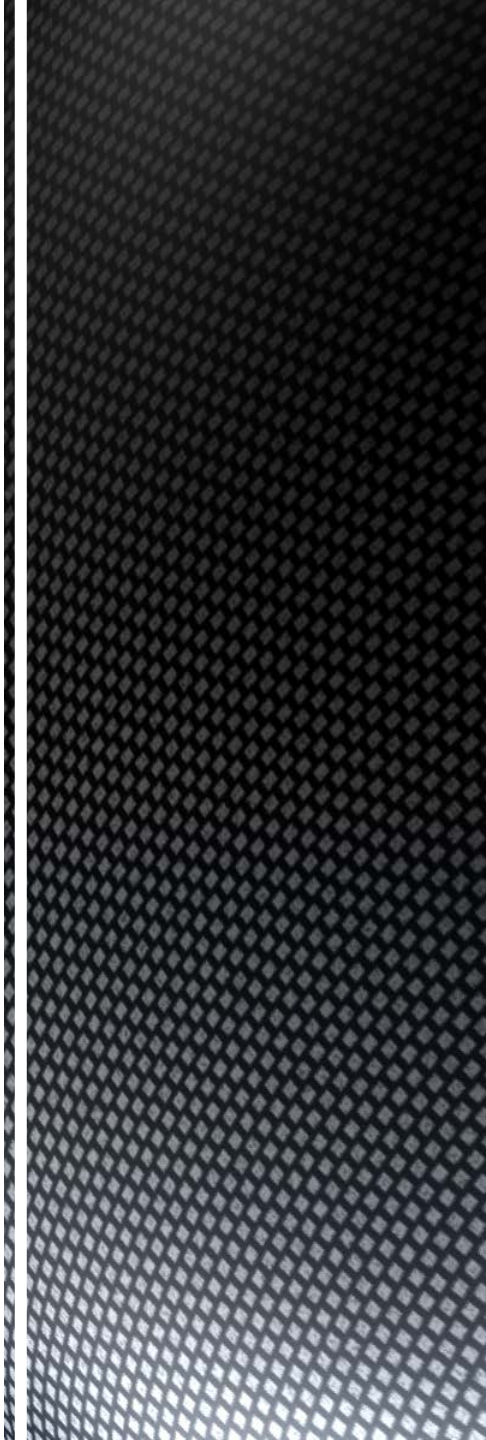
Android

Servicios



Servicios

Un servicio es un componente de Android que representa:

- La necesidad de una aplicacíon de ejecutar una operación pesada **sin mantener interacción con el usuario**
- **Proporcionar funcionalidad** a otras aplicaciones.
- **Puede** realizar operaciones costosas computacionalmente en **background**
- **No proporciona** Interfaz de usuario

Servicios del Sistema

Android proporciona y corre servicios predefinidos y las aplicaciones los usan con los permisos apropiados.

Los servicios del sistema se acceden mediante una Manager class.

Se puede acceder a ellos con el método getSystemService()

La clase Context define varias constantes para acceder a estos servicios.

Servicios de Aplicación.

Una aplicación puede definir nuevos servicios, para uso propio o no.

Podemos gracias a los servicios generar aplicaciones “responsive” o actualizadas, es decir, el servicio recoge información y cuando la aplciación arranca tiene accesible la información actualizada.

Servicios

La Clase de un Service debe tener una declaración <service> en el **Manifest**

Hay que declarar todos los servicios en el Manifest de la aplicación usando:
<service> como hijo de <application>.

El atributo android:name es el único obligatorio. Especifica el nombre de la Clase java que implementa el servicio

Una vez publicada la aplicación **el nombre del servicio no debería cambiar**.

Para garantizar la privacidad del servicio usar android:exported = false

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

- El servicio iniciado por la misma o cualquiera otra aplicación se ejecutará en background incluso cuando el usuario cambie o cierre la aplicación que inició el servicio.
- Un componente o aplicación se puede vincular al servicio para interactuar o comunicar con el (IPC Interprocess Communication)

Servicios Foreground

Un Foreground service es un servicio que tiene la misma prioridad que una Activity activa y por tanto no debe ser matado por Android, incluso si el sistema requiere recursos.

Un Foreground service debe proporcionar una OnGoing Notification para el Status bar, que significa que la notificación no puede ser eliminada a menos que el servicio se pare (stop()) o se elimine del foreground. La OnGoing notificacón se lanzan con:

```
startForeground(ONGOING_NOTIFICATION_ID, notification);
```

Servicios

Un servicio puede funcionar en uno de los siguientes o en ambos modos :

Started

Un servicio esta “Started” cuando un componente (p.ej. una actividad) lo inicia llamando a su método startService().

El servicio corre indefinidamente en backgroud, incluso cuando el elemento que lo lanzó es destruido.

Un “Started Service”, normalmente, realiza una única operación y no devuelve un valor al que lo llama (p.ej. descarga de fichero).

Cuando la operación termina el servicio debería auto-finalizarse.

Bound

Un servicio está “bound” (ligado) cuando un component se vincula a el llamando a bindService()

Un “Bound Service” ofrece una interfaz cliente-servidor que permite a los componentes **interactuar y comunicar** con el, es decir, enviarle solicitudes y recoger resultados.

Un “Bound Service” está corriendo únicamente mientras tenga vínculos activos.

Se pueden vincular a un servicio multiples componentes simultaneamente.

Cuando todos se desvinculan, el servicio es destruido.

Puede estar en ambos modos:

Ser lanzado para ejecutar indefinidamente y permitir vínculos.
Eso depende únicamente de qué y cómo se implmenten ciertos callbacks.

Utilizar un explicit intent para arrancar o vincularse al servicio.

Service Lifecycle

Más simple que el de una Activity.

Muy importante tener controlado cómo/cuando se crea y cómo/cuando se destruye, pues corre en background.

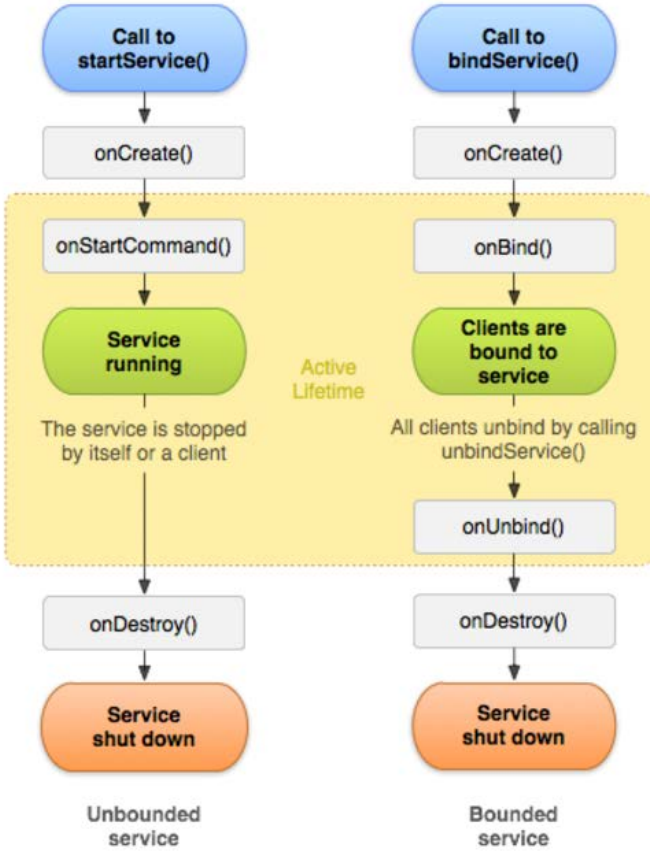
Desde que se crea hasta que se destruye puede tener dos posibilidades.

Started Service

- Se crea cuando un componente llama a startService().
- Corre indefinidamente y se para el mismo con stopSelf()
- Otro componente también puede pararlo llamando a stopService().
- Cuando el servicio se para, el sistema lo destruye.

Bound Service

- El servico se crea cuando otro componente se vincula a él con bindService().
- El cliente comunica con el servicio mediante un interface IBinder.
- El cliente puede cerrar la vinculación llamando a unbindService().
- Múltiples clientes se pueden vincular al mismo
- Cuando el último se des-vincula el sistema destruye el servicio.



Mixto: Started Servicc and Bound Service

Se puede arrancar una canción con startService(). Más adelante el usuario quiere información sobre la canción en curso y se vincula con bindService() al servicio ya creado.

En estos casos mixtos las llamadas a stopSelf() y stopService() no paran el servicio hasta que todos los clientes se hayan desvinculado.

Servicios

Como respuesta a startService() implementaremos onStartCommand()

Como respuesta a bindService() implementaremos onBind()

Privacidad

Por defecto cuando se crea un servicio es accesible por cualquier aplicación.

Podemos declarar el servicio como privado en el Manifest.

Running-thread

Un servicio corre en el main-thread del componente que lo crea.

No crea un thread propio donde correr. No corre en un proceso separado.

Un Servicio **debe** crear un thread para ejecutar operaciones pesadas en él y no sobrecargar el main-thread del creador.

Creación

Debes crear una clase que extienda de Service o de una de sus subclases

Debes sobrecargar ciertos callbacks() para gestionar el ciclo de vida del Service y permitir la vinculación.

Callbacks más importantes

onStartCommand()

Cuando llaman a startService(). Ejecución indefinida. Hay que llamar a stopSelf() o a stopService(). Si sólo queremos vincular no hace falta implementarlo.

onBind()

Cuando llaman a bindServicd(). En la implementación hay que proporcionar un interface de comunicación con el cliente, devolviendo un IBinder.

onCreate()

Cuando se crea el servicio. Inicializaciones. Antes de onStartCommand() y onBind(). Si el servicio ya está corriendo no se llama a este callback.

onDestroy()

Cuando se destruye el servicio. Implementar aquí limpieza de recursos y threads, desregistrar elementos.

Callback Methods for services

```
public class ExampleService extends Service {
    int mStartMode;       // indicates how to behave if the service is killed
    IBinder mBinder;      // interface for clients that bind
    boolean mAllowRebind; // indicates whether onRebind should be used

    @Override
    public void onCreate() {
        // The service is being created
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // The service is starting, due to a call to startService()
        return mStartMode;
    }
    @Override
    public IBinder onBind(Intent intent) {
        // A client is binding to the service with bindService()
        return mBinder;
    }
    @Override
    public boolean onUnbind(Intent intent) {
        // All clients have unbound with unbindService()
        return mAllowRebind;
    }
    @Override
    public void onRebind(Intent intent) {
        // A client is binding to the service with bindService(),
        // after onUnbind() has already been called
    }
    @Override
    public void onDestroy() {
        // The service is no longer used and is being destroyed
    }
}
```

Foreground Services

Servicios en Foreground

Un Foreground service es un servicio que tiene la misma prioridad que una Activity activa y por tanto no debe ser matado por Android, incluso si el sistema requiere recursos.

Un Foreground service debe proporcionar una OnGoing Notification para el Status bar, que significa que la notificación no puede ser eliminada a menos que el servicio se pare (stop()) o se elimine del foreground.

Por ejemplo un reproductor de múscica que ejecute la música desde un servicio debe ser colocado como un foreground service, porque el usuario está activamente pendiente de esta operación (escuchar la música).

La notificación en el Status bar puede indicarle la canción en curso y puede permitir al usuario interactuar con el reproductor.

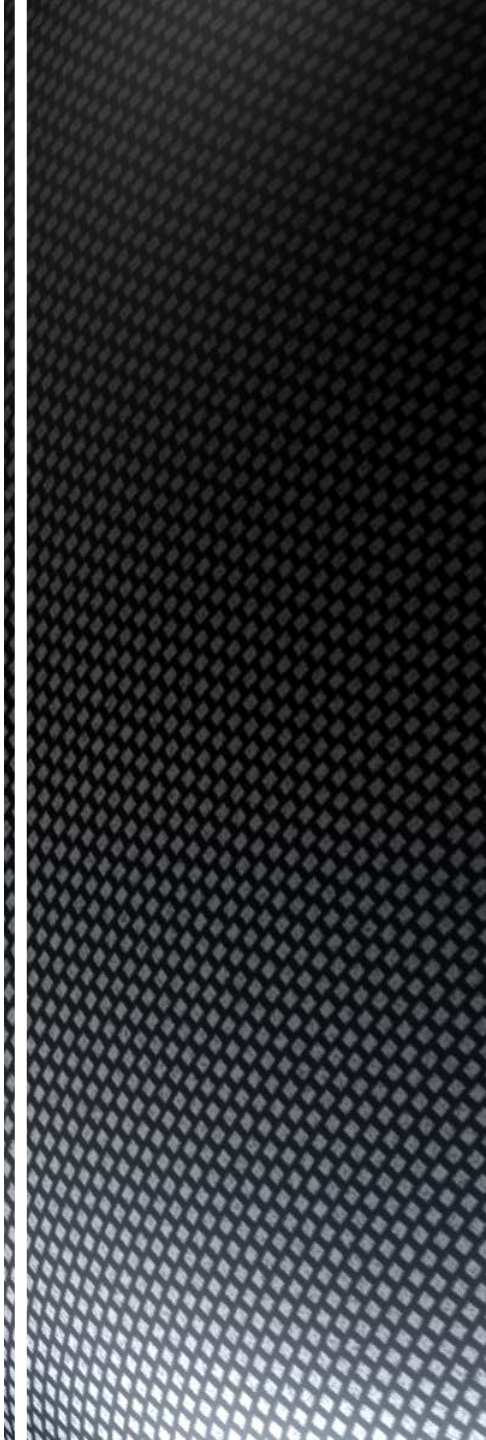
Para solicitar que el servicio corra en foreground y vincularlo con la notificación:

```
startForeground(ONGOING_NOTIFICATION_ID, notification);
```

```
int NOTIFICATION_ID = 1 //valor distinto de 0
…
Notification notification = new NotificationCompat.Builder(getApplicationContext())
                .setSmallIcon(android.R.drawable.ic_media_play)
                .setOngoing(true)
                .setContentTitle("Music Playing")
                .setContentText("Acceder al Music Player")
                .setContentIntent(pendingIntent)
                .build();

//Poner el Servidio en foreground
startForeground(NOTIFICATION_ID,notification);
```

Started Services



Creating a Started Service

Se lanza con startService() pasándole un Intent

Con el Intent se especifica el servicio y se le pasan los datos que usará el servicio.

El servicio recibe el Intent en el onStartCommand().

Ejemplo:

Una Activity necesita guardar datos en una base de datos on-line.
La Activity arranca un Servicio que hemos definido para ello.

Le adjunta los datos en el Intent pasándoselo con startService()

El servicio recibe el intent en el onStartCommand() donde conecta con Internet y realiza la transacción con la base de datos.

Cuando la transacción se ha hecho, el servicio se para a si mimso y es destruido poar el sistema.

¿ Servido o Thread ?

Un servicio es simplemente un componente que puede correr en background cuando el usuario no interacciona con la aplicación. Por eso, sólo hay que crear un servicio si se necesita.

Si hay que realizar operaciones fuera del main thread, pero sólo cuando el usuario está interactuando con la aplicación, entonces probablemente debas crear un nuevo thread y no un servicio.

> Por ejemplo, si quieres oir música, pero sólo cuando la actividad está corriendo, deberías crear un thread en el onCreate() o también puedes considerar las otras alternativas para usar threads como AsyncTask o HandlerThread.
>
> Si por el contrario, quieres que la música siga sonando cuando se cierre la Activity, deberás crear un servicio.
>
> Por supuesto, la música sonará también cuando la Activity esté en ejecución.

Recuerda que si utilizas un servicio éste no crea un thread automáticamente y por tanto la operación corre en el main-thread. Crea un thread desde el servicio si va a realizar operaciones intensivas o bloqueantes.

Creating a Started Service

Para crear un Service se puede hacer con las siguientes clases

Service

Clase base para todos los servicios.

Debes crear un thread para realizar los trabajos pesados.

IntentService

Es una subclase de Service que utiliza un thread de trabajo que gestiona una a una todas las solicitudes.

Es la mejor opción cuando no se requiere la atención simultánea de solicitudes.

Para manejar las solicitudes hay que implementar el callback onHandleIntent() que recibe el intent enviado con cada solicitud de arranque hecha con startService().

Creating a Started Service - IntentService

La clase IntentService hace lo siguiente:

- Crea un thread de trabajo (*worker-thread*) que ejecuta todos los intents enviados a onStartCommand()
- Crea una cola de trabajos que encola los intents para ser pasados en orden a tu implementación de onHandleIntent()
- Detiene el servicio cuando se han atendido todas las peticiones de arranque, por lo que no es necesario llamar a stopSelf()
- Proporciona una implementación por defecto de onBind() que devuelve null.
- Proporciona una implementación por defecto de onStartCommand() que envía el intent recibido a la cola de trabajos y a onHandleIntent()

Nosotros debemos hacer lo siguiente:

- Crear una clase que extienda IntentService
- Declarar el nuevo servicio en el Manifest.xml
- Proporcionar un constructor que llame al constructor padre usando super(String) donde String es el nombre que le queramos dar al *worker-thread*.
- Implementar el callback onHandleIntent() que es donde se hará el trabajo.

E1201_LoggingService

Creating a Started Service - IntentService

```
public class MyIntentService extends IntentService {
  public MyIntentService() {
        super("HelloI MyIntentService");
    }

  @Override
  protected void onHandleIntent(Intent intent) {
    try {
         Thread.sleep(5000);
       } catch (InterruptedException e) {
          // Restore interrupt status.
          Thread.currentThread().interrupt();
       }
     }
}
```

Si quieres puedes implementar también los callbacks onCreate(), onStartCommand() y onDestroy()

Pero debes asegurarte de llamar a las implementaciones padre con super para que el IntentService maneje correctamente el ciclo de vida del *worker-thread*.

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show();
    return super.onStartCommand(intent,flags,startId);
}
```

El único método desde el que no es necesario llamar al super es onBind()

Pero solo lo debes implementarlo si deseas permitir la vinculación.

Creating a Started Service - Service

Implementación y declaración

Declararlo en el Manifest

```
<service
  android:name="MyService"
  android:icon="@drawable/icon"
  android:label="@string/service_name"
  >
</service>
```

Implementar una clase que extienda Service o una de sus subclases.

```
public class MyService extends Service {

  @Override
  public int onStartCommand(Intent intent, int flags, int startId) {
    //TODO do something useful
    return Service.START_NOT_STICKY;
  }

  @Override
  public IBinder onBind(Intent intent) {
  //TODO for communication return IBinder implementation
    return null;
  }
}
```

Creating a Started Service - Service

Lanzar el servicio

Un componente (servicio,receiver,activity) puede lanzar el servicio via startService(intent)

```
// use this to start and trigger a service
Intent i= new Intent(context, MyService.class);
// potentially add data to the intent
i.putExtra("KEY1", "Value to be used by the service");
context.startService(i);
```

Parar el servicio

Con stopService() desde cualquier componente o con stopSelf() desde el propio servicio.

Es suficiente una única llamada aunqeu se haya llamado varias veces a startService()

Comienzo de ejecución

Android lanza el onCreate() del servicio

Una vez creado, lanza el onStartCommand(intent), que recibe el intent.

Si startService(intent) es llamado mientras está en ejecución el servicio, onStartCommand(intent) es vuelto a llamar, pero no onCreate().

El servicio debe estar preparado para reentradas de onStartCommand(intent).

Comportamiento de Restart

onStartCommand(intent) debe retornar un int que define la forma de Restart en caso de que el servicio sea matado por Android. Las constantes más comunes son:

Option	Description
Service.START_STICKY	Service is restarted if it gets terminated. Intent data passed to the `onStartCommand` method is null. Used for services which manages their own state and do not depend on the `Intent` data.
Service.START_NOT_STICKY	Service is not restarted. Used for services which are periodically triggered anyway. The service is only restarted if the runtime has pending `startService()` calls since the service termination.
Service.START_REDELIVER_INTENT	Similar to Service.START_STICKY but the original `Intent` is re-delivered to the `onStartCommand` method.

¿¿Restarted??

El método Intent.getFlags() recibe:

- START_FLAG_REDELIVERY (si lanzado con START_REDELIVER_INTENT)
- START_FLAG_RETRY (si lanzado con START_STICKY)

Creating a Started Service - Service

La clase Service no hace nada por nosotros, lo tenemos que hacer nosotros.

Deberemos sobrecargar los métodos onCreate() y onStartCommand() como mínimo.

Un Started Service debe manejar su propio ciclo de vida.

Por tanto debe ser capaz de terminarse o dejar que lo terminen.

Debe poder terminarse el mismo con stopSelf()

Otro componente de nuestra aplicación puede llamar a StopService()

El método onStartCommand(Intent intent, int flags, int startid) nos pasa startid que es un ID de la solicitud de ejecución actual del servicio.

Para evitar problemas con multiples llamadas a startService() que generan entradas del callback onStartCommand() es bueno llamar a stopSelf(**Int**) pasándole el startid que recibimos, de forma que no matemos una nueva llamada que entró antes de poder cerrar la anterior.

E1202_ServiceMusicPlayer