Notificaciones y AppWidgets

Índice

Los servicios en segundo plano de Android desempeñan funciones que requieren poca interacción con el usuario. Aún así lo normal es que haya que informar de sucesos o de progreso. Un mecanismo de interfaz gráfica específico de Android son las notificaciones. Otro son los AppWidgets, o widgets de la pantalla de inicio, que a menudo realizan la actualización de sus vistas desde servicios en segundo plano.

1. Notificaciones

El típico mecanismo de comunicación con el usuario que los Servicios utilizan son las `Notification`. Se trata de un mecanismo mínimamente intrusivo que no roba el foco a la aplicación actual y que permanece en una lista de notificaciones en la parte superior de la pantalla, que el usuario puede desplegar cuando le convenga.

Desplegando la barra de notificaciones

Para mostrar y ocultar notificaciones hay que obtener de los servicios del sistema el `NotificationManager`. Su método `notify(int, Notification)` muestra la notificación asociada a determinado identificador.

```
Notification notification;
NotificationManager notificationManager;
notificationManager = (NotificationManager)getSystemService(
        Context.NOTIFICATION_SERVICE);
notification = new Notification(R.drawable.icon,
        "Mensaje evento", System.currentTimeMillis());
notificationManager.notify(1, notification);
```

El identificador sirve para actualizar la notificación en un futuro (con un nuevo aviso de notificación al usuario). Si se necesita añadir una notificación más, manteniendo la

anterior, hay que indicar un nuevo ID.

Para actualizar la información de un objeto `Notification` ya creado, se utiliza el método

```
notification.setLatestEventInfo(getApplicationContext(),
                "Texto", contentIntent);
```

donde `contentIntent` es un `Intent` para abrir la actividad a la cuál se desea acceder al pulsar la notificación. Es típico usar las notificaciones para abrir la actividad que nos permita reconfigurar o parar el servicio. También es típico que al pulsar sobre la notificación y abrirse una actividad, la notificación desaparezca. Este cierre de la notificación lo podemos implementar en el método `onResume()` de la actividad:

```
@Override
protected void onResume() {
        super.onResume();
        notificationManager.cancel(MiTarea.NOTIF_ID);
}
```

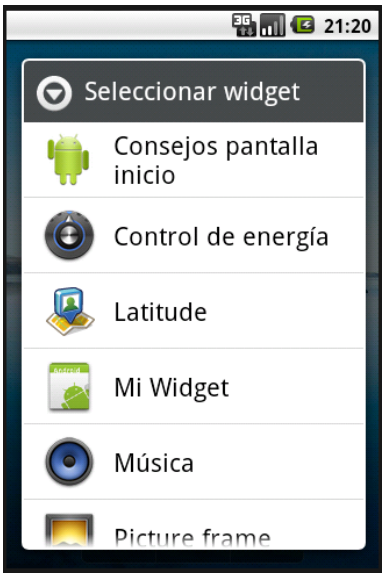
A continuación se muestra un ejemplo completo de notificaciones usadas por una tarea `AsyncTask`, que sería fácilmente integrable con un `Service`. (Sólo haría falta crear una nueva `MiTarea` en `Service.onCreate()`, arrancarla con `miTarea.execute()` desde `Service.onStartCommand(...)` y detenerla con `miTarea.cancel()` desde `Service.onDestroy()`).

```
private class MiTarea
        extends AsyncTask<String, String, String>{
    public static final int NOTIF1_ID = 1;
    Notification notification;
    NotificationManager notificationManager;

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        notificationManager =
                (NotificationManager)getSystemService(
                        Context.NOTIFICATION_SERVICE);
        notification = new Notification(R.drawable.icon,
                "Mensaje evento", System.currentTimeMillis());
    }

    @Override
    protected String doInBackground(String... params) {
        while(condicionSeguirEjecutando){
            if(condicionEvento){
                publishProgress("Información del evento");
            }
        }
        return null;
    }

    @Override
    protected void onProgressUpdate(String... values) {
        Intent notificationIntent = new Intent(
```

```
                        getApplicationContext(),
MiActividadPrincipal.class);
                PendingIntent contentIntent = PendingIntent.getActivity(
                        getApplicationContext(), 0, notificationIntent,
0);
                notification.setLatestEventInfo(getApplicationContext(),
                        values[0], contentIntent);
                notificationManager.notify(NOTIF_ID, notification);
        }
}
```

2. AppWidgets

Los widgets, que desde el punto de vista del programador son `AppWidgets`, son pequeños interfaces de programas Android que permanecen en el escritorio del dispositivo móvil. Para añadir alguno sobra con hacer una pulsación larga sobre un área vacía del escritorio y seleccionar la opción "widget", para que aparezca la lista de todos los que hay instalados y listos para añadir. En la versión 4 de Android para añadir widgets hay que ir al menú de aplicaciones y desde la sección de widgets seleccionar uno y pulsarlo prolongadamente para arrastrarlo a la zona deseada de la pantalla de inicio.



Seleccionar un (App) Widget

Este es un ejemplo de widget, el que muestra un reloj, incluido de serie con Android:

AppWidget reloj de Android

2.1. Crear un Widget

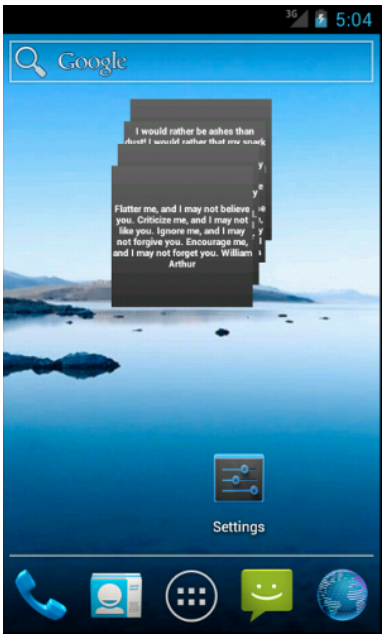
Para crear un widget se necesitan tres componentes: un recurso layout en xml que defina la interfaz gráfica del widget, una definición en XML de los metadatos del widget y por último un `IntentReceiver` que sirva para controlar el widget.

Los AppWidgets ocupan determinado tamaño y se refrescan con determinada frecuencia, metadatos que hay que declarar en el XML que define el widget. Se puede añadir como nuevo recurso XML de Android, y seleccionar el tipo Widget. Se coloca en la carpeta `res/xml/`. Por ejemplo, este es el `res/xml/miwidget.xml`:

```
<?xml version="1.0" encoding="utf-8"? >
<appwidget-provider
        xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="146dip"
    android:minHeight="72dip"
    android:updatePeriodMillis="4320000"
    android:initialLayout="@layout/miwidget_layout"
    />
```

Este XML declara que el Layout del widget se encuentra en `res/layout/miwidget_layout.xml`. Por cuestiones de diseño de Android, relacionadas con seguridad y eficiencia, los únicos componentes gráficos permitidos en el layout del widget son los layouts `FrameLayout`, `LinearLayout`, `RelativeLayout`, los views `AnalogClock`, `Button`, `Chronometer`, `ImageButton`, `ImageView`, `ProgressBar` y `TextView`. Es interesante el hecho de que los `EditText` **no** están permitidos en el layout del widget. Para conseguir este efecto lo que se suele hacer es utilizar una imagen que imite este view y al hacer click, abrir una actividad semi transparente por encima que sea la que nos permita introducir el texto.

Desde la versión 3.0 de Android los widget cuentan con nuevas características y mayor interactividad. Los nuevos componentes que se pueden utilizar son `GridView`, `ListView`, `StackView`, `ViewFlipper`, `AdapterViewFlipper`. Por ejemplo `StackView` no estaba presente entre los componentes de las versiones anteriores y es un view que permite mostrar "tarjetas" e ir pasándolas hacia delante o hacia atrás.



Widget con StackView

Un ejemplo de layout podría ser el siguiente, en el que se coloca un reloj analógico y un campo de texto al lado:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/miwidget"
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content">
    <AnalogClock
        android:id="@+id/analogClock1"
        android:layout_width="45dp"
        android:layout_height="45dp" />

        <TextView android:text=""
                android:id="@+id/TextView01"
                android:textSize="9dp"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:shadowColor="#000000"
                android:shadowRadius="1.2"
                android:shadowDx="1"
                android:shadowDy="1">
        </TextView>
</LinearLayout>
```

La pantalla de inicio de Android, por defecto está dividida en una matriz de 4x4 celdas, cada una de ellas con un mínimo de 74x74 píxeles independientes del dispositivo (dp). Para decidir la altura y la anchura del widget se pueden calcular el número de celdas que se desean ocupar, restándole a esa cantidad dos píxeles (dp) de borde. En los dispositivos donde las dimensiones mínimas que establezcamos no coincidan exactamente con las celdas de la pantalla, el tamaño de nuestro widget será extendido para llenar lo que queda

libre de las celdas.

Los AppWidgets no necesitan ninguna actividad, sino que se implementan como `IntentReceivers` con `IntentFilters` que detecten intents con acciones de actualización de widet como `AppWidget.ACTION_APPWIDGET_UPDATE,DELETED,ENABLED,DISABLED`, así como con acciones personalizadas. Se podría definir una clase que herede de `IntentReceiver` pero también se puede definir una clase que herede de `AppWidgetProvider` que es una alternativa proporcionada por Android para encapsular el procesamiento de los Intent y proveer de handlers para la actualización, borrado, habilitación y deshabilitación del widget.

Por tanto en el `AndroidManifest.xml` ya no necesitamos declarar una actividad principal. Lo que tenemos que declarar es el receptor de intents del widget:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      android:versionCode="1"
      android:versionName="1.0"
      package="es.ua.jtech.av.appwidget">
    <uses-sdk android:minSdkVersion="7" />
    <application android:icon="@drawable/icon"
                            android:label="@string/app_name">

        <receiver android:name=".MiWidget"
                        android:label="Frases Widget">
            <intent-filter>
                <action
android:name="android.appwidget.action.APPWIDGET_UPDATE" />
            </intent-filter>
            <intent-filter>
                <action android:name="es.ua.jtech.av.ACTION_WIDGET_CLICK"
/>
            </intent-filter>
            <meta-data android:name="android.appwidget.provider"
                    android:resource="@xml/miwidget" />
        </receiver>
    </application>

</manifest>
```

En el anterior ejemplo de manifest se declara, por un lado, el intent filter que capturará el evento de actualización del widget y, por otro lado el intent que captura una acción personalizada, la `es.ua.jtech.av.ACTION_WIDGET_CLICK`.

2.2. RemoteViews

La clase `RemoteViews` se utiliza para definir y manipular una jerarquía de views que se encuentra en el proceso de una aplicación diferente. Permite cambiar propiedades y ejecutar métodos. En el caso de los widgets los views están alojados en un proceso (normalmente perteneciente a la pantalla del Home) y hay que actualizarlos desde el `IntentReceiver` que definimos para controlar el widget. Una vez definido el cambio a realizar por medio de `RemoteViews`, el cambio se aplica por medio del

`AppWidgetManager` que podemos obtener a partir del contexto, como se muestra en la última línea del siguiente código.

```
RemoteViews updateViews = new RemoteViews(
context.getPackageName(),
R.layout.miwidget_layout);

//Actualizar un campo de texto del widget
updateViews.setTextViewText(R.id.TextView01, "texto");

//Comportamiento al pulsar el widget
//On-click listener que envía un broadcast intent
Intent intent = new Intent(ACTION_WIDGET_CLICK);
PendingIntent pendingIntent = PendingIntent.getBroadcast(
                                                    context, 0,
intent, 0);
updateViews.setOnClickPendingIntent(
                                              R.id.miwidget,
pendingIntent);

ComponentName thisWidget = new ComponentName(
                                                    context,
MiWidget.class);
AppWidgetManager.getInstance(context).updateAppWidget(
                                                    thisWidget,
updateViews);
```

La actualización se realiza por medio de los métodos de la clase `RemoteViews`, tales como `.setTextViewText(String)`, `.setImageViewBitmap(Bitmap)`, etc, con el propósito de actualizar los valores del layout indicado en la creación del `RemoteViews`, `R.layout.miwidget_layout` en este caso.

`RemoteViews` proporciona métodos para acceder a propiedades como `setInt(...)`, `setBoolean(...)`, `setBitmap(...)`, etc, y métodos para acceder a Views específicos, como `setTextViewText(...)`, `setTextColor(...)`, `setImageViewBitmap(...)`, `setChronometer(...)`, `setProgressBar(...)`, `setViewVisibility(...)`.

Al pulsar el widget podríamos lanzar un intent con un navegador o con una actividad que debamos ejecutar:

```
//Comportamiento al pulsar el widget: lanzar alguna actividad:
Intent defineIntent = new Intent(....);
PendingIntent pendingIntent = PendingIntent.getActivity(
        getApplicationContext(), 0, defineIntent, 0);
updateViews.setOnClickPendingIntent(R.id.miwidget,pendingIntent);
```

2.3. Recibir los intent de actualización

Para actualizaciones periódicas se puede utilizar el `AlarmManager` que podría ser programado para forzar la actualización del widget, enviando periódicamente un broadcast con la acción que hayamos definido para tal fin.

En el actual ejemplo la acción personalizada sería `"es.ua.jtech.av.ACTION_WIDGET_CLICK"`, ya que en el manifest tenemos

```
    <intent-filter>
        <action android:name="es.ua.jtech.av.ACTION_WIDGET_CLICK" />
    </intent-filter>
```

Como su nombre indica, no es una acción para actualizaciones periódicas sino para detectar el click sobre un widget. Por tanto al hacer click sobre el widget, en lugar de abrir un navegador o una actividad, lo que haremos será crear un broadcast intent con la acción definida. Esta definición formaría parte de la actualización por medio de `RemoteViews`:

```
    Intent intent = new Intent("es.ua.jtech.av.ACTION_WIDGET_CLICK");
    PendingIntent pendingIntent = PendingIntent.getBroadcast(context, 0,
intent, 0);
    updateViews.setOnClickPendingIntent(R.id.miwidget, pendingIntent);
```

Para capturar este intent implementaremos no sólo el método `onUpdate()` del widget sino también el método `onReceive()`, ya que es un `IntentReceiver`.

```
public class MiWidget extends AppWidgetProvider {
        @Override
        public void onUpdate(
                        Context context,
                        AppWidgetManager appWidgetManager,
                        int[] appWidgetIds) {
                actualizar(context);
        }
        @Override
        public void onReceive(Context context, Intent intent) {
                super.onReceive(context, intent);
                if(intent.getAction().equals(
                                "es.ua.jtech.av.ACTION_WIDGET_CLICK")){
                        actualizar(context);
                }
        }
}
```

En el método `onReceive` comprobamos si el intent recibido es el que se corresponde con la acción que hemos definido y realizamos las acciones que haga falta, por ejemplo, actualizar el widget.

2.4. Servicio de actualización

Otra alternativa para refrescar la interfaz gráfica es el uso de un servicio que actualice el widget. Para ello habría que declararlo en el manifest y declarar la clase del servicio `UpdateService extends Service` dentro de la clase `MiWidget`.

A continuación se muestra un ejemplo de clase `MiWidget`que implementa un widget:

```
public class MiWidget extends AppWidgetProvider {

        @Override
        public void onUpdate(Context context, AppWidgetManager
```

```
                        appWidgetManager, int[] appWidgetIds) {
                // Inicio de nuestro servicio de actualización:
                context.startService(
                        new Intent(context, UpdateService.class));
        }

        public static class UpdateService extends Service {
                @Override
                public int onStartCommand(Intent intent,
                                                        int flags, int
startId) {
                        RemoteViews updateViews = new RemoteViews(
                                getPackageName(),
R.layout.miwidget_layout);

                        //Aqui se actualizarían todos los tipos de Views:
                        updateViews.setTextViewText(R.id.TextView01,
                                        "Valor con el que refrescamos");
                        // ...

                        //Y la actualización a través del updateViews
creado:
                        ComponentName thisWidget = new ComponentName(
                                        this, MiWidget.class);
AppWidgetManager.getInstance(this).updateAppWidget(
                                        thisWidget, updateViews);

                        return Service.START_STICKY;
                }

                @Override
                public IBinder onBind(Intent intent) {
                        return null;
                }
        }
}
```

2.5. Actividad de configuración

Algunos widgets muestran una actividad de configuración la primera vez que son añadidos a la pantalla. Cuando el sistema solicita la cofiguración de un widget se recibe un broadcast intent con la acción `android.appwidget.action.APPWIDGET_CONFIGURE`, por tanto definiremos en el manifest nuestra actividad con el `intent-filter` correspondiente, que la ejecutará al recibir la acción:

```
<activity android:name=".MiConfigurationActivity">
        <intent-filter>
                <action
android:name="android.appwidget.action.APPWIDGET_CONFIGURE"/>
        </intent-filter>
</activity>
```

La actividad debe devolver un Intent que incluya un extra con el identificador del App Widget, usando la constante `EXTRA_APPWIDGET_ID`. Ésta es incluída en el Intent que lanza la actividad.

```
        Intent resultado = new Intent();
        result.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, appWidgetId);
        setResult(RESULT_OK, resultado);
        finish();
```

La actividad de configuración también debe ser declarada en el xml que contiene los metadatos del widget:

```
<appwidget-provider
        ...
    android:updatePeriodMillis="3600000"
    android:configure="es.ua.jtech.av.appwidget.MiConfigurationActivity"
/>
```

Esta actividad será mostrada sólo al añadir el widget a la pantalla por primera vez.