

Programação Para Dispositivos Móveis I

SERVICES

2024/_25 CTeSP – Desenvolvimento para a Web e Dispositivos Móveis

Ricardo Barbosa , rmb@estg.ipp.pt

Carlos Aldeias, cfpa@estg.ipp.pt

Adaptação do conteúdo dos slides de João Ramos jrmr@estg.ipp.pt e Fábio Silva fas@estg.ipp.pt

Índice

- Services;
- Tipos de Services;
- Ciclo de Vida;
- Unbounded Services;
- Bounded Services;
- Leitura Adicional.

Existem algumas operações que queremos que continuem a ser executadas, independentemente da aplicação que temos ativa de momento.

Por exemplo, se iniciarmos o download de um ficheiro, não queremos que esse processo seja interrompido apenas porque trocamos de aplicação.

Services

O que são?

Componente de uma aplicação Android (tal como uma Activity) que pode controlar **operações demoradas em background, não possui interface gráfica e tem um ciclo de vida simplificado.**

- Utilizado por outros componentes da aplicação e pode ficar em execução mesmo que o utilizador saia da aplicação;
 - Ex. leitor de música ou download de um ficheiro.
- Deve ser independente dos componentes que o utilizam, de forma a poder interagir com novos componentes;
- Pode lançar e controlar uma nova Thread ou AsyncTask se assim for pretendido.

Services vs Threads

Um serviço é simplesmente um componente que pode funcionar em segundo plano, mesmo quando o utilizador não está a interagir com a aplicação, por isso só deve ser criado se for essa a intenção.

Se tiver de executar uma tarefa fora da UI Thread, mas apenas enquanto o utilizador estiver a interagir com a aplicação, deverá, em vez disso, criar uma nova thread no contexto de outro componente da aplicação.

Ex. se quisermos tocar alguma música, mas apenas enquanto a aplicação estiver em execução.

Services

Tipos de Services

- **Started Services:** pode funcionar em segundo plano indefinidamente, mesmo quando a Activity que o iniciou é destruída. Uma vez que a operação é feita, o serviço para. (Ex. download de ficheiro);
- **Bound Services:** está ligado a outra componente de aplicação, tal como uma Actividade, através do método `bindService()`. A Atividade pode interagir com ele, enviar pedidos, e obter resultados. Um Bound Service funciona desde que os componentes estejam vinculados a ele. Quando os componentes já não estão vinculados, o serviço é destruído. (Ex. Odómetro para medir a distância percorrida por um veículo);
- **Schedule Services:** é um serviço que está programado para funcionar num determinado momento temporal.

Services

Started Service (Background)

Um background service realiza uma operação que não é diretamente perceptível pelo utilizador.

- Invocado por um componente (por ex., uma Activity) através do método `startService()`;
- Depois de inicializado o serviço pode estar em **background indefinidamente**, mesmo que componente que o invocou seja destruído;
- Normalmente é utilizado para realizar **uma operação única**, que não devolve um resultado a quem o invocou;
- **É gerido pelo sistema** em caso de falta de memória ou bateria.

Services

Started Service (Foreground)

Um serviço em foreground realiza alguma operação que é perceptível para o utilizador. Estes serviços devem exibir uma notificação para que os utilizadores estejam ativamente conscientes de que o serviço está em execução (esta notificação não pode ser rejeitada a menos que o serviço seja interrompido ou removido do primeiro plano), e continuam a funcionar mesmo quando o utilizador não está a interagir com a aplicação.

- Invocado por um componente (Ex. uma Activity) através do método `startService()`;
- Depois de inicializado o serviço pode estar em background **indefinidamente**, mesmo que componente que o invocou seja destruído;
- Normalmente é utilizado para executar **ações que devem ser perceptíveis** pelo utilizador;
- **Não é gerido pelo sistema** em caso de falta de memória ou bateria.

Services

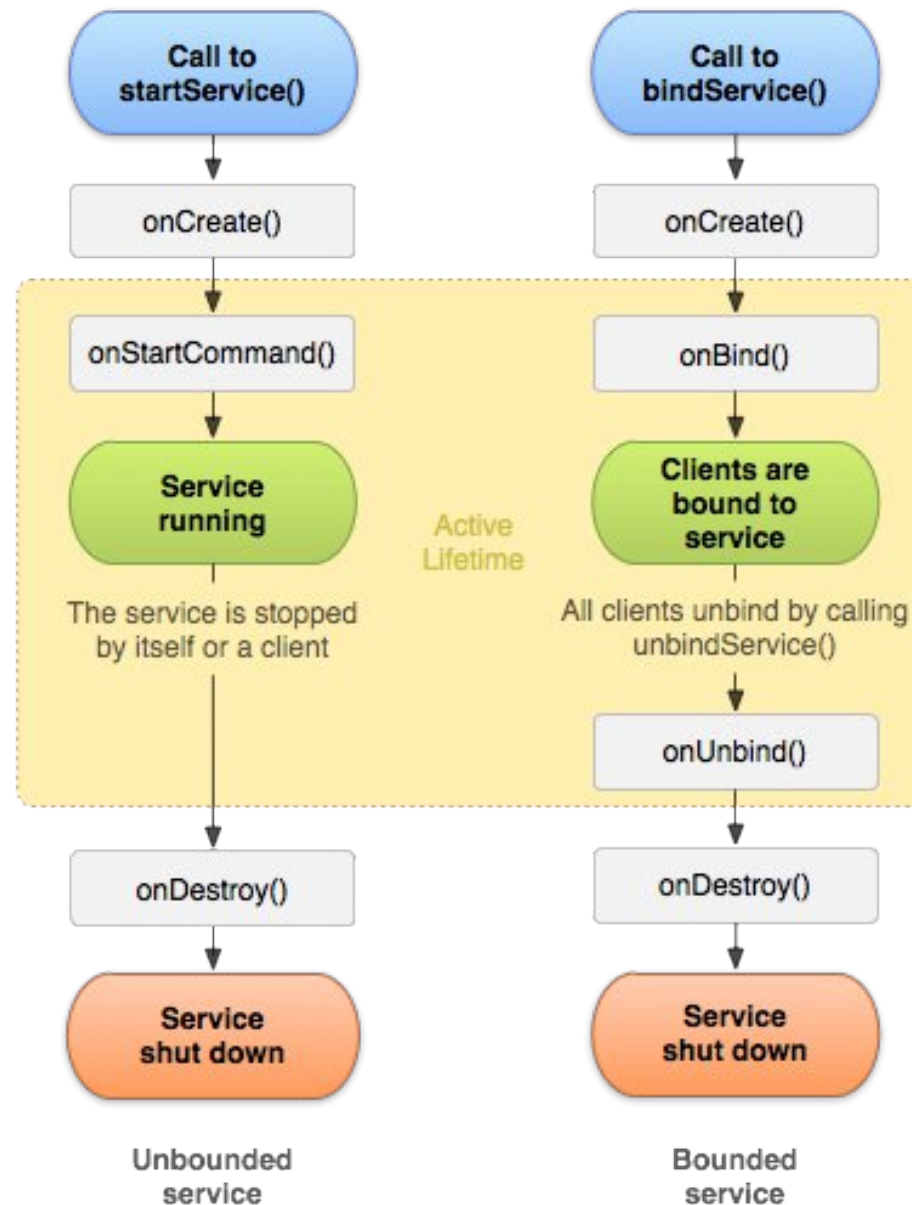
Bound Services

Oferece uma interface cliente-servidor que permite aos componentes interagir com o serviço, enviar pedidos, receber resultados, e mesmo fazê-lo através de processos com comunicação interprocessada (IPC). Um serviço vinculado funciona apenas enquanto outro componente da aplicação estiver vinculado. Múltiplos componentes podem ligar-se ao serviço de uma só vez, mas quando todos eles se desvinculam, o serviço é destruído.

- Invocado por um componente através do método `bindService()`;
- Interface tipo cliente-servidor, que permite aos componentes interagir com o service;
- Existe apenas enquanto houver pelo menos um componente conectado ao mesmo;
- Vários componentes podem estar conectados ao serviço de cada vez, mas quando todos eles fazem o `unbind()`, o service é destruído.

Services

Ciclo de Vida



Services

Ciclo de Vida (Unbounded)

`startService(Intent)`

- Método utilizado na Activity para iniciar um Service;

`onCreate()`

- Executado durante a criação do Service;

`onStartCommand()`

- Invocado quando o Service é utilizado no modo **Started** (através do método `startService()`);
 - O Intent recebido representa um comando dado ao Service.

Services

Ciclo de Vida [onStartCommand()]

Devolve uma das flags com diferentes comportamentos caso o serviço seja destruído após a execução deste método:

- **START_NOT_STICKY**
 - Não recria o serviço, a menos que hajam Intents pendentes de entrega. Esta é a opção mais segura para evitar correr o serviço quando não é necessário, e quando a sua aplicação pode simplesmente reiniciar quaisquer trabalhos inacabados.
- **START_STICKY**
 - o serviço é recriado mas o Intent recebido no onStartCommand não será recebido novamente;
- **START_REDELIVER_INTENT**
 - o service é recriado e o onStartCommand irá receber o último Intent recebido antes do serviço ser destruído.

Services

Ciclo de Vida (Bounded)

onBind()

- Invocado quando o Service é utilizado no modo Bound (através do método `bindService()`);
- Devolve uma interface de comunicação com o Service (`IBinder`);

onUnbind()

- Invocado quando o Service é utilizado no modo Bound (através do método `unbindService()`);
- Devolve `true` ou `false` conforme permite ou não novos bindings (`rebind`);

onDestroy()

- Executado durante a destruição do Service.

Services

Declaração [AndroidManifest.xml]

Os serviços devem ser declarados/registados no Manifest da aplicação

```
<manifest ... >  
  ...  
  <application ... >  
    <service android:name=".ExampleService"  
             android:exported="false" />  
    ...  
  </application>  
</manifest>
```

O nome tem um ponto de forma a que o Android possa combinar com o package name

Este atributo define se o serviço pode ser utilizado por outras Aplicações

Services

Unbounded [ExampleService.java]

```
public class ExampleService extends Service {  
  
    public static final String EXTRA_MESSAGE = "message";  
  
    private final String CHANNEL_ID = "SERVICE_NOTIFICATION";  
    private final int NOTIFICATION_ID = 123;  
  
    @Override  
    public void onCreate() {  
        super.onCreate();  
        createNotificationChannel();  
    }  
  
    @Nullable  
    @Override  
    public IBinder onBind(Intent intent) {  
        return null;  
    }  
}
```


Utilizados pela
notification



Criação de um notification channel para
mostrar uma notificação



Se não tencionarmos criar um Bound
Service, retornamos null no método que
faz o Bound do serviço



Services

Unbounded [ExampleService.java]

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    synchronized (this){
        try {
            wait(10000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            String text = intent.getStringExtra(this.EXTRA_MESSAGE);
            showText(text);
        }
    }

    return START_NOT_STICKY;
}
```

Neste exemplo vamos esperar 10 segundos e depois notificar o utilizador

Método que criamos para notificar o utilizador

Se o serviço for destruído, não é recriado

Services

Unbounded [ExampleService.java]

```
private void showText(String text) {  
    NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)  
        .setSmallIcon(android.R.drawable.sym_def_app_icon)  
        .setContentTitle(getString(R.string.question))  
        .setContentText(text)  
        .setPriority(NotificationCompat.PRIORITY_HIGH)  
        .setVibrate(new long[] {0,1000})  
        .setAutoCancel(true);
```

← Builder da notificação

```
    Intent actionIntent = new Intent(this, MainActivity.class);  
    PendingIntent actionPendingIntent = PendingIntent.getActivity(  
        this,  
        0,  
        actionIntent,  
        PendingIntent.FLAG_UPDATE_CURRENT);
```

← Adicionar um PendingIntent para o utilizador abrir a App
se carregar na notificação

```
    builder.setContentIntent(actionPendingIntent);
```

```
    NotificationManager notificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);  
    notificationManager.notify(NOTIFICATION_ID, builder.build());
```

← Iniciar notificação

Services

Unbounded [ExampleService.java]

Definição do Notification Channel

```
private void createNotificationChannel() {  
    if (Build.VERSION.SDK_INT ≥ Build.VERSION_CODES.O) {  
        Uri defaultSoundUri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);  
  
        String channelName = "Service Notifications";  
        String channelDescription = "Include all Service Notifications";  
        int channelImportance = NotificationManager.IMPORTANCE_DEFAULT;  
  
        NotificationChannel notificationChannel = new NotificationChannel(CHANNEL_ID, channelName, channelImportance);  
  
        notificationChannel.setDescription(channelDescription);  
        notificationChannel.enableVibration(true);  
        notificationChannel.setSound(defaultSoundUri, null);  
  
        NotificationManager notificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);  
        notificationManager.createNotificationChannel(notificationChannel);  
    }  
}
```

Services

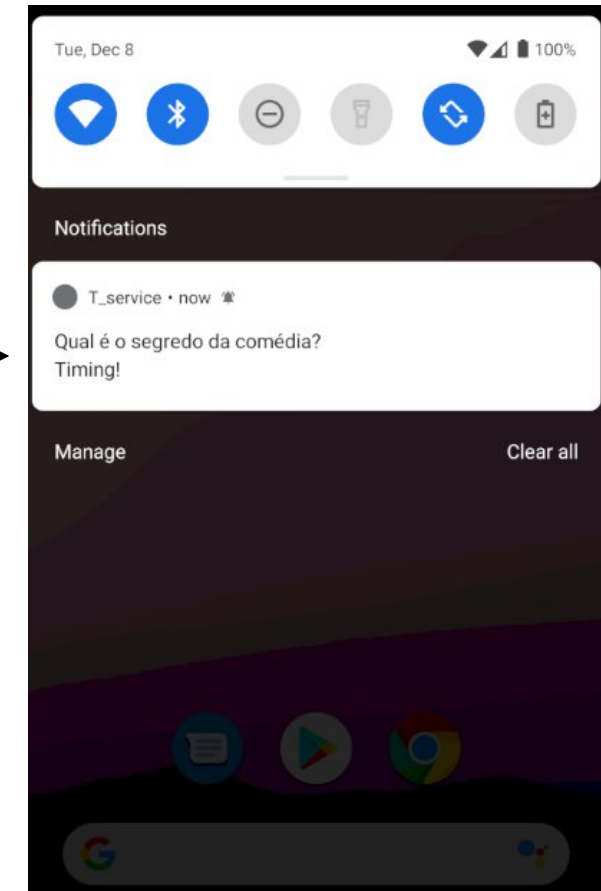
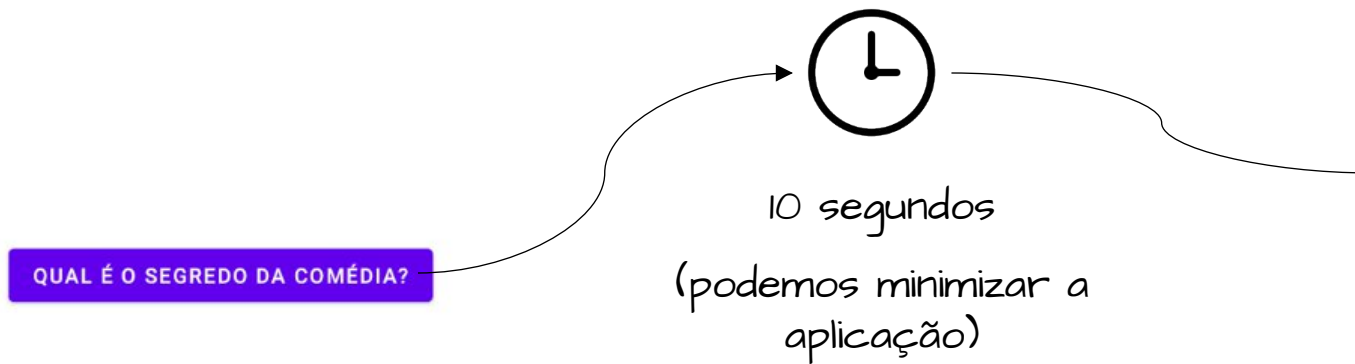
Unbounded [MainActivity.java]

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {  
  
    private Button btnService;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        btnService = findViewById(R.id.btn_service);  
        btnService.setOnClickListener(this::onClick);  
    }  
  
    @Override  
    public void onClick(View v) {  
        switch (v.getId()){  
            case R.id.btn_service:  
                Intent intent = new Intent(this, ExampleService.class);  
                intent.putExtra(ExampleService.EXTRA_MESSAGE, getResources().getString(R.string.response));  
                startService(intent);  
                break;  
        }  
    }  
}
```

← Iniciar o serviço ao pressionar o botão

Services

Unbounded [Resultado]



Services

Declaração [AndroidManifest.xml]

Os serviços devem ser declarados/registados no Manifest da aplicação

```
<manifest ... >
...
<application ... >
  <service android:name=".OdometerService"
           android:exported="false" />
...
</application>
</manifest>
```

O nome tem um ponto de forma a que o Android possa combinar com o package name

Este atributo define se o serviço pode ser utilizado por outras Aplicações

Services

Bounded [OdometerService.java]

```
public class OdometerService extends Service {
```

```
    private final IBinder binder = new OdometerBinder();  
    private final Random random = new Random();  
    private double currentDistance = 0.0;
```

Utilização do objeto Random para
simulação e geração de valores
aleatórios

```
    public class OdometerBinder extends Binder {  
        OdometerService getOdometer() {  
            return OdometerService.this;  
        }  
    }
```

Quando criamos um Bound Service
temos de fornecer uma
implementação de um Binder

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

Retornamos o Binder que
criamos

```
    public double getDistance() {  
        currentDistance += random.nextDouble();  
        return currentDistance;  
    }  
}
```

Método para obter a distância
percorrida

Services

Bounded [MainActivity.java]

```
public class MainActivity extends AppCompatActivity{

    private OdometerService odometerService;
    private boolean bound = false;

    private ServiceConnection serviceConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            OdometerService.OdometerBinder odometerBinder = (OdometerService.OdometerBinder) service;
            odometerService = odometerBinder.getOdometer();
            bound = true;
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            bound = false;
        }
    };
};
```

Criação de um objeto de ligação ao Service

Variável de controlo para percebermos o estado de vinculação do serviço

Utilizamos o IBinder para obter uma referência ao serviço

Ações a executar quando um serviço e a Activity são desconectados

Services

Bounded [MainActivity.java]

```
@Override
protected void onStart() {
    super.onStart();
    Intent intent = new Intent(this, OdometerService.class);
    bindService(intent, serviceConnection, Context.BIND_AUTO_CREATE);
}
```

Iniciamos a vinculação com o

serviço

```
@Override
protected void onStop() {
    super.onStop();
    if(bound){
        unbindService(serviceConnection);
        bound = false;
    }
}
```

Se o serviço estiver vinculado, removemos
essa vinculação

Services

Bounded [MainActivity.java]

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

    displayDistance();
}
```

Services

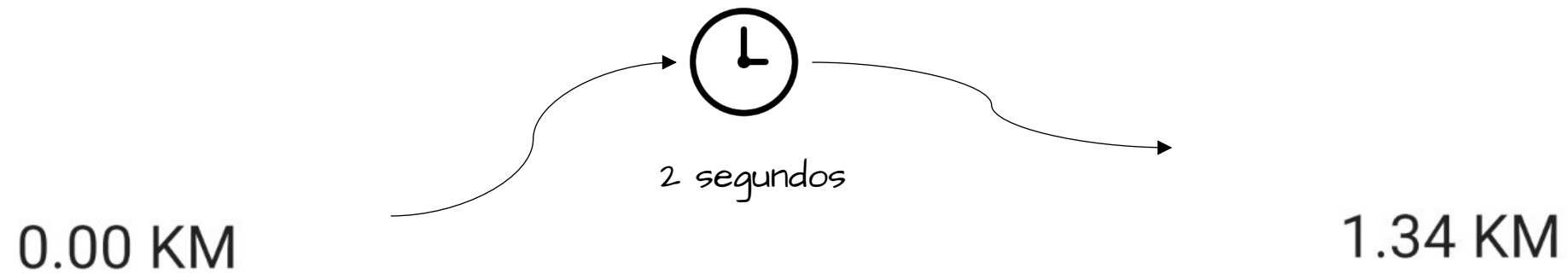
Bounded [MainActivity.java]

```
private void displayDistance(){
    final TextView distanceView = findViewById(R.id.txt_distance);
    final Handler handler = new Handler(Looper.myLooper());
    handler.post(new Runnable() {
        @Override
        public void run() {
            double distance = 0.0;
            if(bound && odometerService != null){
                distance = odometerService.getDistance();
            }
            String distanceStr = String.format(Locale.getDefault(), "%1$, .2f KM", distance);
            distanceView.setText(distanceStr);
            handler.postDelayed(this, 2000);
        }
    });
}
```

Se tivermos uma referência do serviço e estivermos vinculados, chamamos o método

Services

Bounded [Resultado]



Componentes baseados em Services

Existem na plataforma componentes que tentam simplificar o trabalho dos services

- Por exemplo para executar tarefas curtas num service podemos usar o `JobIntentService`

<https://developer.android.com/reference/androidx/core/app/JobIntentService>

- Para facilitar o processamento paralelo podemos usar `Loopers` e `ServiceHandlers` em conjunto com o service, e assegurar que o trabalho dentro do service é executado fora da main thread

<https://developer.android.com/guide/components/services#ExtendingService>

Leitura Adicional

Services:

<https://developer.android.com/guide/components/services>

Threads e Services:

<https://developer.android.com/guide/components/processes-and-threads.html#Threads>

Programação Para Dispositivos Móveis I

SERVICES

2024/_25 CTeSP – Desenvolvimento para a Web e Dispositivos Móveis

Ricardo Barbosa , rmb@estg.ipp.pt

Carlos Aldeias, cfpa@estg.ipp.pt

Adaptação do conteúdo dos slides de João Ramos jrmr@estg.ipp.pt e Fábio Silva fas@estg.ipp.pt