Desarrollo Android

Arkaitz Garro

Master en desarrollo de Apps Móviles

Servicios

Introducción

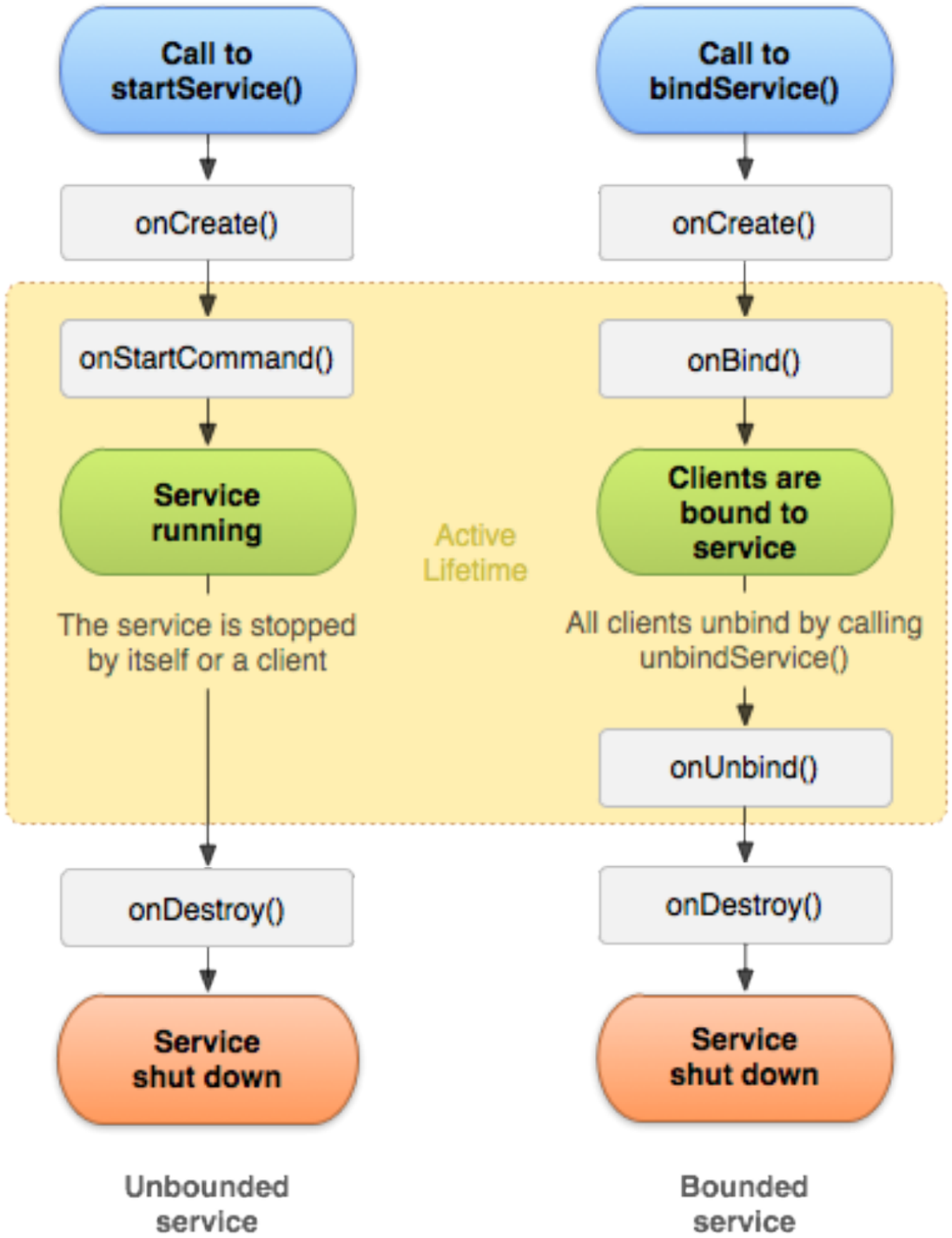
Introducción

A diferencia de las actividades, los servicios se ejecutan de manera invisible, en segundo plano.

Los servicios son iniciados, parados y controlados por otros componentes de la aplicación.

Los servicios en ejecución tienen mayor prioridad que las actividades paradas e inactivas. Además, pueden configurarse para que vuelvan a iniciarse.

Ciclo de vida



Crear servicios

Extender la clase `Service` e implementar los métodos `onCreate` y `onBind`

```
public class MyService extends Service {

  @Override
  public void onCreate() {
    // TODO: Actions to perform when service is created.
  }

  @Override
  public IBinder onBind(Intent intent) {
    // TODO: Replace with service binding implementation.
    return null;
  }
}
```

Crear servicios

Registrar el servicio en el manifiesto

```
<service android:enabled=”true” android:name=”.MyService”/>
```

Asegurar el servicio

```
<service android:enabled=”true”
         android:name=”.MyService”
         android:permission=”com.paad.MY_SERVICE_PERMISSION”/>
```

Ejecutar un servicio

Implementar el método `onStartCommand` para iniciar las tareas. Es posible especificar el comportamiento de reinicio en este método.

Este método es llamado cada vez que se inicia el servicio, así que puede ser ejecutado varias veces durante el ciclo de vida del servicio.

El procesado dentro de `onStartCommand` afecta al hilo principal, por lo que es conveniente crear un nuevo hilo para ejecutar las operaciones

Ejecutar un servicio

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    startBackgroundTask(intent, startId);

    return Service.START_STICKY;
}
```

Reiniciar un servicio

Los posibles comportamientos son:

`START_STICKY:` el servicio será reiniciado si ha sido terminado por Android. En este caso, el nuevo **`Intent`** será **`null`**. Es el modo común a utilizar para servicios los cuales controlamos su inicio y fin.

`START_NOT_STICKY:` específico para servicios con un propósito concreto, y que se terminan con **`stopSelf`**. Si ha sido terminado por Android, el servicio únicamente se reinicia si hay llamadas a **`startService`** pendientes de procesar.

`START_REDELIVER_INTENT:` es una combinación de los anteriores. El servicio sólo se reinicia si llamadas pendientes **o** el proceso se ha terminado antes de ejecutar **`stopSelf`**.

Iniciar un servicio

Al igual que en las actividades, los servicios se pueden iniciar de manera implícita o explícita.

```
private void explicitStart() {
    // Explicitly start My Service
    Intent intent = new Intent(this, MyService.class);

    // TODO Add extras if required.
    startService(intent);
}

private void implicitStart() {
    // Implicitly start a music Service
    Intent intent = new Intent(MyMusicService.PLAY_ALBUM);
    intent.putExtra(MyMusicService.ALBUM_NAME_EXTRA, "United");
    intent.putExtra(MyMusicService.ARTIST_NAME_EXTRA, "Pheonix");

    startService(intent);
}
```

Parar un servicio

Al igual que para iniciar un servicio, para pararlo debemos especificar un `Intent` con la información necesaria.

```
private void stopServices() {
    // Stop a service explicitly.
    stopService(new Intent(this, MyService.class));

    // Stop a service implicitly.
    Intent intent = new Intent(MyMusicService.PLAY_ALBUM);
    stopService(intent);
}
```

Auto-destrucción de servicios

Debido a la alta prioridad de los servicios, no es muy común que sean terminados por Android, por lo que es una buena práctica pararlos siempre que sea posible.

Cuando el servicio ha terminado sus acciones o procesos, puede terminarse llamando a `stopSelf`.

Puede llamarse a este método sin parámetros (forzar el cierre) o con el parámetro `startId`, para confirmar que el proceso se ha completado.

Enlazar con actividades

Los servicios se pueden enlazar con actividades, de manera que sea posible realizar llamadas al servicio como si se tratase de cualquier otra instancia.

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

public class MyBinder extends Binder {
    MyMusicService getService() {
      return MyMusicService.this;
    }
}

private final IBinder binder = new MyBinder();
```

Enlazar con actividades

En la actividad, es necesario implementar un `ServiceConnection`, para obtener la referencia al servicio que estamos ejecutando.

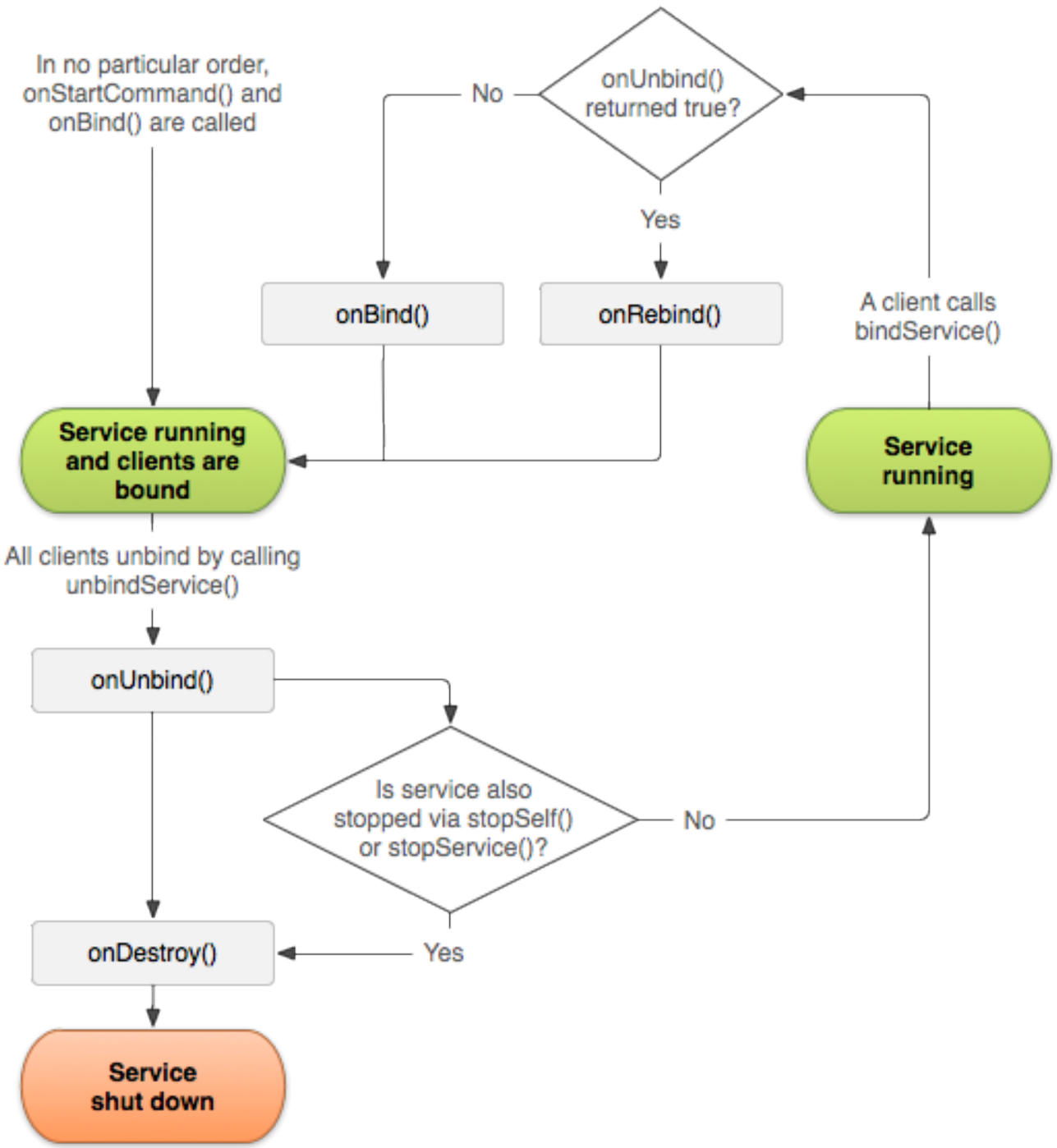
```
//Reference to the service
private MyMusicService serviceRef;

//Handles the connection between the service and activity
private ServiceConnection mConnection = new ServiceConnection() {
   public void onServiceConnected(ComponentName className,
                                  IBinder service) {
     // Called when the connection is made.
     serviceRef = ((MyMusicService.MyBinder)service).getService();
   }

   public void onServiceDisconnected(ComponentName className) {
     // Received when the service unexpectedly disconnects.
     serviceRef = null;
   }
};
```

Enlazar con actividades

Ya solo queda llamar al método `bindService` para completar el enlace y lanzar el servicio.

```
private void bindToService() {
    //Bind to the service
    Intent bindIntent = new Intent(MyActivity.this, MyMusicService.class);
    bindService(bindIntent, mConnection, Context.BIND_AUTO_CREATE);
}
```

Ciclo de vida

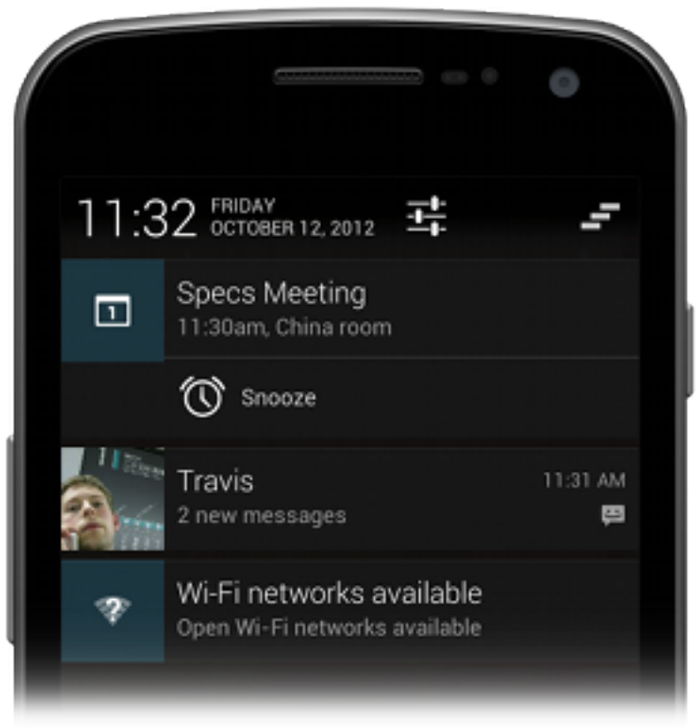


Ejercicio

Servicios en primer plano

En los casos en los que el servicio interactúe directamente con el usuario, podemos ejecutar los servicios con máxima prioridad.

Éste modo de ejecución debe especificar una notificación, que durará el tiempo que el servicio esté ejecutándose en primer plano (por ejemplo, reproduciendo música).



Servicios en primer plano

```
private void startPlayback(String album, String artist) {
    int NOTIFICATION_ID = 1;

    // Create an Intent that will open the main Activity
    // if the notification is clicked.
    Intent intent = new Intent(this, MyActivity.class);
    PendingIntent pi = PendingIntent.getActivity(this, 1, intent, 0);

    // Set the Notification UI parameters
    Notification notification = new Notification(R.drawable.icon,
      "Starting Playback", System.currentTimeMillis());
    notification.setLatestEventInfo(this, album, artist, pi);

    // Set the Notification as ongoing
    notification.flags = notification.flags |
                         Notification.FLAG_ONGOING_EVENT;

    // Move the Service to the Foreground
    startForeground(NOTIFICATION_ID, notification);
}
```

Servicios en primer plano

Cuando el servicio ya no requiere alta prioridad, lo mejor es moverlo de nuevo a segundo plano. La notificación asociada será terminada automáticamente.

```
public void pausePlayback() {
    // Move to the background and remove the Notification
    stopForeground(true);
}
```

Servicios en primer plano

Tener un servicio en primer plano, hace prácticamente imposible terminarlo, y por lo tanto liberar recurso.

Utilizar esta técnica únicamente si es estrictamente necesario.

Alarmas

Introducción

Son un mecanismo para lanzar Intents en determinados momentos o intervalos.

A diferencia de los Timer, funcionan también cuando la aplicación no está funcionando.

Son muy útiles en combinación con Broadcast Receivers, ya que podemos lanzar Intents, iniciar servicios y actividades, aunque la aplicación no esté en marcha.

Crear, ajustar y cancelar alarmas

Para crear una alarma de un único uso, llamar al método set especificando el tipo de alarma, la hora y el Intent a lanzar.

Existen cuatro tipos de alarmas:

`RTC_WAKEUP:` activa el dispositivo en la hora indicada, y lanza el Intent.

`RTC:` lanza el Intent a la hora indicada, pero sin activar el dispositivo.

`ELAPSED_REALTIME:` tiene en cuenta el tiempo desde que el dispositivo se inició. No activa el dispositivo.

`ELAPSED_REALTIME_WAKEUP:` tiene en cuenta el tiempo desde que el dispositivo se inició. Activa el dispositivo.

Crear, ajustar y cancelar alarmas

```
private void setAlarm() {
    // Get a reference to the Alarm Manager
    AlarmManager alarmManager =
     (AlarmManager)getSystemService(Context.ALARM_SERVICE);

    // Set the alarm to wake the device if sleeping.
    int alarmType = AlarmManager.ELAPSED_REALTIME_WAKEUP;

    // Trigger the device in 10 seconds.
    long timeOrLengthofWait = 10000;

    // Create a Pending Intent that will broadcast and action
    String ALARM_ACTION = "ALARM_ACTION";
    Intent intentToFire = new Intent(ALARM_ACTION);
    PendingIntent alarmIntent = PendingIntent.getBroadcast(this, 0,
      intentToFire, 0);

    // Set the alarm
    alarmManager.set(alarmType, timeOrLengthofWait, alarmIntent);

    alarmManager.cancel(alarmIntent);
}
```

Alarmas repetidas

Como las alarmas se ejecutan fuera del ciclo de vida de la aplicación, son perfectas para ejecutar tareas programadas, que no requieren de un servicio en constante ejecución

`setRepeating:` para controlar exactamente cuándo la alarma es lanzada, con un control de milisegundos.

`setInexactRepeating:` cuando la exactitud del intervalo no es importante. Android agrupará y lanzará varias alarmas a la vez, reduciendo el consumo de la batería.

Alarmas repetidas

```
private void setInexactRepeatingAlarm() {
    //Get a reference to the Alarm Manager
    AlarmManager alarmManager =
    (AlarmManager)getSystemService(Context.ALARM_SERVICE);

    //Set the alarm to wake the device if sleeping.
    int alarmType = AlarmManager.ELAPSED_REALTIME_WAKEUP;

    //Schedule the alarm to repeat every half hour.
    long timeOrLengthofWait = AlarmManager.INTERVAL_HALF_HOUR;

    //Create a Pending Intent that will broadcast and action
    String ALARM_ACTION = "ALARM_ACTION";
    Intent intentToFire = new Intent(ALARM_ACTION);
    PendingIntent alarmIntent = PendingIntent.getBroadcast(this, 0,
     intentToFire, 0);

    //Wake up the device to fire an alarm in half an hour, and every
    //half-hour after that.
    alarmManager.setInexactRepeating(alarmType,
                                     timeOrLengthofWait,
                                     timeOrLengthofWait,
                                     alarmIntent);
}
```