Tutorial Para Crear Un Servicio En Android

James Revelo

La clase `Service` es la encargada de crear un *Servicio* en Android.

Pero... *¿para qué un servicio en tu aplicación?*

Bueno, como habíamos visto en Componentes de una aplicación Android, un servicio es un elemento que lleva a cabo operaciones de larga duración en segundo plano y sin ninguna clase de interfaz.

Son de gran utilidad para acciones como:

- Sincronizar aplicaciones con la nube.
- Administrar *Notificaciones Push* (Firebase Cloud Messaging, Parse, etc.).
- Monitorear información.
- Reproducir música sin tener contacto directo con la interfaz.
- Almacenar información en la base de datos.
- Gestionar escritura y lectura de archivos.
- y muchas más...

Su gran ventaja está en que ejecutan operaciones en segundo plano, es decir, cuando el usuario no tiene contacto directo con tu aplicación. A diferencia del uso de tareas asíncronas, las cuales se ejecutan solo si la aplicación está activa.

En este artículo veremos la implementación de servicios en nuestras aplicaciones Android.

Además veremos cómo usar la clase `IntentService`, un tipo de servicio que facilita la creación de un hilo de trabajo aislado para no interferir con el Main Thread.

Descargar Proyecto Android Studio De “Service Out”

Si deseas saber cuál será el resultado final al aplicar las enseñanzas de este tutorial, entonces querrás ver el siguiente video:

Para desbloquear el link de descarga del código completo, sigue estas instrucciones:

1. ¿Qué Es Un Servicio En Android?

Como mencioné al inicio del artículo, un servicio es un componente android que ejecuta instrucciones en segundo plano. Este se representa en Java con la clase `Service`.

Esto significa que si el usuario cambia hacia otra aplicación, el servicio seguirá activo hasta que cumpla su objetivo o sea terminado por el sistema, debido a la falta de recursos.

Iniciar un servicio desde otro componente es fácil, solo usa el método `startService()` junto a un intent explícito que indique la clase específica.

```
startService(new Intent(this, Servicio.class));
```

Del mismo modo, puedes terminar su ejecución con el método `stopService()`:

```
stopService(new Intent(this, Servicio.class));
```

2. Tipos De Servicios En Android

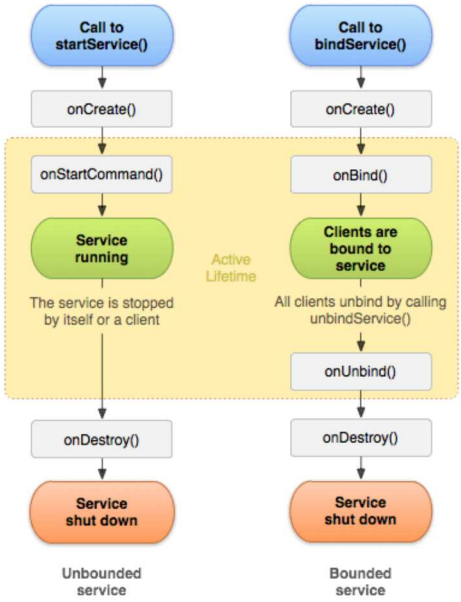
Existen dos tipos de servicios. Aquel que se inicia explícitamente a través del método `startService()` (*Started Service*) y el que se liga a un componente con `bindService()` (*Bound Service*).

El started service realiza una operación determinada y se cierra al momento de finalizarla. Por otro lado, la esperanza de vida de un bound service ó servicio ligado depende de la vida del elemento que lo ató.

Sin embargo, un bound service puede comunicarse directamente con el componente al que está ligado. Esto significa compartir datos estructurados, realizar operaciones en conjunto y gran variedad de interacciones posibles donde un started service se ve limitado.

3. Crear Un Started Service

Para crear una implementación en Java ten en cuenta el ciclo de vida de un servicio.



Dependiendo del tipo de servicio, así mismo se usan los controladores necesarios de la clase `Service`. Veamos los métodos que controlan el flujo:

- `onCreate()`: Se ejecuta cuando el servicio está creado en memoria. Si el servicio ya está activo, entonces se evita de nuevo su llamada.
- `onStartCommand()`:Método que ejecuta las instrucciones del servicio. Se llama solo si el servicio se inició con `startService()`.
- `onBind()`: Solo se ejecuta si el servicio fue ligado con `bindService()` por un componente. Retorna una interfaz Java de comunicación del tipo IBinder. Este método siempre debe llamarse, incluso dentro de los started services, los cuales retornan `null`.
- `onDestroy()`: Se llama cuando el servicio está siendo destruido. Importantísimo que dentro de este método detengas los hilos iniciados.

El siguiente es un ejemplo de un started service ...

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

public class DownloadService extends Service {

    public DownloadService() {
    }

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


    @Override
    public void onCreate() {
        Log.d(TAG, "Servicio creado...");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.d(TAG, "Servicio iniciado...");

        return START_NOT_STICKY;
    }

    @Override
    public void onDestroy() {
        Log.d(TAG, "Servicio destruido...");
    }


}
```

Si notas, `onStartCommand()` retorna en una constante llamada `START_NOT_STICKY`. Este campo hace parte de la clase `Service`, el cual indica que el servicio no debe recrearse al ser destruido sin importar que haya quedado un trabajo pendiente.

De igual modo, también puedes retornar en:

- `START_STICKY`: Crea de nuevo el servicio después de haber sido destruido por el sistema. En este caso llamará a `onStartCommand()` referenciando un intent nulo.
- `START_REDELIVER_INTENT`: Crea de nuevo el servicio si el sistema lo destruyó. A diferencia de `START_STICKY`, esta vez sí se retoma el último intent que recibió el servicio.

Tus servicios no se instanciarán si no los declaras en el Android Manifest como un componente de la aplicación. Así que solo incluye la etiqueta `<service>` dentro del nodo `<application>`:

```
<application
    ... >
    ...
    <service
        android:name=".DownloadService"
        android:enabled="true"
        android:exported="true" >
    </service>
```

```
</application>
```

La etiqueta anterior contiene tres parámetros: `name` se refiere al nombre de la clase de implementación del servicio; `enabled` indica si es posible crear instancias del servicio; y `exported` establece si los componentes de otras apps pueden iniciar o interactuar con el servicio.

Puntos importantes a tener en cuenta:

- Usa el método `stopSelf()` dentro del servicio, para detener el servicio inmediatamente. Habrá ocasiones donde `stopService()` no pueda llamarse debido a que no sabemos el momento preciso en que el servicio terminará su trabajo.
- Un servicio se ejecuta por defecto en el hilo principal. Si usas trabajos pesados dentro de ellos, puede que recibas diálogos ANR. En estos casos debes usar hilos para no interferir.
- Termina los hilos iniciados dentro de `onDestroy()`. Si los dejas vivos, seguirán ejecutándose innecesariamente, si es que aún no han acabado su trabajo.
- Ejecutar un servicio más de una vez, crea varias instancias de este elemento, así que ten cuidado.
- Si solo usarás un hilo para tu trabajo dentro del servicio, entonces usa la subclase `IntentService`, la cual maneja la creación de hilos por tí.

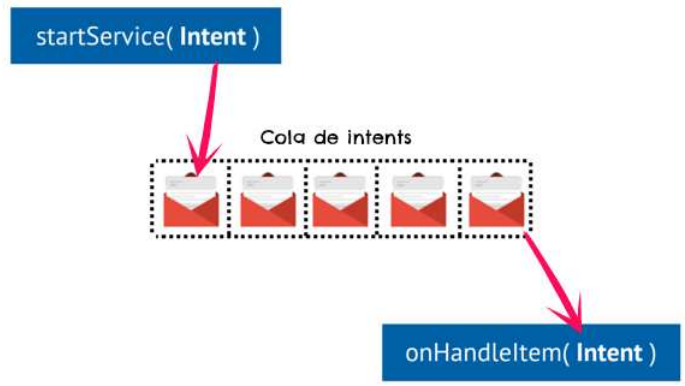
4. Crear Un IntentService En Android Studio

Si te da pereza crear un hilo de trabajo para gestionar tus operaciones dentro de un servicio, entonces usa la clase `IntentService`.

Esta variación crea un hilo de trabajo para evitar interferencias en la UI. Una vez terminado dicho trabajo, el servicio es finalizado automáticamente.

A diferencia de `Service`, `IntentService` no ejecuta varias instancias al mismo tiempo si es llamado múltiples ocasiones.

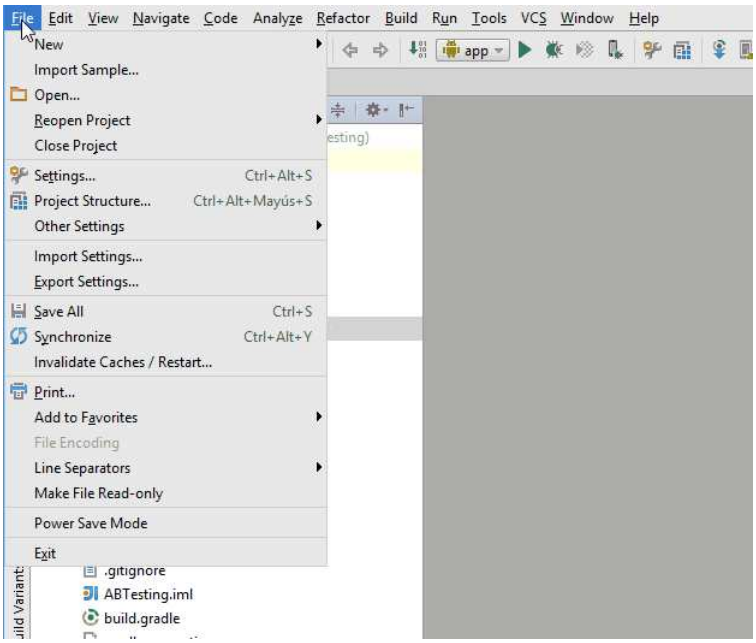
Lo que haces es dejar en cola los intents que van llegando y así ejecutarlos de forma secuencial.



El trabajo será ejecutado dentro del método `onHandleIntent()` y no en `onStartCommand()`. Este método recibe como parámetro un intent que puede traer una acción asociada y parámetros que indiquen que decisión tomar.

```
@Override
protected void onHandleIntent(Intent intent) {
    if (intent != null) {
        // Acciones por realizar...
        }
    }
}
```

Si deseas crear un Intent Service en Android Studio ve a **File** > **New** > **Service** > **Intent Service**. Esto proveerá una clase base lista para extenderse.



El código de abajo es el resultado de la creación automática:

```
import android.app.IntentService;
import android.content.Intent;


/**
 * An {@link IntentService} subclass for handling asynchronous task requests in
 * a service on a separate handler thread.
 * <p/>
 * TODO: Customize class - update intent actions and extra parameters.
 */
public class WorkingIntentService extends IntentService {
    // TODO: Rename actions, choose action names that describe tasks that this
    // IntentService can perform, e.g. ACTION_FETCH_NEW_ITEMS
    public static final String ACTION_FOO = "com.herprogramacion.abtesting.action.FOO";
    public static final String ACTION_BAZ = "com.herprogramacion.abtesting.action.BAZ";

    // TODO: Rename parameters
    public static final String EXTRA_PARAM1 = "com.herprogramacion.abtesting.extra.PARAM1";
    public static final String EXTRA_PARAM2 = "com.herprogramacion.abtesting.extra.PARAM2";

    public WorkingIntentService() {
        super("WorkingIntentService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        if (intent != null) {
            final String action = intent.getAction();
            if (ACTION_FOO.equals(action)) {
                final String param1 = intent.getStringExtra(EXTRA_PARAM1);
                final String param2 = intent.getStringExtra(EXTRA_PARAM2);
                handleActionFoo(param1, param2);
            } else if (ACTION_BAZ.equals(action)) {
                final String param1 = intent.getStringExtra(EXTRA_PARAM1);
                final String param2 = intent.getStringExtra(EXTRA_PARAM2);
                handleActionBaz(param1, param2);
            }
        }
    }

    /**
     * Handle action Foo in the provided background thread with the provided
     * parameters.
     */
    private void handleActionFoo(String param1, String param2) {
        // TODO: Handle action Foo
        throw new UnsupportedOperationException("Not yet implemented");
    }

    /**
     * Handle action Baz in the provided background thread with the provided
     * parameters.
     */
    private void handleActionBaz(String param1, String param2) {
        // TODO: Handle action Baz
        throw new UnsupportedOperationException("Not yet implemented");
    }
}
```

¿Puntos a tener en cuenta?

- No es necesario sobrescribir `onBind()`. `IntentService` ya lo trae implementado con retorno `null`.
- Tampoco se debe llamar a `stopSelf()`, ya que el servicio se detiene automáticamente cuando despacha la cola de intents en proceso.
- Define constantes para acciones y parámetros que se asocien a los intents que recibe o envía el servicio. Con ello puedes decidir qué acción tomar dentro de `onHandleIntent()` y comunicar datos. Un ejemplo son las acciones *FOO* y *BAZ* del Intent Service de arriba, las cuales traen un método personalizado de manejo tipo `handleAction*()`.

5. Crear Un Bound Service

El enfoque de este artículo no es el uso de bound services o servicios ligados, ya que se utilizan en temas avanzados que aún no vamos a tocar. Sin embargo, podemos entender un poco la lógica de estos.

La idea es proveer acceso al servicio a otros componentes (clientes). Normalmente serán actividades. Para ello se usa la interfaz `IBinder` que retorna la instancia del servicio hacia el elemento que desea interactuar con él.

Veamos un ejemplo...

```
import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import android.util.Log;

public class JsonParsingService extends Service {

    private static final String LOGTAG = JsonParsingService.class.getSimpleName();


    @Override
    public void onCreate() {
        Log.i(LOGTAG, "Tracking Service Running...");
    }

    @Override
    public void onDestroy() {
        Log.i(LOGTAG, "Tracking Service Stopped...");
    }


    /* Método de acceso */
    public class ParsingBinder extends Binder {
        JsonParsingService getService() {
            return JsonParsingService.this;
        }
```

```
    }

    private final IBinder binder = new ParsingBinder();

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

El servicio anterior crea una clase interna llamada `ParsingBinder`, que extiende de `Binder` con el fin de retornar la instancia del servicio a través de `getService()`.

Luego se crea una instancia de esta clase y se retorna en `onBind()`. Con esta convención, es posible acceder a la instancia del servicio.

Ahora veamos a groso modo como accederlo desde una actividad:

```
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.support.v7.app.AppCompatActivity;

public class DataActivity extends AppCompatActivity {
    Intent serviceIntent;
    JsonParsingService parsingService;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_data);

        serviceIntent = new Intent(this, JsonParsingService.class);
    }

    @Override
    public void onResume() {
        super.onResume();
        // Iniciar el servicio
        startService(serviceIntent);
        // Atar el servicio a la actividad
        bindService(serviceIntent, serviceConnection, Context.BIND_AUTO_CREATE);
    }

    private ServiceConnection serviceConnection = new ServiceConnection() {
        public void onServiceConnected(ComponentName className, IBinder service) {
            parsingService = ((JsonParsingService.ParsingBinder) service).getService();
            // Acciones...
        }

        public void onServiceDisconnected(ComponentName className) {
            parsingService = null;
            // Acciones...
        }
    };
}
```

Desde el cliente solo iniciamos el servicio y luego lo atamos a la actividad con `bindService()`.

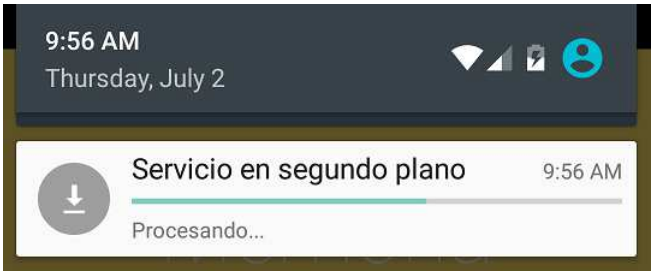
`bindService()` recibe una instancia de la interfaz `ServiceConnection`, la cual monitorea el estado del servicio dentro de la aplicación.

Esta clase provee dos controladores: `onServiceConnected()` que se ejecuta cuando el servicio está listo y `onServiceDisconnected()` para indicar cuando el servicio quedó desligado.

Aunque es una interacción básica, espero crear un artículo más específico para este tema.

6. Poner Servicios En Primer Plano

Un servicio en primer plano tiene una prioridad crítica en Android. Tanto así, que el usuario no puede terminarlo manualmente hasta que se termine.



Para que esto sea posible, el servicio en primer plano debe proveer una notificación en la barra de estado que indique su existencia.

La única forma de quitarlo del primer plano es terminándolo programáticamente o cuando este finalice su trabajo.

El traslado a primer plano se lleva cabo con el método `startForeground()` dentro del servicio. Si deseas pasarlo a segundo plano de nuevo, entonces usas `stopForeground()`.

Te recomiendo leas mi artículo sobre notificaciones en Android para poder entender la creación de estas.

Por ejemplo...

```
// Se construye la notificación
NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
        .setSmallIcon(android.R.drawable.stat_sys_download_done)
```

```
        .setContentTitle("Servicio en segundo plano")
        .setContentText("Procesando...");

// Crear Intent para iniciar una actividad al presionar la notificación
Intent notificationIntent = new Intent(this, SomeActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);
builder.setContentIntent(pendingIntent);

// Poner en primer plano
startForeground(1, builder.build());

// Acciones de proceso...
```

Luego del código anterior es posible que `stopForeground()` se ejecute en algún lugar por si se desea descartar la notificación. Es necesario que pases el valor de `true` como parámetro para hacerlo efectivo.

```
stopForeground(true);
```

7. Ejemplo De Servicios En Android

Como viste en el video del inicio, la aplicación de ejemplo tiene una actividad experimental, dividida en dos secciones.

En la parte superior tenemos un servicio que monitorea la memoria disponible en el dispositivo incluso si la aplicación es puesta en segundo plano.

La parte inferior contiene un `IntentService` que simula un proceso de 10 segundos en primer plano. Se despliega una notificación que muestra el progreso de avance y al terminar el servicio se destruye.

Así que manos a la obra...

Paso 1. Abre Android Studio y ve a **File > New > New Project...** para crear un nuevo proyecto. Asígnale el nombre de **"Service Out"**, añade una actividad en blanco con **Blank Activity** y confirma.

Paso 2. Ve a **res/values/styles.xml** y modifica el estilo de la aplicación para eliminar la action bar de nuestra actividad. Recuerda que para ello usamos el estilo padre `Theme.AppCompat.NoActionBar`.

styles.xml

```
<resources>
    <style name="AppTheme" parent="Theme.AppCompat.NoActionBar">
    </style>
</resources>
```

Paso 3. Ahora crea un archivo colors.xml dentro de **res/values** y añade los colores de abajo. Allí pondremos los colores de los fondos y los botones.

colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="window_background">@android:color/white</color>

    <color name="topPanelColor">#F5D563</color>
    <color name="bottomPanelColor">#00BBD2</color>

    <color name="blueFlatShadow">#07A4B5</color>
    <color name="yellowFlatShadow">#DFB43F</color>
</resources>
```

Paso 4. Lo siguiente es aumentar el tamaño que usaremos para los text views y cambiar las dimensiones de los botones. Dentro de **res/values /dimens.xml** añade los ítems de abajo.

```
<resources>
    <dimen name="metric_size">48sp</dimen>

    <dimen name="run_button_width">300dp</dimen>
    <dimen name="run_button_height">60dp</dimen>
</resources>
```

Paso 5. Ahora diseñaremos el layout de la actividad principal. Añadiremos un `LinearLayout` como padre de otros dos linear layout. Con ello tendremos una división entre los paneles con un peso equivalente de 50 unidades.

En el panel superior debemos añadir un ToggleButton para cambiar entre INICIAR/DETENER.

activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <LinearLayout
        android:id="@+id/topPanel"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="50"
        android:background="@color/topPanelColor"
        android:gravity="center"
        android:orientation="vertical">

        <TextView
            android:id="@+id/memory_ava_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:fontFamily="sans-serif-thin"
            android:text="Memoria"
            android:textColor="@android:color/white"
            android:textSize="@dimen/metric_size" />

        <ToggleButton
```

```
            android:id="@+id/toggleButton"
            android:layout_width="@dimen/run_button_width"
            android:layout_height="@dimen/run_button_height"
            android:layout_gravity="center_horizontal"
            android:layout_marginBottom="24dp"
            android:background="@color/yellowFlatShadow"
            android:text="New ToggleButton"
            android:textOff="INICIAR SERVICE"
            android:textOn="DETENER SERVICE" />

    </LinearLayout>

    <LinearLayout
        android:id="@+id/bottomPanel"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="50"
        android:background="@color/bottomPanelColor"
        android:gravity="center_vertical"
        android:orientation="vertical">

        <TextView
            android:id="@+id/progress_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:fontFamily="sans-serif-thin"
            android:text="Progreso"
            android:textColor="@android:color/white"
            android:textSize="@dimen/metric_size" />

        <Button
            android:id="@+id/turn_intent_service"
            android:layout_width="@dimen/run_button_width"
            android:layout_height="@dimen/run_button_height"
            android:layout_gravity="center_horizontal"
            android:background="@color/blueFlatShadow"
            android:onClick="onClickTurnIntentService"
            android:text="INICIAR INTENT SERVICE" />
    </LinearLayout>
</LinearLayout>
```

Paso 6. Crea una nueva clase llamada **Constants.java** dentro del proyecto. Su propósito será contener las constantes que usaremos para las acciones y parámetros de los intents de los servicios.

Constants.java

```
/**
 * Contiene las constantes de las acciones de los servicios y sus parámetros
 */
public class Constants {
    /**
     * Constantes para {@link MemoryService}
     */
    public static final String ACTION_RUN_SERVICE = "com.herprogramacion.memoryout.action.RUN_SERVICE";
    public static final String ACTION_MEMORY_EXIT = "com.herprogramacion.memoryout.action.MEMORY_EXIT";

    public static final String EXTRA_MEMORY = "com.herprogramacion.memoryout.extra.MEMORY";

    /**
     * Constantes para {@link ProgressIntentService}
     */
    public static final String ACTION_RUN_ISERVICE = "com.herprogramacion.memoryout.action.RUN_INTENT_SERVICE";
    public static final String ACTION_PROGRESS_EXIT = "com.herprogramacion.memoryout.action.PROGRESS_EXIT";

    public static final String EXTRA_PROGRESS = "com.herprogramacion.memoryout.extra.PROGRESS";

}
```

Cada servicio tendrá dos acciones asociadas. Una para indicar que está corriendo (*RUN*) y otra para indicar que ya terminó (*EXIT*).

Debido a que deseamos mandar los valores desde los servicios a las acciones para notificarlos en los text views de la actividad, necesitamos extras para obtener los valores.

En este caso `EXTRA_MEMORY` se refiere a la cantidad de memoria disponible que se notificará. Y `EXTRA_PROGRESS` al progreso que se da en el segundo servicio.

Paso 7. Añade una nueva clase con el nombre de **MemoryService.java** con la definición de abajo. La idea es repetir una misma tarea cada segundo, para obtener la memoria disponible en el sistema.

Recuerda que la clase Timer nos permite realizar una tarea del tipo TimerTask repetidas ocasiones. En este caso deseamos hacerlo hasta que el usuario decida parar el servicio. Para ello se usa el método `scheduleAtFixedRate()`, donde el tercer parámetro es el tiempo entre ejecuciones.

MemoryService.java

```
import android.app.ActivityManager;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;

import java.util.Timer;
import java.util.TimerTask;

/**
 * Un {@link Service} que notifica la cantidad de memoria disponible en el sistema
 */
public class MemoryService extends Service {
    private static final String TAG = MemoryService.class.getSimpleName();
    TimerTask timerTask;

    public MemoryService() {
    }

    @Override
```

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


    @Override
    public void onCreate() {
        Log.d(TAG, "Servicio creado...");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.d(TAG, "Servicio iniciado...");


        final ActivityManager.MemoryInfo memoryInfo = new ActivityManager.MemoryInfo();
        final ActivityManager activityManager =
                (ActivityManager) getSystemService(ACTIVITY_SERVICE);


        Timer timer = new Timer();

        timerTask = new TimerTask() {
                    @Override
                    public void run() {
                        activityManager.getMemoryInfo(memoryInfo);
                        String availMem = memoryInfo.availMem / 1048576 + "MB";

                        Log.d(TAG, availMem);

                        Intent localIntent = new Intent(Constants.ACTION_RUN_SERVICE)
                                .putExtra(Constants.EXTRA_MEMORY, availMem);

                        // Emitir el intent a la actividad
                        LocalBroadcastManager.
                                getInstance(MemoryService.this).sendBroadcast(localIntent);
                    }
                };

        timer.scheduleAtFixedRate(timerTask, 0, 1000);

        return START_NOT_STICKY;
    }

    @Override
    public void onDestroy() {
        timerTask.cancel();
        Intent localIntent = new Intent(Constants.ACTION_MEMORY_EXIT);

        // Emitir el intent a la actividad
        LocalBroadcastManager.
                getInstance(MemoryService.this).sendBroadcast(localIntent);
        Log.d(TAG, "Servicio destruido...");
    }


}
```

La información sobre la memoria se obtiene a través de `MemoryInfo`.

Primero obtén el `ActivityManager` a través de `getSystemService()` con la constante `ACTIVITY_SERVICE`.

Luego pides la información con `ActivityManager.getMemoryInfo()` y se la asignas al objeto `MemoryInfo`. Con esto ya puedes consultar el atributo `availMem`, el cual contiene el dato de la memoria disponible actual.

Pero... *¿Cómo emitir el intent y la memoria disponible?*

A través del `LocalBroadcastManager`. Este elemento emite un mensaje en forma de intent hacia los elementos de nuestra aplicación con el método `sendBroadcast()`.

Importante resaltar que en `onDestroy()` cancelamos la timer task, ya que no deseamos que se ejecute indefinidamente. Además emitimos la acción de terminación del servicio para que la actividad tome la decisión necesaria.

Paso 8. Ahora es turno de crear el `IntentService` que gestionará el proceso de conteo en primer plano. Añade una clase llamada **ProgressIntentService.java** e incluye el código de abajo:

```
import android.app.IntentService;
import android.content.Intent;
import android.support.v4.app.NotificationCompat;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;
import android.widget.Toast;

/**
 * Un {@link IntentService} que simula un proceso en primer plano
 * <p>
 */
public class ProgressIntentService extends IntentService {
    private static final String TAG = ProgressIntentService.class.getSimpleName();


    public ProgressIntentService() {
        super("ProgressIntentService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        if (intent != null) {
            final String action = intent.getAction();
            if (Constants.ACTION_RUN_ISERVICE.equals(action)) {
                handleActionRun();
            }
        }
    }
```

```
    /**
     * Maneja la acción de ejecución del servicio
     */
    private void handleActionRun() {
        try {
            // Se construye la notificación
            NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
                    .setSmallIcon(android.R.drawable.stat_sys_download_done)
                    .setContentTitle("Servicio en segundo plano")
                    .setContentText("Procesando...");

            // Bucle de simulación
            for (int i = 1; i <= 10; i++) {

                Log.d(TAG, i + ""); // Logueo

                // Poner en primer plano
                builder.setProgress(10, i, false);
                startForeground(1, builder.build());

                Intent localIntent = new Intent(Constants.ACTION_RUN_ISERVICE)
                        .putExtra(Constants.EXTRA_PROGRESS, i);

                // Emisión de {@code localIntent}
                LocalBroadcastManager.getInstance(this).sendBroadcast(localIntent);

                // Retardo de 1 segundo en la iteración
                Thread.sleep(1000);
            }
            // Quitar de primer plano
            stopForeground(true);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onDestroy() {
        Toast.makeText(this, "Servicio destruido...", Toast.LENGTH_SHORT).show();

        // Emisión para avisar que se terminó el servicio
        Intent localIntent = new Intent(Constants.ACTION_PROGRESS_EXIT);
        LocalBroadcastManager.getInstance(this).sendBroadcast(localIntent);
    }
}
```

Esta vez incluimos las acciones en `onHandleIntent()` como vimos en los apartados anteriores.

Dentro de este método comprobamos que la acción del intent sea `Constants.ACTION_RUN_ISERVICE`. Si es así, entonces iniciamos el método `handleActionRun()`.

`handleActionRun()` se encarga de:

- Crear una notificación para mostrar cuando se invoque `startForeground()`.
- Iniciar un bucle que simula una operación que tarda 10 segundos en terminar.
- Actualizar el progreso de la notificación con `setProgress()`
- Enviar el progreso hacia la actividad a través del `LocalBroadcastManager`.
- Quitar el servicio de primer plano con `stopForeground()` al terminar el bucle.

Dentro de `onDestroy()` se envía la señal de terminación del servicio.

Paso 9. Declara los servicios en el **AndroidManifest.xml**.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.herprogramacion.memoryout" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name=".MemoryService"
            android:enabled="true"
            android:exported="true" >
        </service>
        <service
            android:name=".ProgressIntentService"
            android:exported="false" >
        </service>
    </application>

</manifest>
```

Paso 10. Por último se deben poner a correr los servicios cuando los botones sean presionados. Esto se lleva a cabo con `startService()` dentro de las respectivas escuchas.

Ten en cuenta que para recibir los intents emitidos desde los servicios es necesario registrar un BroadcastReceiver asociado a la actividad.

En el artículo sobre componentes android vimos que este elemento se comporta como una especie de antena que recibe señales, que si se ajusta a las

frecuencias necesarias (acción del intent), podrá recibir información de otros componentes.

Veamos...

MainActivity.java

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.support.v4.content.LocalBroadcastManager;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.CompoundButton;
import android.widget.TextView;
import android.widget.ToggleButton;

public class MainActivity extends AppCompatActivity {
    /**
     * Text views para los datos
     */
    private TextView memoryUsageText;
    private TextView progressText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Obtener las etiquetas
        memoryUsageText = (TextView) findViewById(R.id.memory_ava_text);
        progressText = (TextView) findViewById(R.id.progress_text);

        // Obtener botón de monitoreo de memoria
        ToggleButton button = (ToggleButton) findViewById(R.id.toggleButton);

        // Setear escucha de acción
        button.setOnCheckedChangeListener(
                new CompoundButton.OnCheckedChangeListener() {
                    @Override
                    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
                        Intent intentMemoryService = new Intent(
                                getApplicationContext(), MemoryService.class);
                        if (isChecked) {
                            startService(intentMemoryService); //Iniciar servicio
                        } else {
                            stopService(intentMemoryService); // Detener servicio
                        }
                    }
                }
        );


        // Filtro de acciones que serán alertadas
        IntentFilter filter = new IntentFilter(
                Constants.ACTION_RUN_ISERVICE);
        filter.addAction(Constants.ACTION_RUN_SERVICE);
        filter.addAction(Constants.ACTION_MEMORY_EXIT);
        filter.addAction(Constants.ACTION_PROGRESS_EXIT);

        // Crear un nuevo ResponseReceiver
        ResponseReceiver receiver =
                new ResponseReceiver();
        // Registrar el receiver y su filtro
        LocalBroadcastManager.getInstance(this).registerReceiver(
                receiver,
                filter);
    }

    /**
     * Método onClick() personalizado para {@code turn_intent_service}
     * @param v View presionado
     */
    public void onClickTurnIntentService(View v) {
        Intent intent = new Intent(this, ProgressIntentService.class);
        intent.setAction(Constants.ACTION_RUN_ISERVICE);
        startService(intent);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
    }

    // Broadcast receiver que recibe las emisiones desde los servicios
    private class ResponseReceiver extends BroadcastReceiver {

        // Sin instancias
        private ResponseReceiver() {
        }

        @Override
        public void onReceive(Context context, Intent intent) {
            switch (intent.getAction()) {
                case Constants.ACTION_RUN_SERVICE:
                    memoryUsageText.setText(intent.getStringExtra(Constants.EXTRA_MEMORY));
                    break;

                case Constants.ACTION_RUN_ISERVICE:
                    progressText.setText(intent.getIntExtra(Constants.EXTRA_PROGRESS, -1) + "");
                    break;

                case Constants.ACTION_MEMORY_EXIT:
                    memoryUsageText.setText("Memoria");
                    break;

                case Constants.ACTION_PROGRESS_EXIT:
```

```
                    progressText.setText("Progreso");
                    break;
                }
            }
        }
    }
```

¿Que hice para que la actividad recibiera los mensajes?

1. Declaré un `BroadcastReceiver` interno y sobrescribí el controlador `onReceive()`. Dentro de él usé un `switch` para tomar decisiones basado en las acciones del intent entrante.
2. Dentro de `onCreate()` instancié un `IntentFilter`, el cual contiene las acciones por las que estaremos pendientes.
3. Creé una nueva instancia del broadcast receiver y luego registré su presencia con el método `LocalBroadcastManager.registerReceiver()`, asociando los filtros que deseamos que coincidan con las señales entrantes.

Paso 11. Y ahora es turno de ejecutar la aplicación en Android Studio.

Conclusión

Este artículo te mostró como crear servicios iniciados para correr tareas en segundo plano. También vimos como enviar un servicio en primer plano hasta que termine su cometido.

Los servicios nos permitirán construir aplicaciones más avanzadas que provean mejores características a nuestros usuarios y que mantenga la integridad de la información.

Así que ahora puedes combinar los servicios con otros componentes y características Android para dar forma a tus apps.

Icono de la aplicación