

Programação Para Dispositivos Móveis I

THREAD, EXECUTOR SERVICE & WORKMANAGER

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

Ricardo Barbosa , rmb@estg.ipp.pt

Carlos Aldeias, cfpa@estg.ipp.pt

Índice

- Background Processing;
- Thread;
- ExecutorService;
- WorkManager;
- Leitura Adicional.

Prefácio

Recordamos o conteúdo apresentado nos slides de Databases. Uma aplicação que reproduzisse os passos da implementação apresentada teria o seguinte comportamento:

1. No momento da abertura da aplicação, iria procurar por um ficheiro de base de dados;
2. Caso esse ficheiro não existisse teria de criar uma base de dados nova;
3. Depois teria de correr todas as instruções de SQL para criar as tabelas dentro da base de dados, e alguns dados iniciais;
4. Por fim teria de executar algumas queries para retirar dados da base de dados e apresentá-los ao utilizador.

Prefácio

Para uma base de dados pequena, este processo não demoraria muito tempo e seria impercetível para o utilizador. Contudo, com o constante crescimento de uma base de dados ao longo do tempo, não iria demorar muito até a aplicação tornar-se lenta, até ao ponto de ser inutilizável.

Não há muito que se possa fazer para aumentar as velocidades de escrita e leitura numa base de dados, mas **podemos impedir** que estes processos tornem a nossa aplicação mais lenta.

Threads

Desde a versão Lollipop (API 21), existem três tipos de threads que devemos considerar:

1. Main Thread (ou User Interface (UI) Thread)

Esta é a thread principal de Android. Está à escuta de intents, recebe inputs do ecrã, e procede à chamada de métodos dentro das activities.

2. Render Thread

Normalmente não existe interação com esta thread, mas é responsável por “ler” uma lista de requisitos para atualizações do ecrã, e depois instruir o hardware gráfico do dispositivo para “repintar” o ecrã e tornar a aplicação bonita.

3. Outras Threads que sejam criadas

Threads

Sem algum cuidado e controlo, a aplicação irá executar a maioria das suas tarefas na UI Thread (Main Thread).

Ao colocar todo o código associado com a base de dados (por exemplo) no método `onCreate()`, a UI Thread vai estar ocupada a comunicar com a base de dados, em vez de rapidamente procurar por eventos a partir do ecrã ou outras aplicações. Se algum conjunto de instruções relacionada com a base de dados demorar muito tempo, o utilizador irá sentir que está a ser ignorado ou irá questionar-se se a aplicação terá falhado.

O “truque” é remover estas instruções “pesadas” da UI Thread e executa-las numa Thread personalizada em background.

Background Processing

Um processo contém um ambiente de execução com **um conjunto de recursos associado** (ex. espaço de memória), necessários para a execução de instruções.

Em sistemas multiprocessamento (como é o caso do Android), a execução destes processos é feita de forma **concorrente/paralela**, dando ao utilizador a sensação de que diversas aplicações estão a ser executadas em simultâneo.

No entanto, a criação de múltiplos processos para a execução de código em paralelo facilmente se torna uma operação pesada, devido aos recursos alocados à sua criação.

Background Processing

Um processo pode ter múltiplas Threads que:

- Executam de forma concorrente/paralela;
- Partilham alguns dos recursos do processo (por ex., espaço de memória);
- São muito mais leves de criar e lançar.

Algumas operações não devem ser executadas na UI Thread, visto existir o risco de bloquearem a interação com o utilizador (congelamento da aplicação):

- Tarefas que sejam demoradas ou com um tempo de execução imprevisível;
- Operações de I/O (por ex., sobre ficheiros, comunicações de rede), que não são permitidas na UI Thread a partir da versão Android 2.3.3.

Mais em: <http://developer.android.com/guide/components/processes-and-threads.html> e <http://docs.oracle.com/javase/tutorial/essential/concurrency/>

Background Processing

Threads e Executor Services

Em Android, existem diversas formas de lançarmos novas linhas de execução paralelas (Threads):

- Classe Thread (tradicionalmente utilizada em Java);
 - Classe ExecutorService (abstração que faz parte da API Android).
-
- Na Thread, o código que executa uma nova linha de instrução encontra-se no método `run()`
 - Todo o resto, **a não ser que seja invocado a partir deste método**, executa na UI Thread.

Thread

Utilização

A definição de uma Thread é simples, sendo a sua complexidade adicional derivada da **sincronização com a UI Thread**.

```
public class DownloadThread extends Thread {  
    public DownloadThread(Context context, URL url) { ... }  
    public void run() { ... }  
}
```

Extende Thread

Método construtor

As instruções são executadas aqui

Lançamento da DownloadThread a partir da UI Thread (ex. MainActivity.java)

```
DownloadThread thread = new DownloadThread(url);  
thread.start();
```

Thread

Utilização

- Cancelamento da Thread

O mecanismo tem que ser implementado (não existe na classe Thread)

- Obtenção do estado da Thread

`Thread.State sts = thread.getState();`

Possíveis estados: NEW, RUNNABLE, BLOCKED, WAITING, TIMED_WAITING, TERMINATED;

- Esperar pela sua execução e obter o resultado

`thread.join();`

Thread

Utilização

- Execução de uma operação na UI Thread a partir de outra Thread

A UI Thread é obtida através do Contexto da aplicação

```
Handler mainHandler = new Handler(context.getMainLooper());
```

```
Runnable postRunnable = new Runnable() {  
    public void run() { ... }  
}
```

O método run() irá ser executado na UI Thread

```
mainHandler.post(postRunnable);
```

A instância de Runnable é enviada para a UI Thread

ExecutorServices

Os ExecutorServices permitem a criação de uma *pool* de *worker threads* (thread pool) onde diferentes tarefas podem ser executadas de forma concorrente.

Lançam uma nova Thread e possuem métodos específicos para a execução de instruções na UI Thread.

Executor Services

Implementação

Cria uma única thread

```
ExecutorService executorService = Executors.newSingleThreadExecutor(AVAILABLE_CORES);
```

```
private static final int AVAILABLE_CORES = Runtime.getRuntime().availableProcessors();
```

```
ExecutorService executorService = Executors.newFixedThreadPool(AVAILABLE_CORES);
```

Cria uma Thread Pool com n threads

Podemos utilizar o número de cores disponíveis para definir o total de threads.

1 core = 1 thread

Executor Services

Implementação

```
private void runBackgroundRunnable() {  
    executorService.execute(new Runnable() {  
        @Override  
        public void run() {  
            ( ... )  
            runOnUiThread(new Runnable() {  
                ( ... )  
            });  
        }  
    });  
}
```

← Todo o código que deverá executado pela thread está no método run()

← É utilizado o método runOnUiThread para poder comunicar com a Main Thread.
Utilizado para, por exemplo, atualizar Views.

Executor Services

Implementação

```
private void runBackgroundRunnable() {  
    executorService.execute(new Runnable() {  
        @Override  
        public void run() {  
            try {  
                URL imageURL = new URL(String.valueOf(inputExternalURL.getText()));  
                HttpURLConnection httpURLConnection = (HttpURLConnection) imageURL.openConnection();  
                httpURLConnection.connect();  
                resultPicture = BitmapFactory.decodeStream(httpURLConnection.getInputStream());  
                Log.d("ImageURL :: ", imageURL.toString());  
            } catch (IOException | RuntimeException exception) {  
                Log.e("BitmapFactory Exception :: ", exception.getMessage());  
                executorService.shutdown();  
            }  
            runOnUiThread(new Runnable() {  
                @Override  
                public void run() {  
                    externalPictureView.setImageBitmap(resultPicture);  
                }  
            });  
        }  
    });  
}
```

A thread irá realizar o download de uma imagem a partir de um URL fornecido pelo utilizador

Conversão do resultado para um objeto Bitmap (imagem)

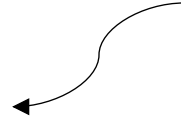
Atualizar a ImageView com a imagem que foi obtida a partir do URL.

Executor Services

Implementação

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

Não esquecer de terminar a execução sempre que existirem interrupções na Thread.



Executor Services (Future)

[BitmapDownloadTask.java]

```
public class BitmapDownloadTask implements Callable<Bitmap> {  
  
    private static final String THREAD_TAG = "CURRENT THREAD";  
    private final String userUrl;  
    private Bitmap resultPicture;  
  
    public BitmapDownloadTask(String userUrl) {  
        this.userUrl = userUrl;  
    }  
  
    @Override  
    public Bitmap call() {  
        Log.d(THREAD_TAG, Thread.currentThread().getName());  
        try {  
            URL imageURL = new URL(userUrl);  
            HttpURLConnection httpURLConnection = (HttpURLConnection) imageURL.openConnection();  
            httpURLConnection.connect();  
            resultPicture = BitmapFactory.decodeStream(httpURLConnection.getInputStream());  
        } catch (IOException exception) {  
            Log.e("BitmapDownloadTask Exception :: ", exception.getMessage());  
            executorService.shutdown();  
        }  
  
        return resultPicture;  
    }  
}
```

Método onde deverá ser incluído o código a ser executado numa nova Thread.

Informação sobre a Thread selecionada para correr este método

Em vez de manipular a Main Thread, vamos apenas retornar o resultado

Executor Services (Future)

[MainActivity.java]

Criação do objeto Future, que irá conter o resultado do pedido assíncrono.

```
Future<Bitmap> resultExternalPicture = null;
```

@Override

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

Envio da "nova tarefa" para ser executada pela Thread Pool

```
    resultExternalPicture = executorService.submit(  
        new BitmapDownloadTask(String.valueOf(inputExternalURL.getText())));
```

```
}
```

O retorno da função Call() será armazenado neste objeto.

Executor Services (Future)

[MainActivity.java]

Verificar se o pedido executado
pela Thread já foi concluído

```
if(resultExternalPicture != null && resultExternalPicture.isDone()){  
    try {  
        Bitmap resultExternalPicture = this.resultExternalPicture.get();  
        externalPictureView.setImageBitmap(resultExternalPicture);  
    } catch (ExecutionException | InterruptedException exception) {  
        Log.e(BITMAP_TAG, exception.getMessage());  
    }  
}
```

Obtemos o conteúdo do Future
Object (que estaria a guardar o
resultado do pedido assíncrono).

Executor Services (CompletableFuture)

[MainActivity.java]

@Override

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

Envio da "nova tarefa" para ser executada pela Thread Pool

```
CompletableFuture<Void> completableFuture = CompletableFuture.supplyAsync(  
    () -> resultExternalPicture = executorService.submit(new BitmapDownloadTask(DEFAULT_URL)))  
    .thenAccept(resultPicture -> {  
        try {  
            externalPictureView.setImageBitmap(resultPicture.get());  
        } catch (ExecutionException | InterruptedException exception) {  
            throw new RuntimeException(exception);  
        }  
    });  
});
```

Depois da tarefa estar concluída, podemos utilizar o resultado da mesma para interagir com a Main Thread, ou para encadear novos processos concorrentes.

WorkManager

Componente integrante do Android Jetpack que permite a **execução fiável de tarefas** em background. (Também permite a execução de tarefas fora da UI Thread)

- Permite configurar condições de execução:
 - Apenas quando o dispositivo estiver livre;
 - Apenas quando a carregar;
 - (...)
- Permite encadear tarefas.

WorkManager

build.gradle (module)

```
dependencies {  
    implementation "androidx.work:work-runtime:2.10.0"  
    ...  
}
```

WorkManager

MyWorker.java

```
public class MyWorker extends Worker {  
  
    public MyWorker(@NonNull Context context, @NonNull WorkerParameters workerParams) {  
        super(context, workerParams);  
    }  
  
    @NonNull  
    @Override  
    public Result doWork() {  
  
        //Obter input  
        getInputData().getString("KEY");  
  
        //TODO: realizar tarefa  
  
        //Criar output  
  
        Data outputData = new Data.Builder()  
            .putString("KEY", "VALUE")  
            .build();  
  
        return Result.success();  
    }  
}
```


WorkManager

Execução [ex. MainActivity.java]

Também pode ser utilizado um *PeriodicWorkRequest*



```
OneTimeWorkRequest workRequest = new OneTimeWorkRequest.Builder(MyWorker.class)
                                   .build();

WorkManager.getInstance(this).enqueue(workRequest);
```

WorkManager


Execução com inputs [ex. MainActivity.java]

```
Data.Builder inputs = new Data.Builder()  
    .putString("KEY", "VALUE");
```

```
OneTimeWorkRequest workRequest = new OneTimeWorkRequest.Builder(MyWorker.class)  
    .setInputData(inputs.build())  
    .build();
```

```
WorkManager.getInstance(this).enqueue(workRequest);
```

Adicionamos os valores
de input



WorkManager

Definição de condições de execução [ex. MainActivity.java]

```
Constraints constraints = new Constraints.Builder()  
    .setRequiresDeviceIdle(true)  
    .setRequiresBatteryNotLow(true)  
    .build();
```

Só é executado se o dispositivo estiver em Idle (inativo)

Só é executado se não tivermos níveis baixos de bateria

```
OneTimeWorkRequest compressionWorker = new OneTimeWorkRequest.Builder(MyWorker.class)  
    .setConstraints(constraints)  
    .build();
```

```
WorkManager.getInstance(this).enqueue(compressionWorker);
```

Adicionamos as condições de execução

WorkManager

Consultar o estado da classe Worker[ex. MainActivity.java]

```
WorkManager.getInstance(this).getWorkInfoByIdLiveData(workRequest.getId())
    .observe(this, new Observer<WorkInfo>() {
        @Override
        public void onChanged(WorkInfo workInfo) {
            if(workInfo != null && workInfo.getState() == WorkInfo.State.SUCCEEDED){
                String output = workInfo.getOutputData().getString("KEY");
            }
        }
    });
```

Leitura Adicional

Background Processing:

<https://developer.android.com/guide/background>

Threads:

<https://developer.android.com/guide/background/threading>

ExecutorServices:

<https://developer.android.com/reference/java/util/concurrent/ExecutorService>

WorkManager:

<https://developer.android.com/reference/androidx/work/WorkManager?hl=en>

Programação Para Dispositivos Móveis I

THREAD, EXECUTOR SERVICE & WORKMANAGER

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