Instituto Tecnológico Superior del Sur de Guanajuato

“Fragmentos en Android”

Programación Móvil I

Elaborado por:

Isabel Contreras Rico

No. Control

S17120218

Catedrático:

Ing. Gustavo Iván Vega Olvera

Fecha:

10/10/2020

Índice

Introducción

Un `Fragment` representa un comportamiento o parte de una interfaz de usuario en una `FragmentActivity`. Se puede combinar varios fragmentos en una misma actividad, así como reutilizar un fragmento en distintas actividades. Puede verse como una sección modular de una actividad, con un ciclo de vida propio.

Un fragmento debe contenerse en una actividad, su ciclo de vida se ve afectado por el de la actividad padre. Cuando se agrega un fragmento como parte de la presentación, este se encuentra en `ViewGroup`, dentro de la jerarquía de vistas de la actividad.

Filosofía de diseño

Al dividir el diseño de una actividad en fragmentos, es posible modificar el aspecto durante el tiempo de ejecución y emplear una pila de actividades para conservar los cambios.

Los fragmentos deben diseñarse como componentes modulares y reutilizables, evitando la manipulación directa de un fragmento desde otro. Una ventaja de un diseño modular es la practicidad para cambiar las combinaciones de fragmentos en diferentes pantallas.

Cómo crear un fragmento

Para crear un fragmento, es necesario crear una subclase `Fragment`. Esta clase tiene un código semejante al de un `Activity`, contienen métodos de devolución de llamada similares. A continuación se listan los métodos del ciclo de vida que deben implementarse.

`onCreate()`

Se manda llamar cuando se crea el fragmento. Aquí se inicializan componentes esenciales del fragmento que se deseen conservar cuando este se pause o detenga y posteriormente se reanude.

`onCreateView()`

Se ejecuta cuando el fragmento requiere diseñar su interfaz de usuario por primera vez. Debe mostrarse un View desde este método, el cual será la raíz del diseño del fragmente. Este puede ser nulo si el fragmento no ofrece IU.

`onPause()`

Con el primer indicador de que el usuario abandona el fragmento (no necesariamente destruyendo), se manda llamar este método, donde se confirma qué cambios deben conservarse más allá de la sesión del usuario actual.

Hay otros métodos de devolución de llamada que también se emplean para manipular el ciclo de vida del fragmento.

Existen también subclases que pueden extenderse en lugar de `Fragmen`, por ejemplo:

`DialogFragment`

Muestra un dialogo flotante. Es una alternativa para el uso de métodos del asistente de diálogos en una clase `Activity`.

`ListFragment`

Muestra una lista de elementos administrados por un adaptador, al igual que `ListActivity`. Proporciona métodos para controlar una vista de lista, como `onListItemClick()` para manipular eventos de clic.

`PreferenceFragmentCompat`

Muestra una jerarquía de objetos `Preference` en forma de lista. Ese objeto es empleado para crear una pantalla de configuración de la app.

Agregar una interfaz de usuario

Como se ha mencionado anteriormente, un fragmento le aporta su propio diseño a una actividad. Para implementarle un diseño al fragmento, es necesario retornar un `View` desde el método `onCreateView()`. Este método proporciona un objeto LayoutInflater que puede definir un diseño establecido en XML.

A continuación se muestra un ejemplo de lo anterior.

```
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
            ViewGroup container, Bundle savedInstanceState) {
        // Inflar el layout para este fragmento
        return inflater.inflate(R.layout.example_fragment,
                                container, false);
    }
}
```

`Layout.example_fragment` es el XML que define el diseño. Este es traído por el inflador y retornado como `View` del fragmento.

El `container` que pasa a `onCreateView()` es el `ViewGroup` principal (es el diseño de la actividad) en el cual se insertará el diseño del fragmento. El parámetro

`savedInstanceState`, es `Bundle` un que proporciona datos acerca de la instancia previa del fragmento si este se está reanudando. El método `inflate()` adopta tres argumentos:

- ID del recurso de diseño a inflar (en este caso `R.layout.example_fragment`)
- El `ViewGroup` que será el elemento principal del diseño inflado.
- Un valor booleano, que indica si se debe anexar el diseño inflado al `ViewGroup` durante el agrandamiento. Se indica false, en caso de que el sistema ya esté insertando el diseño aumentado al `container`. En caso de señalar true, se crea un grupo de vistas redundante en el diseño final.

Agregar un fragmento a una actividad

Existen dos maneras de agregar un fragmento a una actividad:

- *Declarar el fragmento en el archivo de diseño de la actividad.*

  En este caso se pueden especificar propiedades de diseño para el fragmento como si fuera una vista.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.ArticleListFragment"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
            android:id="@+id/viewer"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

  El atributo `android:name` de `<fragment>` especifica la clase Fragment para crear una instancia en el diseño. Al crear el diseño, se crea una instancia del fragmento y se ejecuta su evento `onCreateView(),` para recuperar el diseño. El sistema inserta el objeto `View` del fragmento donde está el elemento `<fragment>`.

- *Guardar el fragmento de forma programática en un **ViewGroup** existente.*

  Mientras se está ejecutando la actividad, se pueden agregar fragmentos al diseño. Solo necesita especificarse un `ViewGroup` donde irá el fragmento.

Para realizar transacciones de fragmentos, debe usarse la API de `FragmentTransaction`, como se muestra a continuación, en la primer imagen, se instancia un objeto de `FragmentTansaction` y `FragmentManager`. Posteriormente, en la otra figura, se agrega un fragmento en la vista especificando el fragmento y la vista donde este se insertará. El primer parámetro pasado al método `add()` representa el `ViewGroup`, y el segundo parámetro es el fragmento a añadir.

```
FragmentManager fragmentManager = getSupportFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

```
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

Cuando se finalicen los cambios en el `fragmentTransaction` debe ejecutarse el método `commit()`, para aplicar los cambios.

Administrar fragmentos

Para controlar los fragmentos de una actividad, debe emplearse `FragmentManager`. Para conseguirlo, debe llamarse `getSupportFragmentManager()`. Este administrador puede realizar las siguientes tareas:

- `findFragmentById()` o `findFragmentByTag()`, obtiene fragmentos existentes en la actividad.
- `popBackStack()`, activa fragemntos de la pila de retroceso, simula ‘atras’ del usuario.
- `addOnBackStackChangedListener()`, registra un escucha para los cambios producidos en la pila de retroceso

Transacciones de fragmentos

Cada conjunto de cambios que se confirman en una actividad reciben la denominación de transacción. Las transacciones se pueden almacenar en la pila de actividades de la Actividad, permitiendo al usuario navegar hacia atrás por los cambios realizados en el fragmento.

Cada transacción se compone de un conjunto de cambios que se desean realizar al mismo tiempo. Estos cambios se pueden realizar a través de los métodos `add()`, `remove()`, y `replace()`. Finalmente `commit()` guarda los cambios. Es conveniente llamar al método `addToBackStack()` antes de finalizar.

```
// Create new fragment and transaction
Fragment newFragment = new ExampleFragment();
FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();

// Replace whatever is in the fragment_container view with this fragment,
// and add the transaction to the back stack
transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);

// Commit the transaction
transaction.commit();
```

La imagen anterior muestra un ejemplo de cómo se realiza un reemplazo de fragmento por otro, conservando a su vez el fragmento anterior en la pila de actividades.

Si dentro de una transacción se realizan múltiples acciones, y luego se manda llamar a la pila, y finalmente al commit(), todos los cambios realizados se revierten juntos al extraer la transacción de la pila.

Comunicación con la actividad

Una instancia determinada de un fragmento está vinculada directamente con la actividad que la contiene. El fragmento puede acceder a la instancia `FragmentActivity` a través del método `getActivity()` y realizar tareas.

```
View listView = getActivity().findViewById(R.id.list);
```

De igual modo, la actividad puede llamar a métodos del fragmento mediante una referencia a `Fragment` desde `FragmentManager`, empleando `findFragmentById()` o `findFragmentByTag()`.

```
ExampleFragment fragment = (ExampleFragment) getSupportFragmentManager().
findFragmentById(R.id.example_fragment);
```

Devoluciones de llamadas a eventos a la actividad

Para compartir datos desde el fragmento con la actividad, es preciso usar un `ViewModel` compartido. Si lo que se requiere es propagar eventos, puede definirse una interfaz de devolución de llamada dentro de un fragmento, y forzar a la actividad a que lo implemente, a través del método `onAttach()` del fragmento, creando una instancia de `OnArticleSelectedListener` ordenando la `Activity` que se pasa a `onAttach()`.

Cuando la actividad recibe una devolución de llamada, puede compartir la información con otros fragmentos.

Agregar artículos a la barra de app

Los fragmentos pueden agregar elementos al menú de opciones de la actividad, implementando `onCreateOptionsMenu()`, para lograrlo, durante el `onCreate()` del fragmento debe agregársele `setHasOptionsMenu()` para indicar que se intenta agregar elementos.

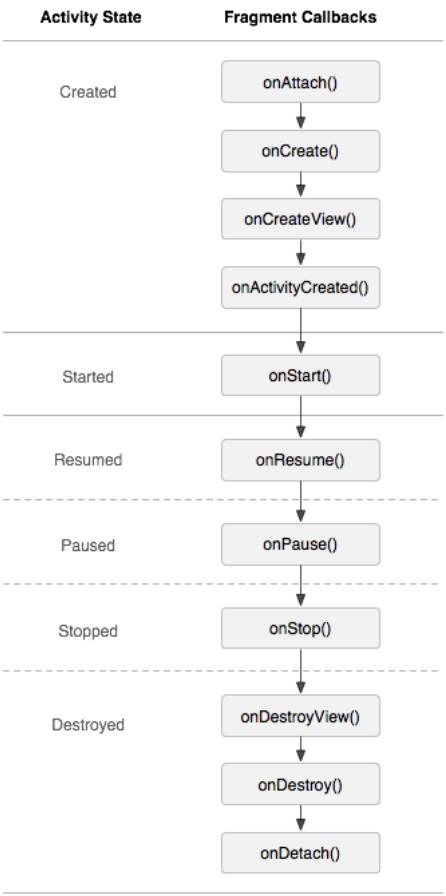
Cuando se hayan agregado opciones en el menú, y alguna de estas sea seleccionada, el fragmento recibirá devoluciones de llamadas a `onOptionsItemSelected().`

También puede registrarse una vista en el diseño del fragmento que ofrezca un menú contextual llamando a `registerForContextMenu()`, cuando el usuario abra el menú contextual, se recibirá una llamada a `onCreateContextMenu()`, y cuando seleccione un elemento, se recibirá una llamada a `onContextItemSelected()`.

Control del ciclo de vida de un fragmento

Un fragmento puede tener tres estados:

- *Reanudado*. El fragmento es visible en la actividad que lo ejecuta.
- *Pausado.* Otra actividad está en primer plano y tiene el foco. La actividad donde reside el fragmento está visible y en segundo plano.
- *Detenido.* El fragmento no está visible, o se detuvo la actividad anfitriona. Aún está activo.



Coordinación con el ciclo de vida de la actividad

El ciclo de vida del fragmento se ve afectado directamente por la actividad que lo contiene. Las devoluciones de llamada del ciclo de vida de la actividad generan una devolución semejante para el fragmento. Por ejemplo, si la actividad recibe un onPause(), cada fragmento lo recibirá también. La figura de la derecha relaciona las llamadas del ciclo de vida del fragmento con el ciclo de vida de una actividad.