PROGRAMMATION DE COMPOSANTS MOBILES (ANDROID)

Wieslaw Zielonka

BroadcastReceiver

Installer le BroadcastReceiver

BroadcastReceiver : un composant Android (comme Activity ou Service) qui reçoit les Intents envoyés en diffusion (broadcast).

Deux manières pour installer un BroadcastReceiver:

Installer le BroadcastReceiver de façon statique

Déclarer dans AndroidManifest.xml :

```
<receiver android:name=".MyReceiver"
          android:exported="true" ... >
```

où `MyReceiver` votre implémentation de `BroadcastReceiver` -- classe dérivée de `BroadcastReceiver`

`MyReceiver` sera toujours disponible, peu importe l'état de l'application dont il fait parti.

Installer le BroadcastReceiver de façon dynamique

Dans votre programme, par exemple dans le ViewModel d'une activité :

définir l'objet qui implémente le BroadcastReceiver :

```
private val receiver = object : BroadcastReceiver(){

    override fun onReceive(p0 : Context?, p1 : Intent?){
    /* implementer onReceive() qui est appelé à la
     * reception d'un Intent envoyé en diffusion */


    }
}
```

Installer le BroadcastReceiver de façon dynamique

Pour faire démarrer le BroadcastReceiver dynamique il faut d'abord l'enregistrer.
Ici le code copier depuis un ViewModel :

```
/* preparer les filtres pour BroadcastReceiver */
val filter = IntentFilter(DownloadManager.ACTION_DOWNLOAD_COMPLETE)
/* enregistrer les BroadcastReceivers */
getApplication<Application>().registerReceiver(receiver, filter)
```

Ce receiver reçoit uniquement les actions (Intents) de la part de BroadcastReceiver, de type DownloadManager.*ACTION_DOWNLOAD_COMPLETE.*

C'est le premier filtrage, cela évite de recevoir les Intents envoyés en diffusion pur d'autres raisons.

Si nous voulons installer un autre BroadcastReceiver qui reçoit les Intents quand utilisateur touche une notification de DownloadManager alors on procède de façon suivante :

```
val cancelFilter = IntentFilter(DownloadManager.ACTION_NOTIFICATION_CLICKED)

getApplication<Application>().registerReceiver(cancelReceiver, cancelFilter)
```

Implémenter un BroadcastReceiver

On suppose qu'on stocke dans le ModelView les objets suivants :

```
// map pour les couples
// <id de téléchargement ,  url la source de téléchargementnt >
var mapIdToUrl: MutableMap<Long, String> = mutableMapOf()

// map de couple <id téléchargement, nom de fichier local
var mapIdToFile: MutableMap<Long, String> = mutableMapOf()

// la liste de bitmaps à afficher, obtenue en transformant
// les images jpeg, pong ou autre en bitmaps
val bitmapList = mutableListOf<Bitmap>()
val liveBitmapList: MutableLiveData<List<Bitmap>> =
    MutableLiveData(bitmapList)
```

Implémenter un BroadcastReceiver

```
private val receiver = object : BroadcastReceiver() {
    override fun onReceive(p0: Context?, p1: Intent?) {

        /* trouver la reference de téléchargement */
        val idDownload: Long = p1?.getLongExtra(
            DownloadManager.EXTRA_DOWNLOAD_ID, -1
        ) ?: -1

        if (idDownload == -1L) {
            Log.d("Receiver", "something wrong with idDownload")
            return
        }

        /* vérifier si la référence se trouve sur la liste
         * de nos téléchargement
        *  nécessaire si d'autres téléchargement par d'autres applications
*/
        if (idDownload !in mapIdToUrl.keys) {
            Log.d("idDownload", "$idDownload  n'est pas sur la liste de
telech")
            return
        }
```

Implémenter un BroadcastReceiver

```
private val receiver = object : BroadcastReceiver() {
    override fun onReceive(p0: Context?, p1: Intent?) {

        /* trouver la reference de téléchargement */
        val idDownload: Long = p1?.getLongExtra(
            DownloadManager.EXTRA_DOWNLOAD_ID, -1
        ) ?: -1

        if (idDownload == -1L) {
            Log.d("Receiver", "something wrong with idDownload")
            return
        }

        /* vérifier si la référence se trouve sur la liste
         * de nos téléchargement
        *  nécessaire si d'autres téléchargement par d'autres applications
*/
        if (idDownload !in mapIdToUrl.keys) {
            Log.d("idDownload", "$idDownload  n'est pas sur la liste de
telech")
            return
        }
```

Implémenter un BroadcastReceiver

```
//transformer l'image téléchargé en un Bitmap dans un  nouveau thread
// parce que cette opération peut prendre beaucoup de temps
thread {
    /* créer un  File pour la destination de téléchargement */
    val file = File( getApplication<Application>()
                .getExternalFilesDir(Environment.DIRECTORY_PICTURES),
                 mapIdToFile[idDownload]  //le nom de fichier destination
    )

    if (file == null) {
        Log.d("file null ",
            "${mapIdToFile[idDownload]}  ${mapIdToUrl[idDownload]}")
        return@thread
    }

    /*  transformer le fichier d'image en bitmap
     * à ne pas faire dans le thread principal*/

val bitmap = BitmapFactory.decodeFile(  file.absolutePath  )
```

Implémenter un BroadcastReceiver

```
    if (bitmap == null) {
        Log.d("fichier null ",
     "${mapIdToFile[idDownload]}  ${mapIdToUrl[idDownload]}")
        return@thread
    }

`

/* plusieurs thread peuvent essayer ajouter les bitmaps
   * en même temps il faut synchronizer */

    synchronized(bitmapList) {
        bitmapList.add(bitmap)
        //le contenu de bitmapList a changé,
        //il faut la remettre dans LiveData pour que
        //l'observer réagisse
        liveBitmapList.postValue(bitmapList)
    }
}// fin thread
```

Implémenter un BroadcastReceiver

```
    if (bitmap == null) {
        Log.d("fichier null ",
     "${mapIdToFile[idDownload]}  ${mapIdToUrl[idDownload]}")
        return@thread
    }

`

/* plusieurs thread peuvent essayer ajouter les bitmaps
   * en même temps il faut synchronizer */

    synchronized(bitmapList) {
        bitmapList.add(bitmap)
        //le contenu de bitmapList a changé,
        //il faut la remettre dans LiveData pour que
        //l'observer réagisse
        liveBitmapList.postValue(bitmapList)
    }
}// fin de thread
}} // et fin de la définition de BroadcastReceiver
```

et si on n'a pas donner de fichier de destination

```
thread {

    lateinit var parcelFileDescriptor: ParcelFileDescriptor
    try {
        parcelFileDescriptor =
            downloadManager.openDownloadedFile(idDownload)
    } catch (e: FileNotFoundException) {
        Log.d("fichier", " '${mapIdToUrl[idDownload]}' non trouvé")
        return@thread
    }


    val bitmap: Bitmap = BitmapFactory
            .decodeFileDescriptor( parcelFileDescriptor.fileDescriptor )
    parcelFileDescriptor.close()

  synchronized(bitmapList) {
      bitmapList.add(bitmap)
     //le contenu de bitmapList a changé,
     //il faut la remettre dans LiveData pour que l'observer reagisse
     liveBitmapList.postValue(bitmapList)
  }

}// fin thread
```

Il faut créer et enregistrer le BroadcastReceiver dynamique avant l'envoi de demandes de téléchargement vers le DownloadManager.

Pourquoi ?

Affichage de l'image

On affiche les images dans ImageView :

```
<ImageView
    android:id="@+id/image"
    android:layout_width="match_parent"
    android:layout_weight="6"
    android:layout_height="wrap_content"
    />
```

L'image à afficher doit être de type **Bitmap.**

Pour obtenir un Bitmap à partir de fichier avec un image png, jpeg etc on utilise les fonctions static de la classe java BitmapFactory :

```
imageView.setImageBitmap(  bitmap )
```

BitmapFactory possède plusieurs fonctions static pour décoder l'image

BitmapFactory .decodeFile(.....) : Bitmap

permissions AndroidManifest

A ajouter avant la balise <aplication ...>

```
<uses-permission android:name="android.permission.INTERNET"/>
```

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

Chaque application qui fait de connexion internet doit avoir la permission INTERNET.

Exemple complet

Téléchargement des images depuis internet et l'affichage dans RecyclerView.

Toutes les données dans un (Android)ViewModel associé à l'activité.

Exemple complet - Activité

```
class MainActivity : AppCompatActivity() {

    val binding by lazy{ ActivityMainBinding.inflate(layoutInflater) }
    val viewModel by lazy { ViewModelProvider(this).get(MyViewModel::class.java) }

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

        //recuprer le tableau d'adresse http dans les ressources
        viewModel.urlListe = resources
            .getStringArray(R.array.httpAdresses).toList()

        if (resources.configuration.orientation == Configuration.ORIENTATION_PORTRAIT) {
            binding.recyclerView.layoutManager = LinearLayoutManager(this)
        } else {
            binding.recyclerView.layoutManager = GridLayoutManager(this, 3)
        }

        /*créer Adapter et attache au recyclerView */
        val adapter = Adapter().apply {
            binding.recyclerView.adapter = this
            bitmapList = viewModel.bitmapList
            mapListPositionToChecked = viewModel.mapListPositionToChecked
        }

        viewModel.liveBitmapList.observe(this){
            adapter.notifyDataSetChanged()
        }

        /* commencer le téléchargement */
        viewModel.fetchBitmaps()
```

RecyclerView.Adapter

```
class Adapter : RecyclerView.Adapter<Adapter.VH>() {

    /* liste de Bitmaps à afficher */
    lateinit var bitmapList: List<Bitmap>

    /* le map qui pour chaque couple <i, b> où i une position sur la liste bitmapList
     * b == true si et seulement si le checkBox correspondant est sélectionné
     * Par défaut si (i !in mapListPositionToChecked ) alors b == false.
     */
    lateinit var mapListPositionToChecked: MutableMap<Int, Boolean>

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): Adapter.VH {
        val binding = ImageBinding.inflate( LayoutInflater.from(parent.context), parent, false)

        val holder = VH( binding )
        binding.checkbox
       .setOnClickListener{  mapListPositionToChecked[ holder.absoluteAdapterPosition ] =
                                                    holder.binding.checkbox.isChecked    }
        return holder
    }

    override fun onBindViewHolder(holder: Adapter.VH, position: Int) {
        holder.binding.image.setImageBitmap(  bitmapList[position] )
        holder.binding.checkbox.isChecked = mapListPositionToChecked[position] ?: false
    }

    override fun getItemCount(): Int {
        return bitmapList.size
    }

    class VH(val binding : ImageBinding ) : RecyclerView.ViewHolder( binding.root )

}
```

AndroidViewModel

```
class MyViewModel(application: Application) : AndroidViewModel(application) {

    /* récupérer le DownloadService */
    val downloadManager = //getApplication<Application>()
        application.getSystemService(Context.DOWNLOAD_SERVICE) as
DownloadManager

    /* liste des adresses http des images à télécharger */
    lateinit var urlListe: List<String>

    var mapListPositionToChecked: MutableMap<Int, Boolean> = mutableMapOf()

    // map pour les couples <id de téléchargement , url de téléchargement >
    var mapIdToUrl: MutableMap<Long, String> = mutableMapOf()

    var mapIdToFile: MutableMap<Long, String> = mutableMapOf()

    val bitmapList = mutableListOf<Bitmap>()
    val liveBitmapList: MutableLiveData<List<Bitmap>> =
        MutableLiveData(bitmapList)


private val receiver = object : BroadcastReceiver() {// déjà défini dans les transparents précédents }
```

AndroidViewModel

```
// un autre BroadcastReceiver pour supprimer un téléchargement non terminé

val cancelReceiver = object : BroadcastReceiver() {
    override fun onReceive(p0: Context?, p1: Intent?) {

        /* trouver les references de téléchargement à supprimer */
        val reference = p1?.getLongArrayExtra(
            DownloadManager.EXTRA_NOTIFICATION_CLICK_DOWNLOAD_IDS
        )
.....................
        thread {
    downloadManager.remove(*reference)
    Log.d("remove download size = ", "${reference.size}")
        }
}
```

AndroidViewModel

```
// débuter le téléchargement

fun fetchBitmaps() {
    /* preparer les filtres pour BroadcastReceiver */
    val filter = IntentFilter(DownloadManager.ACTION_DOWNLOAD_COMPLETE)
    val cancelFilter = IntentFilter(DownloadManager.ACTION_NOTIFICATION_CLICKED)

    /* enregistrer les BroadcastReceivers */
    getApplication<Application>().registerReceiver(receiver, filter)
    getApplication<Application>().registerReceiver(cancelReceiver, cancelFilter)

    var i = 1
    for (adr in urlListe) {
        /* fabriquer Uri de telechargement*/
        val uri = Uri.parse(adr)

        var nomFichier = "image" + i + ".jpeg"
        i++

        if( adr in mapIdToUrl.values)
            return

    val request = DownloadManager.Request(uri)
         .setNotificationVisibility(DownloadManager.Request.VISIBILITY_VISIBLE)
         .setDestinationInExternalFilesDir(
             getApplication<Application>(),
             Environment.DIRECTORY_PICTURES,
             nomFichier
         )
     val idLoad: Long = downloadManager.enqueue(request)

     mapIdToFile[idLoad] = nomFichier
     /* ajouter <id, adresse> de téléchargement à la liste */
     //idListe.add(id)
     mapIdToUrl[idLoad] = adr
```

AndroidViewModel

```
/* onCleared() called when the ViewModel no longer used */
override fun onCleared() {
    super.onCleared()
    /* unregister les BroadcastReceivers */
    getApplication<Application>().unregisterReceiver(receiver)
    getApplication<Application>().unregisterReceiver(cancelReceiver)
}
```

Des-enregistrer les BroadcastReceivers (seulement dans le cas dynamique)