Servicios Avanzados

Índice

1. Servicios en segundo plano

Los servicios en segundo plano, `Services` son similares a los demonios de los sistemas GNU/Linux. No necesitan una aplicación abierta para seguir ejecutándose. Sin embargo para el control de un servicio (iniciarlo, detenerlo, configurarlo) sí que es necesario programar aplicaciones con sus actividades con interfaz gráfica. En esta sección vamos a ver cómo crear nuestros propios servicios.

Los servicios heredan de la clase `Service` e implementan obligatoriamente el método `onBind(Intent)`. Este método sirve para comunicación entre servicios y necesita que se defina una interfaz AIDL (Android Interface Definition Language). Devolviendo `null` estamos indicando que no implementamos tal comunicación.

```
public class MiServicio extends Service {

        @Override
        public void onCreate() {
                super.onCreate();
                //Inicalizaciones necesarias
        }

        @Override
        public int onStartCommand(Intent intent, int flags, int
startId) {
                //Comenzar la tarea de segundo plano
                return Service.START_STICKY;
        }


        @Override
        public void onDestroy() {
                super.onDestroy();
                //Terminar la tarea y liberar recursos
        }

        @Override
        public IBinder onBind(Intent arg0) {
                return null;
        }

}
```

Es muy importante declarar el servicio en `AndroidManifest.xml`:

```
        ...
        <service android:name=".MiServicio" />
    </application>
```

Si el servicio se encontrara declarado dentro de otra clase, el `android:name` contendría:

`.MiOtraClase$MiServicio`.

El ciclo de vida del servicio empieza con la ejecución del método `onCreate()`, después se invoca al método `onStartCommand(...)` y finalmente al detener el servicio se invoca al método `onDestroy()`.

Nota:
En versiones anteriores a Android 2.0 los servicios se arrancaban con el método `onStart()`. Utilizar el método `onStartCommand` y devolver la constante `Service.START_STICKY` es similar a usar el meodo `onStart()`. Esta constante se utiliza para servicios que se arrancan y detienen explícitamente cuando se necesitan.

Para arrancar y detener el servicio desde una aplicación se utilizan los métodos

```
startService(new Intent(main, MiServicio.class));
stopService( new Intent(main, MiServicio.class));
```

El servicio también puede detenerse a sí mismo con el método `selfStop()`.

2. 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 trabajar 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;
```

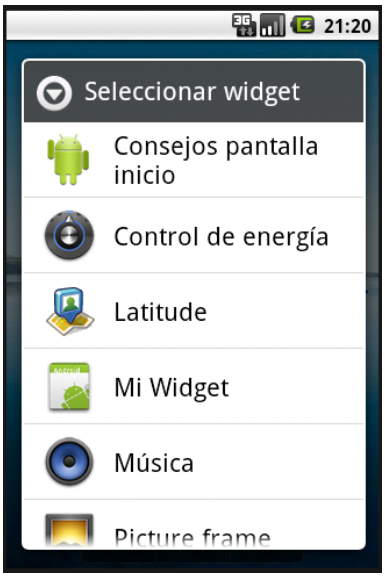
```
                @Override
                protected void onPreExecute() {
                        super.onPreExecute();
                        notificationManager =
(NotificationManager)getSystemService(
                                Context.NOTIFICATION_SERVICE);
                        notification = 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);
                }
        }
```

3. 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.



Seleccionar un (App) Widget

Este es un ejemplo de widget, el del reloj, que viene con Android:

AppWidget reloj de Android

Los AppWidgets ocupan determinado tamaño y se refrescan con determinada frecuencia, datos 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. Lo 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="600000"
    android:initialLayout="@layout/miwidget_layout"
    />
```

Este XML declara que el Layout del widget se encuentra en `res/layout/miwidget_layout.xml`.

Los AppWidgets no necesitan ninguna actividad, sino una clase que herede de `AppWidgetProvider`. Por tanto en el `AndroidManifest.xml` ya no necesitamos declarar una actividad principal. Lo que tenemos que declarar es el widget:

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

        <receiver android:name=".MiWidget" android:label="Mi
Widget">
            <intent-filter>
                <action
android:name="android.appwidget.action.APPWIDGET_UPDATE" />
            </intent-filter>
            <meta-data android:name="android.appwidget.provider"
                    android:resource="@xml/miwidget" />
        </receiver>
        <service android:name=".MiWidget$UpdateService" />

    </application>
    <uses-sdk android:minSdkVersion="8" />
</manifest>
```

También puede ser necesario declarar, además de los permisos que se requieran, el uso del servicio que actualice el widget o que realice alguna tarea en background. En el anterior manifest se declara el servicio `UpdateService` 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 que hubiera:
                        updateViews.setTextViewText(R.id.TextView01,
                                        "Valor con el que
refrescamos");
                        // ...
```

```
                        //Además, ¿qué hacer si lo pulsamos? Lanzar
alguna actividad:
                        Intent defineIntent = new Intent(....);
                        PendingIntent pendingIntent =
PendingIntent.getActivity(
                                getApplicationContext(), 0,
defineIntent, 0);
                        updateViews.setOnClickPendingIntent(R.id.miwidget,pendingInte

                        //Y la actualización del widget con el
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;
                }
        }
}
```

La actualización se realiza por medio de la clase `RemoteViews`, con métodos 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.

4. Publicación de software

Para publicar nuestras aplicaciones primero tenemos que empaquetarlas. Antes de empaquetar debemos preparar el código y comprobar que todo esté correcto:

- Nombre de la aplicación, icono y versión.
- Deshabilitar debugging en el AndroidManifest.xml (atributo `android:debuggable="false"` del tag de application).
- Eliminar cualquier mensaje de Log.
- Pedir sólo los permisos que de verdad la aplicación use, y no más de los necesarios.
- Por supuesto, haber probado la aplicación en terminales reales, a ser posible en más de uno.

Los paquetes de aplicaciones Android se pueden generar fácilmente con el plugin de Eclipse, pulsando sobre el proyecto con el botón derecho y seleccionando la opción Android Tools / Export Signed Application Package. Esto nos generaría el paquete con extensión apk (android package).

Para distribuir nuestra aplicación el paquete debe ir firmado (la firma nos la pide el

asistente de Eclipse al empaquetar). Una firma digital puede ir certificada por una entidad certificadora conocida, o bien sin autoridad certificadora, "self-signed". Se puede usar la herramienta `keytool` para generar un certificado así.

```
keytool -genkey -v -keystore myandroidapplications.keystore
        -alias myandroidapplications -keyalg RSA -validity 10000
```

Para firmar un .apk ya generado se puede utilizar la herramienta `jarsigned`. Ambas herramientas vienen con el Android SDK.

```
jarsigned -verbose -keystore myandroidapplications.keystore
          miaplicacion.apk myandroidapplications
jarsigned -verbose -certs -verify miaplicacion.apk
```

Una vez firmado el paquete, ya está listo para ser publicado.

Nota:

Si cambiamos de certificado en las versiones siguientes del programa, podemos tener el problema de que el terminal compruebe el certificado y no le coincida. En este caso hay que desinstalar por completo la aplicación y entonces instalar la nueva versión. Los updates de la aplicación asumen que el certificado no va a cambiar.

Para publicar en el Android Market, http://market.android.com/ debemos darnos de alta como desarrollador, pagando 25$ a través de Google Checkout, facilitando la información que nos pida el Google Checkout Merchant. Al subir la aplicación nos preguntará información sobre ésta en varios idiomas, para qué paises funciona, su tipo y categoría, el precio, la información de copyright y un contacto de soporte.

Otra opción es colgar la aplicación empaquetada en nuestro propio servidor. En este caso quien desee instalársela necesitará tener habilitada la instalación desde fuentes desconocidas en la configuración de su terminal Android.