PROGRAMMATION DE COMPOSANTS MOBILES (ANDROID)

Notification

WIESLAW ZIELONKA

Notifications

Envoi de notification

Préparation d'envoi de notification

1. Préparer Intent qui sera envoyer quand l'utilisateur touche la notification
2. "Emballer" cet intent dans un PendingIntent
3. Construire et enregistrer un "notification channel" (sauf si le channel existe déjà)
4. Envoyer la notification vers le NotificationManager. Vous attribuez un id (Int) à la notification. Cet id permet ensuite :
   - supprimer la notification depuis le programme
   - remplacer la notification par une autre (en gardant id)

PendingIntent

PendingIntent est un objet qui contient un Intent.

Supposons que l'application B reçoit un PendingIntent contenant un Intent de l'application A.

L'application B peut utiliser le Intent obtenu avec les droits de l'application A.

Utilisation de PendingIntent :

- déclarer un Intent à envoyer quand l'utilisateur agit sur une Notification (PendingIntent est utilisé par NotificationManager)
- déclarer un Intent à utiliser à un moment précis dans le futur (Intent à utiliser par AlarmManager)
- déclarer un Intent à utiliser quand l'utilisateur interagit avec AppWidget

PendingIntent

Application A

prépare PendingIntent
PendingIntent contient
un intent qui sera envoyé
qu'on sélectionne la notification
dans NotificationDrawer

envoie PendingIntent
vers NotificationManager

NotificationManager

reçoit PendingIntent

Utilisateur clique sur
la notification

NotificationManager utilise Intent
contenu dans PendingIntent pour lancer
une activité (avec les droits de
l'application A).

nouvelle activité C

Création de PendingIntent

`Intent` sera utilisé par un composant spécifique : `Activity, Service, BroadcastReceiver.`

Trois méthodes static de la classe `PendingIntent` pour construire un `PendingIntent` spécifique :

PendingIntent

`PendingIntent getActivity(Context context, int requestCode, Intent intent, int flags)`
 	
`PendingIntent getBroadcast(Context context, int requestCode, Intent intent, int flags)`
 	
`PendingIntent getService(Context context, int requestCode, Intent intent, int flags)`

`intent` -- Intent associé à PendingIntent

Notification simple (et pas vraiment correcte)

L'exemple : une application composée de deux activités.

1. L'activité Main envoie vers NotificationManager une Notification contenant un PendingIntent. Le PendingIntent contient un Intent qui permet de démarrer l'activité C.
2. Un clic sur la notification dans NotifictionDrawer lance l'activité C.

NotificationManager un service de Android qui gère les notification.

NotificationManager

Pour récupérer une référence vers le NotificationManager :

```
val notificationManager by lazy
{ getSystemService(NOTIFICATION_SERVICE) as NotificationManager }
```

NotificationChannel

A partir de Android 8.0 (API 26) on attribue à chaque notification un NotificationChannel.

Le channel possède la propriété importance qui indique la priorité des toutes les notifications envoyées sur ce channel.

Le channel sert à regrouper les notifications.
Par exemple interdir que les notification envoyées sur un channel particulier le dérangent.

NotificationChannel

A partir de Android 8.0 (API 26) on attribue à chaque notification un NotificationChannel.

Le channel possède la propriété importance qui indique la priorité des toutes les notifications envoyées sur ce channel.

Le channel sert à regrouper les notifications.
Par exemple interdir que les notification envoyées sur un channel particulier le dérangent.

création de NotificationChannel

La fonction createNotificationManager() ci-dessus doit être appelée avant qu'on envoie une notification. Cela n'a pas d'importance si la fonction est appelée plusieurs fois. C'est le premier appel qui créera le channel demandé.
Si le channel existe pas la peine d'appeler cette fonction mais la faire appeler n'est pas nuisible non plus.

```
private fun createNotificationChannel() {
    // Create the NotificationChannel, but only on API 26+ because
    // the NotificationChannel class is new and not in the support library
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        val channel = NotificationChannel(
            CHANNEL_ID,  /* Int : identifiant de channel */
            "private channel",  /* String : nom de channel */
            NotificationManager.IMPORTANCE_DEFAULT
        ).apply { description = "private channel" }

        // Register the channel with the system;
        // you can't change the importance
        // or other notification behaviors after this
        notificationManager.createNotificationChannel(channel)
}
Vous devez adapter CHANNEL_ID,  nom de channel et description */
```

envoi de notification

```
/* préparer entent que NotificationManager envoie quand l'utilisateur
touche la notification */
val intent = Intent(this, ActiviteC::class.java)



/* préparer le PendingIntent qui contient l'intent précédent */


    val pendingIntent = PendingIntent.getActivity(
        context = this,
        requestCode=1,
        intent,
        PendingIntent.FLAG_IMMUTABLE   )
```

construire la notification

```
/*création de notification */
val notification = NotificationCompat.Builder(this, CHANNEL_ID)
    .setContentTitle(  titre )
    .setContentText( message )
    .setContentIntent(pendingIntent)
    .setAutoCancel(true)
    .setSmallIcon(R.drawable.small)
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)
    .setLargeIcon( BitmapFactory.decodeResource( resources,
R.drawable.large ) )
    .build()
```

id de notification channel

SmallIcon est obligatoire,

*setLargeIcon() : prend un bitmap en paramètre donc si l'image dans un autre format alors utilisez BitmapFactory pour transformer en bitmap (mais LargeIcon n'est pas obligatoire */*

priority concerne les version android <= 7.1, à partir de la version 8 c'est le channel qui est utilisé

setAutoCancel(true) : la notification disparaît quand utilisateur la touche

ContentTitle - titre de la notification

ContextText - le texte de la notification

envoi de notification

```
/* envoi de notification */
 notificationManager.notify(notId, notification)
```

notId : ID (Int) de la notification, peut-être utilisé pour faire cancel et update

remplacer une notification

Pour remplacer un notification il suffit d'envoyer une autre notification avec le même identifiant notId

supprimer une notification

```
notificationManager.cancel( notId )
```

supprimer une Notification

Une notification reste visible jusqu'à ce que :

- l'utilisateur écarte la notification du Notification drawer
- l'utilisateur tape sur la notification et on a utilisé `NotificationCompat.Builder.setAutoCancel(true)` pendant la création de la notification
- on exécute la méthode

  `NotificationManager.cancel( int id )`

  avec l'ID de la notification ou la méthode

le problème de navigation correcte)

Le problème :

le bouton Back n'a pas de comportement qui correspond aux recommandations d'Android.

Il faut que l'activité lancée depuis notifications soit unique dans une nouvelle tâche.

La solution :

https://developer.android.com/training/notify-user/navigation

(la section : start an activity from a notification)

Les notifications avec le bouton d'action

Parfois nous voulons avoir une notification avec un bouton d'action.

Quand l'utilisateur touche la notification en dehors de bouton, il est envoyé vers une activité -- c'est le mécanisme présenté dans les transparents précédents.

Quand l'utilisateur clique sur le bouton de notification il y a un intent envoyé soit vers un Service soit vers un BroadcastReceiver.

Donc il y a maintenant deux intentas à préparer qu'il faudra emballer dans deux PendintIntent.

notification avec le bouton d'action

```
val notificationManager : NotificationManager =
    getSystemService(Context.NOTIFICATION_SERVICE)

/* intent à envoyer vers un service MyService */
val cancel_intent = Intent(this, MyService::class.java)
                    .apply{ action = "stop" }

val pending_cancel = PendingIntent.getService(
            context = this, reqCode = NOTIFICATION_CODE,
            cancel_intent, pendingFlag )


val notification: Notification = NotificationCompat.Builder(this, CHANNEL_ID)
    .setContentTitle(CONTENT_TITLE)
    .setContentText("cancel stop alarm")
    .setContentIntent( intent ),  /* le contentIntent avec un intent vers Activity*/
    .setSmallIcon(R.drawable.small)
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)
    .setVisibility(NotificationCompat.VISIBILITY_PUBLIC)
    .addAction(
        R.drawable.presence_audio_busy, /* l'icône affichée dans le bouton */
       "stop service",    /* texte dans le bouton */
        pending_cancel    )
    .setAutoCancel(true)
    .build()

notificationManager.notify( NOTIFICATION_ID, notification )
```