24

Lancement d'activités et de sous-activités

La théorie sous-jacente de l'architecture de l'interface utilisateur d'Android est que les développeurs devraient décomposer leurs applications en activités distinctes, chacune étant implémentée par une `Activity` accessible *via* des intentions, avec une activité "principale" lancée à partir du menu d'Android. Une application de calendrier, par exemple, pourrait avoir des activités permettant de consulter le calendrier, de visualiser un simple événement, d'en modifier un (et d'en ajouter un), etc.

Ceci implique, bien sûr, que l'une de vos activités ait un moyen d'en lancer une autre. Si, par exemple, l'utilisateur clique sur un événement à partir de l'activité qui affiche tout le calendrier, vous voudrez montrer l'activité permettant d'afficher cet événement. Ceci signifie que vous devez pouvoir lancer cette activité en lui faisant afficher un événement spécifique (celui sur lequel l'utilisateur a cliqué).

Cette approche peut utiliser deux scénarios :

- Vous connaissez l'activité à lancer, probablement parce qu'elle fait partie de votre application.
- Vous disposez d'une `Uri` vers… quelque chose et vous voulez que vos utilisateurs puissent en faire… quelque chose, bien que vous ne sachiez pas encore comment.

Ce chapitre présente le premier scénario ; le suivant détaillera le second.

Activités paires et sous-activités

Lorsque vous décidez de lancer une activité, une question essentielle à laquelle vous devez répondre est : "Est-ce que mon activité a besoin de savoir quand se termine l'activité qu'elle a lancée ?"

Supposons par exemple que vous vouliez créer une activité pour collecter des informations d'authentification pour un service web auquel vous vous connectez – vous devrez peut-être vous authentifier avec OpenID[1] pour utiliser un service OAuth[2]. En ce cas, votre activité principale devra savoir quand se termine l'authentification pour pouvoir commencer à utiliser le service web.

Imaginons maintenant une application de courrier électronique Android. Lorsque l'utilisateur décide de visualiser un fichier attaché, ni vous ni l'utilisateur ne s'attend à ce que l'activité principale sache quand cette visualisation se terminera.

Dans le premier scénario, l'activité lancée est clairement subordonnée à l'activité qui l'a lancée. La première sera donc sûrement lancée comme une sous-activité, ce qui signifie que la seconde sera prévenue de la fin de son activité fille.

Dans le second scénario, l'activité lancée est plutôt un "pair" de l'activité qui l'a lancée. Elle sera donc plutôt lancée comme une activité classique. Votre activité ne sera pas informée de la fin de sa "fille" mais, encore une fois, elle n'a pas vraiment besoin de le savoir.

Démarrage

Pour démarrer une activité, il faut une intention et choisir comment la lancer.

Création d'une intention

Comme on l'a expliqué au Chapitre 1, les intentions encapsulent une requête pour une activité ou une demande adressée à un autre récepteur d'intention, afin qu'il réalise une certaine tâche.

1. **http://openid.net/**.
2. **http://oauth.net/**.

Si l'activité que vous comptez lancer vous appartient, il peut être plus simple de créer une intention explicite, nommant le composant à lancer. À partir de votre activité, vous pourriez par exemple créer une intention de la façon suivante :

```
new Intent(this, HelpActivity.class);
```

Cette instruction indique que vous voulez lancer `HelpActivity`.

Vous pourriez également créer une intention pour une `Uri` donnée, demandant une action particulière :

```
Uri uri=Uri.parse("geo:" + lat.toString() + "," + lon.toString());
Intent i=new Intent(Intent.ACTION_VIEW, uri);
```

Ici, à partir de la latitude et de la longitude d'une position (respectivement `lat` et `lon`), nous construisons une `Uri` de schéma `geo` et nous créons une intention demandant de l'afficher (`ACTION_VIEW`).

Faire appel

Lorsque l'on dispose de l'intention, il faut la passer à Android et récupérer l'activité fille à lancer. Quatre choix sont alors possibles :

- Le plus simple consiste à appeler `startActivity()` en lui passant l'intention – Android recherchera l'activité qui correspond le mieux et lui passera l'intention pour qu'elle la traite. Votre activité ne sera pas prévenue de la fin de l'activité fille.
- Vous pouvez appeler `startActivityForResult()` en lui passant l'intention et un identifiant (unique pour l'activité appelante). Android recherchera l'activité qui correspond le mieux et lui passera l'intention. Votre activité sera prévenue par la méthode de rappel `onActivityResult()` de la fin de l'activité fille (voir plus loin).
- Vous pouvez appeler `sendBroadcast()`. Dans ce cas, Android passera l'intention à tous les `BroadcastReceiver` enregistrés qui pourraient vouloir cette intention, pas uniquement à celui qui correspond le mieux.
- Vous pouvez appeler `sendOrderedBroadcast()`. Android passera alors l'intention à tous les `BroadcastReceiver` candidats, chacun leur tour – si l'un deux "consomme" l'intention, les autres candidats ne sont pas prévenus.

La plupart du temps, vous utiliserez `startActivity()` ou `startActivityForResult()` – les intentions diffusées sont plutôt lancées par le système Android lui-même.

Comme on l'a indiqué, vous pouvez implémenter la méthode de rappel `onActivityResult()` lorsque vous utilisez `startActivityForResult()`, afin d'être prévenu de la fin de l'activité fille. Cette méthode reçoit l'identifiant unique fourni à `startActivityForResult()` pour que vous puissiez savoir quelle est l'activité qui s'est terminée. Vous récupérez également :

- Le code résultat de l'activité fille qui a appelé `setResult()`. Généralement, ce code vaut `RESULT_OK` ou `RESULT_CANCELLED`, bien que vous puissiez créer vos propres codes (choisissez un entier à partir de la valeur `RESULT_FIRST_USER`).
- Un objet `String` optionnel contenant des données du résultat, comme une URL vers une ressource interne ou externe – une intention `ACTION_PICK` se sert généralement de cette chaîne pour renvoyer le contenu sélectionné.
- Un objet `Bundle` optionnel contenant des informations supplémentaires autres que le code résultat et la chaîne de données.

Pour mieux comprendre le lancement d'une activité paire, examinons le projet `Activities/Launch`. Le fichier de description XML est assez simple puisqu'il contient deux champs pour la latitude et la longitude, ainsi qu'un bouton :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
  <TableLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:stretchColumns="1,2"
  >
    <TableRow>
      <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingLeft="2dip"
        android:paddingRight="4dip"
        android:text="Situation :"
      />
      <EditText android:id="@+id/lat"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:cursorVisible="true"
        android:editable="true"
        android:singleLine="true"
        android:layout_weight="1"
      />
      <EditText android:id="@+id/lon"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:cursorVisible="true"
        android:editable="true"
        android:singleLine="true"
        android:layout_weight="1"
      />
```

```
    </TableRow>
  </TableLayout>
  <Button android:id="@+id/map"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Montre moi !"
  />
</LinearLayout>
```

L'`OnClickListener` du bouton prend la latitude et la longitude pour les intégrer dans une `Uri` de schéma `geo`, puis lance l'activité.

```
package com.commonsware.android.activities;
import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
public class LaunchDemo extends Activity {
  private EditText lat;
  private EditText lon;

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);

    Button btn=(Button)findViewById(R.id.map);
    lat=(EditText)findViewById(R.id.lat);
    lon=(EditText)findViewById(R.id.lon);

    btn.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        String _lat=lat.getText().toString();
        String _lon=lon.getText().toString();
        Uri uri=Uri.parse("geo:" + _lat + "," +_lon);

        startActivity(new Intent(Intent.ACTION_VIEW, uri));
      }
    });
  }
}
```

Comme on le voit à la Figure 24.1, cette activité ne montre pas grand-chose lorsqu'elle est lancée.

L'affichage devient plus intéressant si l'on entre un emplacement (38.8891 de latitude et –77.0492 de longitude, par exemple) et que l'on clique sur le bouton (voir Figure 24.2).

Figure 24.1
L'application `LaunchDemo`*, dans laquelle on a saisi un emplacement.*

Notez qu'il s'agit de l'activité de cartographie intégrée à Android : nous n'avons pas créé d'activité pour afficher cette carte.

Au Chapitre 34, nous verrons comment créer des cartes dans nos activités, au cas où l'on aurait besoin de plus de contrôle sur leur affichage.

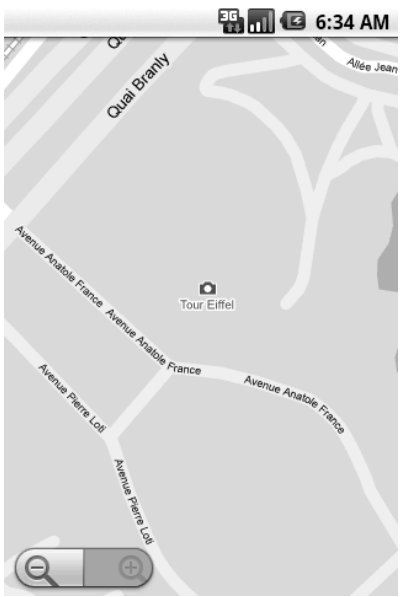

Figure 24.2
La carte lancée par `LaunchDemo`*, montrant l'emplacement de la Tour Eiffel à Paris.*

Navigation avec onglets

La navigation par onglet est l'une des principales fonctionnalités des navigateurs web actuels : grâce à elle, une même fenêtre peut afficher plusieurs pages réparties dans une série d'onglets. Sur un terminal mobile, cela a moins d'intérêt car on gaspillerait la précieuse surface de l'écran pour afficher les onglets eux-mêmes. Toutefois, pour les besoins de la démonstration, nous montrerons comment créer ce genre de navigateur, en utilisant `TabActivity` et les intentions.

Au Chapitre 10, nous avons vu qu'un onglet pouvait contenir une vue ou une activité. Dans ce dernier cas, vous devez fournir une intention qui lancera l'activité souhaitée ; le framework de gestion des onglets placera alors l'interface utilisateur de cette activité dans l'onglet.

Votre premier instinct pourrait être d'utiliser une `Uri` `http:` comme nous l'avions fait avec une `Uri` `geo:` dans l'exemple précédent :

```
Intent i=new Intent(Intent.ACTION_VIEW);
i.setData(Uri.parse("http://commonsware.com"));
```

Vous pourriez ainsi utiliser le navigateur intégré et disposer de toutes ses fonctionnalités.

Malheureusement, cela ne marche pas car, pour des raisons de sécurité, vous ne pouvez pas héberger les activités d'autres applications dans vos onglets – uniquement vos propres activités.

Nous allons donc dépoussiérer nos démonstrations de `WebView` du Chapitre 13 pour créer le projet `Activities/IntentTab`.

Voici le code source de l'activité principale, celle qui héberge le `TabView` :

```
public class IntentTabDemo extends TabActivity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    TabHost host=getTabHost();

    host.addTab(host.newTabSpec("un")
            .setIndicator("CW")
            .setContent(new Intent(this, CWBrowser.class)));
    host.addTab(host.newTabSpec("deux")
            .setIndicator("Android")
            .setContent(new Intent(this, AndroidBrowser.class)));
  }
}
```

Comme vous pouvez le constater, notre classe hérite de `TabActivity` : nous n'avons donc pas besoin de créer un fichier de description XML – `TabActivity` s'en occupe pour nous.

Nous nous contentons d'accéder au `TabHost` et de lui ajouter deux onglets, chacun précisant une intention qui fait directement référence à une autre classe. Ici, nos deux onglets hébergeront respectivement un `CWBrowser` et un `AndroidBrowser`.

Ces activités sont de simples variantes de nos précédents exemples de navigateurs :

```
public class CWBrowser extends Activity {
  WebView browser;

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);

    browser=new WebView(this);
    setContentView(browser);
    browser.loadUrl("http://commonsware.com");
  }
}
public class AndroidBrowser extends Activity {
  WebView browser;

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);

    browser=new WebView(this);
    setContentView(browser);
    browser.loadUrl("http://code.google.com/android");
  }
}
```

Tous les deux chargent simplement une URL différente dans le navigateur : la page d'accueil de CommonsWare dans l'un (voir Figure 24.3), celle d'Android dans l'autre (voir Figure 24.4). Le résultat montre l'aspect qu'aurait un navigateur à onglets avec Android.

L'utilisation de sous-classes différentes pour chaque page cible est plutôt onéreuse. À la place, nous aurions pu empaqueter l'URL pour qu'elle s'ouvre comme un "extra" dans une intention et utiliser cette intention pour créer une activité `BrowserTab` généraliste qui extrairait l'URL de cet "extra" afin de l'utiliser. Cette approche est laissée en exercice au lecteur.

Figure 24.3
L'application `IntentTabDemo` *montrant le premier onglet.*

Figure 24.4
L'application `IntentTabDemo` *montrant le second onglet.*