

Android - Fragments, et App Bars

Jérémy S. Cochoy

INRIA Paris-Saclay | jeremy.cochoy@gmail.com

Mars 2017

1 MVC

- Vue
- Modèle
- Contrôleur
- Intérêt ?

2 Fragments

- Kézako
- Créer un fragment
- L'utiliser dans une vue
- UI dynamiques
- Communication entre fragments

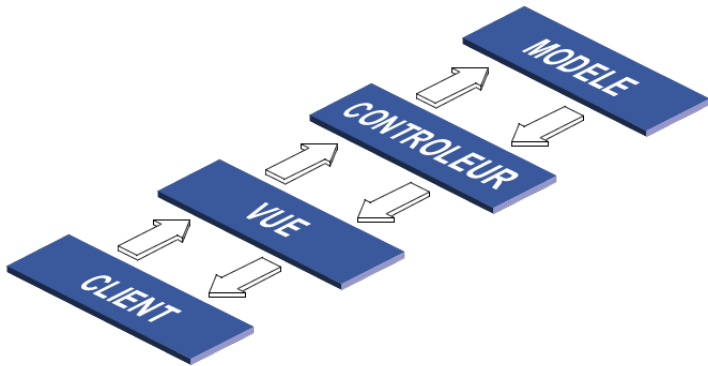
3 App Bar

- Ajouter une App Bar
- Configurer son AppBar
- Réagir aux actions

Votre nouveau livre de chevet.

<https://developer.android.com/guide/index.html>

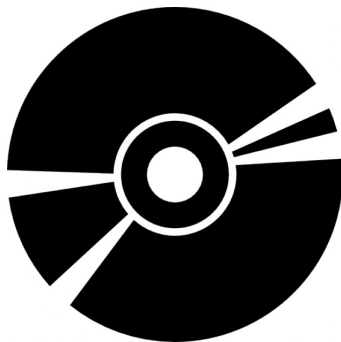
Le pattern MVC





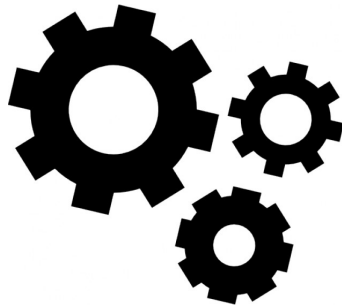
La vue

C'est l'interface qu'affiche votre application. Dans un projet Android, il s'agit du XML, ainsi que des composants que vous ajoutez dynamiquement via le code de votre activité.



Le modèle

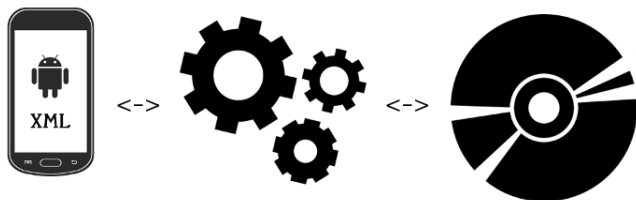
Une ou des classes qui gèrent l'accès à vos données. Elles peuvent provenir d'un accès distant, un serveur ftp, un fichier local, une base de données locale (SQLite) ou distante (MySQL)...



Le contrôleur

C'est essentiellement le code de vos activités : le code formate les données, et les envoie dans la vue. Le code gère les actions de l'utilisateur et réagit en conséquence (appel d'une nouvelle activité pour changer la vue, prise de photo, changement des images affichées...).

Résumé



Modèle MVC :

- Modèle (Model) : Ce qu'on veut afficher.
- Vue (View) : Comment l'afficher.
- Contrôleur (Controller) : Formate les données pour les afficher et gère les événements tels que les entrées utilisateur.

Avantages du MVC ?

Indépendance

Les 3 blocs sont indépendants et communiquent via une interface claire et précise.

Substitution du modèle

Il est possible de remplacer un modèle "fichier local" par un modèle "base de données distante", avec un minimum de modification du code, et sans modification de la vue.

Substitution de la vue

En modifiant uniquement l'XML, on peut revoir le design de l'interface.

Dark side

Attention aux fausses bonnes idées...

Avantages du MVC ?

Indépendance

Les 3 blocs sont indépendants et communiquent via une interface claire et précise.

Substitution du modèle

Il est possible de remplacer un modèle "fichier local" par un modèle "base de données distante", avec un minimum de modification du code, et sans modification de la vue.

Substitution de la vue

En modifiant uniquement l'XML, on peut revoir le design de l'interface.

Dark side

Attention aux fausses bonnes idées...

Avantages du MVC ?

Indépendance

Les 3 blocs sont indépendants et communiquent via une interface claire et précise.

Substitution du modèle

Il est possible de remplacer un modèle "fichier local" par un modèle "base de données distante", avec un minimum de modification du code, et sans modification de la vue.

Substitution de la vue

En modifiant uniquement l'XML, on peut revoir le design de l'interface.

Dark side

Attention aux fausses bonnes idées...

Avantages du MVC ?

Indépendance

Les 3 blocs sont indépendants et communiquent via une interface claire et précise.

Substitution du modèle

Il est possible de remplacer un modèle "fichier local" par un modèle "base de données distante", avec un minimum de modification du code, et sans modification de la vue.

Substitution de la vue

En modifiant uniquement l'XML, on peut revoir le design de l'interface.

Dark side

Attention aux fausses bonnes idées...

Les fragments

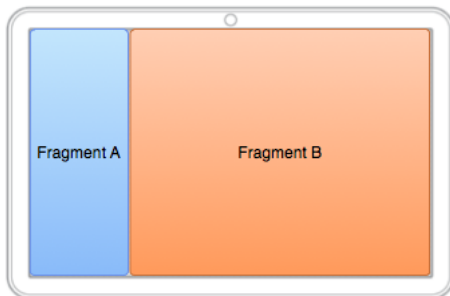


FragmentActivity

Fragments

Qu'est-ce qu'un fragment ?

C'est une partie modulaire d'une activité.



Fragments

Pourquoi utiliser des fragments ?

Pour permettre à nos applications de s'adapter aux supports physiques.



Créer un fragment

Comment faire ?

Comme une Activité.

Une subtilité :

C'est la méthode `onCreateView()` que l'on redéfinit, et non `onCreate()`.

Un fragment vide

Code pour créer un fragment

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.ViewGroup;

public class ArticleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container,
        Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.article_view,
            container, false);
    }
}
```

Ajouter un fragment à son activité

Ajouter un fragment via XML

On peut ajouter du code XML dans `layout-large`, et un autre dans `layout`.



Ajouter un fragment via l'API

A l'exécution, on peut déterminer la résolution de l'écran et remodeler dynamiquement l'interface.



Ajouter un fragment via XML

Code pour ajouter deux fragments

```
<LinearLayout ...>
  <fragment android:name="com.example.android.fragments.
    HeadlinesFragment"
    android:id="@+id/headlines_fragment"
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="match_parent" />

  <fragment android:name="com.example.android.fragments.
    ArticleFragment"
    android:id="@+id/article_fragment"
    android:layout_weight="2"
    android:layout_width="0dp"
    android:layout_height="match_parent" />

</LinearLayout>
```

Ajouter un fragment à l'exécution

FragmentManager

Il nous faut un gestionnaire de fragment (`FragmentManager`) pour construire un `FragmentManager` qui, lui, pourra ajouter / supprimer / remplacer des fragments dans notre activité.

Il faut un container

Il faut un container de type `View` pour y placer nos fragments. Un simple

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

convient.

Ajouter un fragment à l'exécution

FragmentManager

Il nous faut un gestionnaire de fragment (`FragmentManager`) pour construire un `FragmentManager` qui, lui, pourra ajouter / supprimer / remplacer des fragments dans notre activité.

Il faut un container

Il faut un container de type `View` pour y placer nos fragments. Un simple

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

convient.

Ajouter un fragment à l'exécution

Dans onCreate, on ajoute un fragment de la façon suivante :

```
// On verifie qu'on trouve bien notre container.
if (findViewById(R.id.fragment_container) != null) {
// On verifie que c'est le premier lancement de l'activite.
if (savedInstanceState != null) {
    return;
}
// On cree le fragment a placer.
HeadlinesFragment firstFragment = new HeadlinesFragment();
// On transmet d'éventuels arguments.
firstFragment.setArguments(getIntent().getExtras());
// On ajoute le fragment au FrameLayout 'fragment_container'.
getSupportFragmentManager().beginTransaction()
    .add(R.id.fragment_container, firstFragment).commit();
}
```

Ajouter un fragment à l'exécution

Détail de la transaction :

```
//Recuperation du manager
FragmentManager manager = getSupportFragmentManager();

//Creation de la transaction
FragmentManager transaction = manager.beginTransaction();
//On ajoute un/des fragments
transaction.add(R.id.fragment_container, firstFragment)
//On 'commit' les operations
transaction.commit();
```

Parce que son ajout est dynamique, ce fragment pourra être retiré ou remplacé.

Remplacer un fragment

Comment remplacer un fragment ?

Avec la méthode `replace()` au lieu de `add()`.

Récupérer le fragment remplacé

Souvent, on souhaite permettre à l'utilisateur "d'annuler" la transaction et récupérer le fragment précédent. Pour ça, il suffit d'appeler `addToBackStack()` durant la transaction.

Vie d'un fragment

Si un fragment est poussé sur la pile de retour, alors il est *stoppé*. Après un retour en arrière, il passera dans l'état *redémarré*. Sinon, il est *détruit*.

Remplacer un fragment

Comment remplacer un fragment ?

Avec la méthode `replace()` au lieu de `add()`.

Récupérer le fragment remplacé

Souvent, on souhaite permettre à l'utilisateur "d'annuler" la transaction et récupérer le fragment précédent. Pour ça, il suffit d'appeler `addToBackStack()` durant la transaction.

Vie d'un fragment

Si un fragment est poussé sur la pile de retour, alors il est *stoppé*. Après un retour en arrière, il passera dans l'état *redémarré*. Sinon, il est *détruit*.

Remplacer un fragment

Comment remplacer un fragment ?

Avec la méthode `replace()` au lieu de `add()`.

Récupérer le fragment remplacé

Souvent, on souhaite permettre à l'utilisateur "d'annuler" la transaction et récupérer le fragment précédent. Pour ça, il suffit d'appeler `addToBackStack()` durant la transaction.

Vie d'un fragment

Si un fragment est poussé sur la pile de retour, alors il est *stoppé*. Après un retour en arrière, il passera dans l'état *redémarré*. Sinon, il est *détruit*.

Remplacer un fragment

Création d'un nouveau fragment et de ses arguments :

```
//Le fragment
ArticleFragment newFragment = new ArticleFragment();

//Les arguments
Bundle args = new Bundle();
args.putInt(ArticleFragment.ARG_POSITION, position);
newFragment.setArguments(args);
```

Remplacer un fragment

Création d'un nouveau fragment

```
// Commence une transaction
FragmentManager transaction = getSupportFragmentManager()
    .beginTransaction();
// Remplace le fragment
transaction.replace(R.id.fragment_container, newFragment);
// Conserve le fragment precedent
transaction.addToBackStack(null);
// Effectue la transaction
transaction.commit();
```

addToBackStack()

`addToBackStack()` prend en argument une chaîne de caractères optionnelle qui permet de donner un identifiant unique à la transaction, pour effectuer des opérations avancées.

Communiquer entre fragments

Comment ?

La communication se fait via l'activité en lui imposant d'implémenter une *interface*.

On récupère un pointeur vers l'activité que l'on downcast vers notre interface. Il y a donc un certain nombre de précautions à prendre.

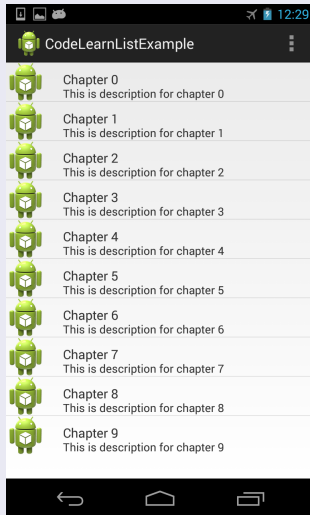
Communiquer entre fragments

Comment ?

La communication se fait via l'activité en lui imposant d'implémenter une *interface*.

On récupère un pointeur vers l'activité que l'on downcast vers notre interface. Il y a donc un certain nombre de précautions à prendre.

Disons que notre fragment hérite de ListFragment...



Forcer une interface pour notre activité

On définit notre interface ...

```
// Notre fragment
public class HeadlinesFragment extends ListFragment {
    // Variable global qui contiendra un pointeur vers l'
    Activity
    OnHeadlineSelectedListener mCallback;

    // L'Activity contenant le fragment devra implementer l'
    interface :
    public interface OnHeadlineSelectedListener {
        public void onArticleSelected(int position);
    }

    ...
}
```


Forcer une interface pour notre activité

... puis on récupère un pointeur vers l'activité, et on s'assure qu'elle implémente bien notre interface.

```
@Override
public void onAttach(Activity activity) {
    super.onAttach(activity);

    // Pour être sûr de la présence d'une implémentation,
    // on effectue une conversion explicite vers
    // OnHeadlineSelectedListener.
    try {
        mCallback = (OnHeadlineSelectedListener) activity;
    } catch (ClassCastException e) {
        throw new ClassCastException(activity.toString()
            + " must implement OnHeadlineSelectedListener");
    }
}
```

Exemple de communication

Un exemple d'utilisation de l'interface :

```
@Override
    public void onItemClick(ListView l, View v, int
        position, long id) {
        // Appelle la fonction de l'Activity.
        mCallback.onArticleSelected(position);
    }
```

Coté activité...

Implémentation de l'interface

```
public static class MainActivity extends Activity
    implements HeadlinesFragment.
        OnHeadlineSelectedListener{
    ...

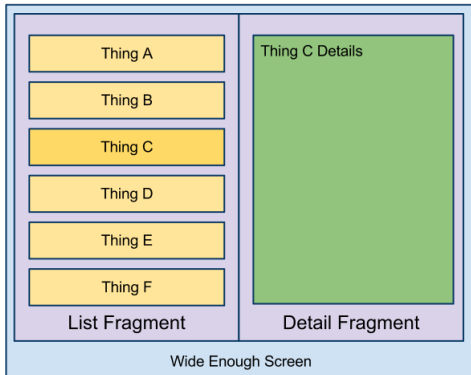
    public void onArticleSelected(int position) {
        // L'utilisateur a choisi un item dans la liste
        // On effectue le nécessaire pour afficher l'article
        // correspondant.
    }
}
```

Pour gérer un mode tablette et un mode mobile...

L'activité qui parlait à l'oreille des fragments...

```
public void onArticleSelected(int position) {  
    //On cherche notre fragment  
    ArticleFragment articleFrag = (ArticleFragment)  
        getSupportFragmentManager().findFragmentById(R.id.  
            article_fragment);  
  
    if (articleFrag != null) {  
        // On a bien notre fragment, donc on change l'affichage  
        articleFrag.updateArticleView(position);  
    } else {  
        // Suite au prochain slide...  
    }  
}
```

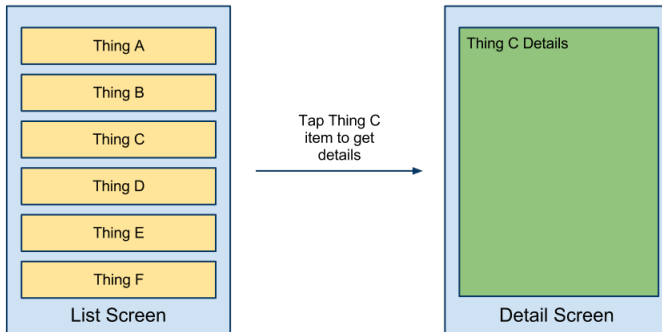
La configuration correspondante



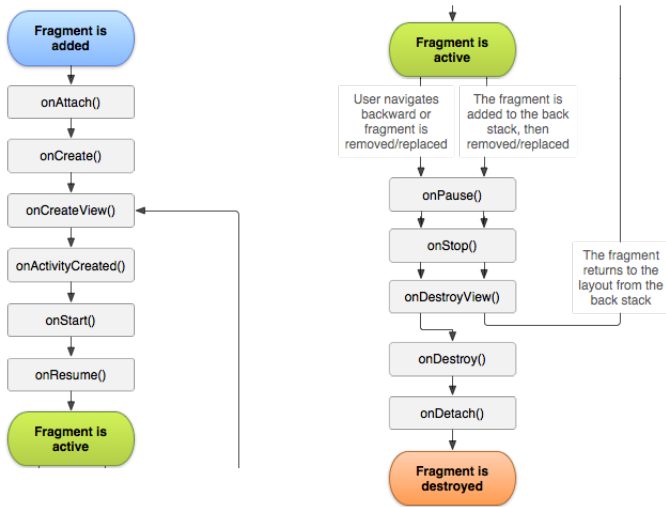
Cas mono-screen...

```
} else {  
    // On est en mode 'un seul ecran'.  
  
    // On construit le nouveau fragment  
    ArticleFragment newFragment = new ArticleFragment();  
    Bundle args = new Bundle();  
    args.putInt(ArticleFragment.ARG_POSITION, position);  
    newFragment.setArguments(args);  
  
    //Et on effectue la transaction...  
    FragmentTransaction transaction = getSupportFragmentManager  
        ().beginTransaction();  
  
    transaction.replace(R.id.fragment_container, newFragment);  
    transaction.addToBackStack(null);  
  
    //On realise la transaction  
    transaction.commit();  
}
```

La configuration correspondante



Cycle de vie d'un fragment



Qu'est-ce qu'une barre d'application ?



Comment ajouter une Toolbar ?

Il faut le support de la fonctionnalité.

Pour ça, il vous faut éventuellement installer la bibliothèque v7 appcompat, si ce n'est pas déjà fait.

Et hériter de la bonne classe Activity.

```
public class MyActivity extends AppCompatActivity {  
    // ...  
}
```

Comment ajouter une Toolbar ?

Il faut le support de la fonctionnalité.

Pour ça, il vous faut éventuellement installer la bibliothèque v7 appcompat, si ce n'est pas déjà fait.

Et hériter de la bonne classe Activity.

```
public class MyActivity extends AppCompatActivity {  
    // ...  
}
```

Comment ajouter une Toolbar ?

Il faut le support de la fonctionnalité.

Pour ça, il vous faut éventuellement installer la bibliothèque v7 appcompat, si ce n'est pas déjà fait.

Et hériter de la bonne classe Activity.

```
public class MyActivity extends AppCompatActivity {  
    // ...  
}
```

Et choisissez le bon thème.

```
<application  
    android:theme="@style/Theme.AppCompat.Light.NoActionBar"  
>
```

Comment ajouter une Toolbar ?

Il faut le support de la fonctionnalité.

Pour ça, il vous faut éventuellement installer la bibliothèque v7 appcompat, si ce n'est pas déjà fait.

Et hériter de la bonne classe Activity.

```
public class MyActivity extends AppCompatActivity {  
    // ...  
}
```

Et choisissez le bon thème.

```
<application  
    android:theme="@style/Theme.AppCompat.Light.NoActionBar"  
>
```

Comment ajouter une Toolbar ?

On ajoute la Toolbar à la vue :

```
<android.support.v7.widget.Toolbar
    android:id="@+id/my_toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:elevation="4dp"
    android:theme="@style/ThemeOverlay.AppCompat.ActionBar"
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light"/>
```

Comme c'est une barre d'application, on veut la positionner en haut de l'application.

Comment ajouter une Toolbar ?

On lie la Toolbar au niveau de l'activité :

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_my);
    Toolbar myToolbar = (Toolbar) findViewById(R.id.
        my_toolbar);
    setSupportActionBar(myToolbar);
}
```

Contenu de la barre :

Par défaut, on y trouve le titre de l'application et un menu déroulant avec pour seul élément "Settings".

Personnalisez votre AppBar

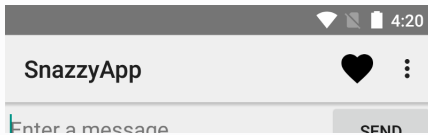
Une vue XML

Les boutons et autres objets sont stockés dans une vue, dans `res/menu`.

Ajouter un bouton

Pour chaque élément à ajouter, on place un élément `item`.

Ce que l'on veut :



Personnalisez votre AppBar

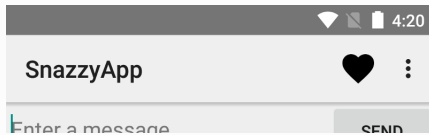
Une vue XML

Les boutons et autres objets sont stockés dans une vue, dans `res/menu`.

Ajouter un bouton

Pour chaque élément à ajouter, on place un élément `item`.

Ce que l'on veut :



Personnalisez votre AppBar

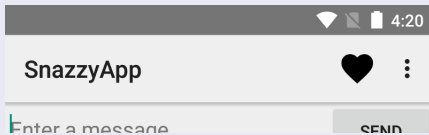
Une vue XML

Les boutons et autres objets sont stockés dans une vue, dans `res/menu`.

Ajouter un bouton

Pour chaque élément à ajouter, on place un élément `item`.

Ce que l'on veut :



Personnalisez votre AppBar

Exemple d'AppBar

```
<menu xmlns:android="http://schemas.android.com/apk/res/
  android" >
  <!-- Si possible, "Mark Favorite", doit apparaitre comme
    un bouton. -->
  <item
    android:id="@+id/action_favorite"
    android:icon="@drawable/ic_favorite_black_48dp"
    android:title="@string/action_favorite"
    app:showAsAction="ifRoom"/>

  <!-- Configuration doit Toujours etre dans le menu
    deroulant. -->
  <item android:id="@+id/action_settings"
    android:title="@string/action_settings"
    app:showAsAction="never"/>
</menu>
```

Comment réagir à un clic ?

Au moment où l'utilisateur clique :

La méthode `onOptionsItemSelected()` est appelé, avec en argument un `MenuItem` correspondant à l'item cliqué. La méthode `MenuItem.getItemId()` permet de récupérer l'ID de l'élément.

Retrouvons notre exemple :

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_settings:
            // Lancer l'Activity "configuration".
            return true;

        case R.id.action_favorite:
            // Ajouter l'article courant aux favoris
            return true;

        default:
            // Action non reconnue, dans le doute on laisse
            // faire la classe parente.
            return super.onOptionsItemSelected(item);
    }
}
```

Pour me contacter : jeremy.cochoy@gmail.com, merci et à bientôt.

