()

}

//Ici c'est toujours pareil

companion object

{

fun from(parent: ViewGroup): ViewHolder

{

val layoutInflater = LayoutInflater.from(parent.context)

val binding = ItemPersonneBinding.inflate(layoutInflater, parent, false)

return ViewHolder(binding)

}

}

}

}

//Ici c'est toujours pareil aussi

class PersonneDiffCallback : DiffUtil.ItemCallback<Personne>()

{

override fun areItemsTheSame(oldItem: Personne, newItem: Personne): Boolean

{

return oldItem.id == newItem.id

}

override fun areContentsTheSame(oldItem: Personne, newItem: Personne): Boolean

{

return oldItem == newItem

}

}

1. Démonstration : Initialiser et les lier les éléments

Le code suivant montre comment lier tous les éléments vus précédemment dans le contrôleur de la vue contenant le RecyclerView.

class ListePersonnesFragment : Fragment()

{

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,savedInstanceState: Bundle?): View?

{

val binding: FragmentListePersonnesBinding = DataBindingUtil.inflate(

inflater,

R.layout.fragment\_liste\_personnes,

container,

false)

val listePersonnesViewModel =

ViewModelProvider(this).get(ListePersonnesViewModel::class.java)

val adapter = ListPersonnesAdapter()

binding.rvPersonnes.adapter = adapter

adapter.submitList(listePersonnesViewModel.personnes)

​

return binding.root

}

}

1. Démonstration : Interaction avec les items
   1. Présentation

Il est possible d'exécuter une action lors d'un clic sur un élément d'un RecyclerView.

* 1. Mise en place

Créer un listener

Lier le listener à l’action onClick de chaque ligne du RecyclerView dans l’IHM.

Lier l'action dans l’adapter

Définir l’action à exécuter

* 1. Démonstration : Créer un listener

Le code suivant montre comment définir un listener personnalisé.

class PersonneListener(val clickListener: (personneId: Long) -> Unit)

{

fun onClick(personne: Personne) = clickListener(personne.id)

}

* 1. Démonstration : Lier le listener à l’action onClick

Le code suivant montre comment lier un listener personnalisé dans la définition du style des lignes affichées dans le RecyclerView. L'utilisation du databinding est utilisé pour faire cette liaison.

<data>

<variable

name="personne"

type="fr.eni.utilisateurauhasardkotlin.model.Personne" />

<variable

name="clickListener"

type="fr.eni.utilisateurauhasardkotlin.liste\_personnes.PersonneListener"/>

</data>

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical"

android:layout\_margin="16dp"

android:onClick="@{() -> clickListener.onClick(personne)}">

[…]

* 1. Démonstration : Lier l'action dans l’adapter

Le code suivant montre que l'adapter attend en parametre une instance du listener personnalisé, cela indique que l'adapter attend en paramètre l'action qui sera réalisé par le RecyclerView.

class ListPersonnesAdapter(val clickListener: PersonneListener) : ListAdapter<Personne,

ListPersonnesAdapter.ViewHolder>(PersonneDiffCallback()) {

override fun onBindViewHolder(holder: ViewHolder, position: Int)

{

val item = getItem(position) holder.bind(clickListener,item)

}

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder

{

return ViewHolder.from(parent) }

class ViewHolder private constructor(val binding:ItemPersonneBinding):RecyclerView.ViewHolder(binding.root)

{

fun bind(clickListener: PersonneListener,item: Personne) {

Log.i("ACOS","BIND")

binding.personne = item

binding.clickListener = clickListener

binding.executePendingBindings()

}

[…]

* 1. Démonstration : Définir l'action à exécuter

Le code suivant montre comment définir l'instance du listener et comment la fournir à l'adapter à 'l'intérieur du contrôleur de la vue contenant le RecyclerView.

val adapter = ListPersonnesAdapter(PersonneListener

{

personneId -> Toast.makeText(context, "${personneId}", Toast.LENGTH\_LONG).show()

})

binding.rvPersonnes.adapter = adapter

1. Conclusion

La création d'un RecyclerView est fastidieuse, les prochaines versions de l'API offriront certainement de nouvelles façons de faire des RecyclerView de manière plus simplifiée. Lors de l'apprentissage et des premiers développements de ce type de composants il est tout à fait normal d'être toujours accompagné d'un modèle. Ce chapitre permet donc d'avoir un exemple concret de RecyclerView et de comprendre son utilité.