Servicios Avanzados

Servicios en segundo plano

Los servicios en segundo plano, `Service`s 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()`.

> 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()`.

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.

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.

```
// Gets an instance of the NotificationManager service
NotificationManager mNotifyMgr =
        (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
NotificationCompat.Builder mBuilder =
    new NotificationCompat.Builder(this)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("My notification")
    .setContentText("Hello World!");
```

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.

```
NotificationCompat.Builder mBuilder;
...
// Sets an ID for the notification
int mNotificationId = 001;

// Builds the notification and issues it.
mNotifyMgr.notify(mNotificationId, mBuilder.build());
```

Si queremos que la notificación habra una actividad:

```
Intent contentIntent = new Intent(this, ResultActivity.class);
...
// Because clicking the notification opens a new ("special") activity, there's
// no need to create an artificial back stack.
PendingIntent pendingIntent = PendingIntent.getActivity(this,0,resultIntent,0);

mBuilder.setContentIntent(resultPendingIntent);
```

`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:

```
mNotificationManager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// Sets an ID for the notification, so it can be updated
int notifyID = 1;
mNotifyBuilder = new NotificationCompat.Builder(this)
    .setContentTitle("New Message")
    .setContentText("You've received new messages.")
    .setSmallIcon(R.drawable.ic_notify_status)
numMessages = 0;
// Start of a loop that processes data and then notifies the user
...
    mNotifyBuilder.setContentText(currentText)
        .setNumber(++numMessages);
    // Because the ID remains unchanged, the existing notification is
    // updated.
    mNotificationManager.notify(
            notifyID,
            mNotifyBuilder.build());
...
```