

Interfaces modulaires et dynamiques

Chantal Keller

9 octobre 2015

Plan

- 1 Divers
- 2 Interfaces modulaires et dynamiques
- 3 Utilisation dans une activité
- 4 Interaction
- 5 À vous

Erratum TD5

Question 3 :

- erreur avec l'API 23 :

A WebView method was called on thread 'JavaBridge'. All WebView methods must be called on the same thread.

- warning avec l'API 15

Explication

- Le thread principal gère l'interface, les autres threads gèrent les opérations coûteuses (sans accès à l'interface)
- `webview.loadUrl(url)`, `webview.reload()`, `webview.goBack()`... dans le thread principal
- Mais les méthodes Android appelées par JavaScript s'exécutent dans un autre thread (`JavaBridge`)

Conclusion

Conclusion :

- `webView.loadUrl(url)` ne peut être utilisé directement par une méthode exposée à JavaScript
- il faut passer l'appel au thread principal

Rappelez-vous `AsyncTask` :

- `doInBackground` exécutée dans un autre thread
- `onPreExecute`, `onProgressUpdate` et `onPostExecute` exécutées dans le thread principal

↪ directement fourni par la classe `AsyncTask`

Solution

Avec des threads plus généraux :

- la classe `View` fournit la méthode `boolean post (Runnable action)`
- but : envoyer une action au thread principal

Dans notre cas :

```
@JavascriptInterface
public void loadUserUrl() {
    ...
    webview.post(new Runnable() {
        public void run() {
            webview.loadUrl(url);
        }
    });
}
```

↪ pas fondamental, mais savoir que ça existe en cas de besoin

Rendu du TD5

Ergonomie :

- Boutons et zones de texte sur la page web (utiliser `RelativeLayout` et les poids)
- Bouton "avant" à gauche du bouton "arrière"

Autre :

- Ajout de `http://` avant l'url

Plan

- 1 Divers
- 2 Interfaces modulaires et dynamiques
- 3 Utilisation dans une activité
- 4 Interaction
- 5 À vous

Rappelez-vous...

Rappels :

- plusieurs activités au sein d'une même application
- chaque activité a son interface

Aujourd'hui :

- interface d'une même activité découpée en blocs
- blocs : *fragments*

Buts

Fragments :

- partager le code des interfaces
- créer des interfaces dynamiques :
 - attachement/détachement de fragments
 - remplacement d'un fragment par un autre

Fragment

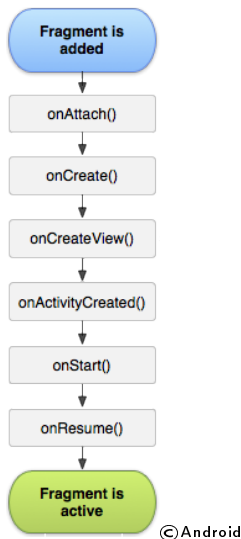
Un fragment :

- est une classe Java qui hérite de `Fragment`
- a sa propre interface XML (même syntaxe que les interfaces des activités (sauf `onClick`, voir TD))

Cycle de vie :

- ensemble de méthodes appelées lorsque le fragment est attaché et détaché

Attachement d'un fragment



`onCreateView()`

- méthode appelée lorsque l'interface du fragment est affichée
- c'est ici que l'on récupère les objets de l'interface

Détachement d'un fragment



Plan

- 1 Divers
- 2 Interfaces modulaires et dynamiques
- 3 Utilisation dans une activité
- 4 Interaction
- 5 À vous

Sans dynamisme

Dans l'interface XML de l'activité :

```
<fragment
    android:id="@+id/fragment"
    tools:layout="@layout/mon_fragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

↔ impossible de détacher ou de remplacer le fragment

Dynamiquement

Dans l'interface XML de l'activité :

```
<FrameLayout
    android:id="@+id/fragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Dans le code Java de l'activité, à l'endroit souhaité :

```
MonFragment frag = new MonFragment();
FragmentManager transaction = getFragmentManager().beginTransaction();
transaction.add(R.id.fragment, frag);
transaction.commit();
```

Le gestionnaire de fragments :

- renvoyé par `getFragmentManager()` (méthode de la classe `Activity`)
- permet d'attacher, de détacher ou de remplacer des fragments, grâce à des *transactions*

Transactions

3 étapes :

- Début : `FragmentTransaction transaction = getFragmentManager().beginTransaction();`
- Opérations (autant que l'on souhaite) :
 - attachement : `transaction.add(R.id.fragment, newFrag);`
 - détachement : `transaction.remove(oldFrag);`
 - remplacement : `transaction.replace(R.id.fragment, newFrag);`
- Fin (ici que ça prend effet) : `transaction.commit();`

Remarques :

- l'appel à `getFragmentManager()` peut être fait une fois pour toutes, et le résultat mis dans une variable
- possible de faire des animations (voir TD)

Plan

- 1 Divers
- 2 Interfaces modulaires et dynamiques
- 3 Utilisation dans une activité
- 4 Interaction
- 5 À vous

Le fragment veut accéder aux objets de l'activité

`getActivity()`

- méthode de la classe `Fragment`
- renvoie l'activité à laquelle le fragment est attaché
- on a ensuite accès à toutes les méthodes de la classe `Activity`, notamment `findViewById`

L'activité veut accéder aux objets du fragment

Pas aussi direct, mais :

- l'activité connaît le fragment : `frag`
- elle peut appeler les méthodes qu'elle souhaite dessus
- ex :

- dans l'activité : `frag.changerAffichage("Bonjour !")`
- dans le fragment :

```
public void changerAffichage(String s) {  
    affichage.setText(s);  
}
```

Plan

- 1 Divers
- 2 Interfaces modulaires et dynamiques
- 3 Utilisation dans une activité
- 4 Interaction
- 5 À vous

TD

Interfaces dynamiques :

- “Click the button !”
- Plusieurs fragments au sein d’une même activité