

# Unit 3 : Làm việc trong nền.

Tài liệu PDF này chứa một bức tranh tĩnh một lần về các bài học trong Unit 3: Làm việc ở hậu trường.

## Các bài học trong đơn vị này

### Bài học 7: Nhiệm vụ nền

7.1 : AsyncTask

7.2 : AsyncTask và AsyncTaskLoader

7.3 : Bộ nhận phát sóng

### Bài học 8: Kích hoạt, lập lịch và tối ưu hóa tác vụ nền

8.1 : Thông báo

8.2 : Trình quản lý báo động

8.3 : Lịch trình công việc

## Bài học 7.1: AsyncTask

### Giới thiệu

Một luồng là một con đường thực thi độc lập trong một chương trình đang chạy. Khi một chương trình Android được khởi động, hệ thống sẽ tạo ra một luồng chính, còn được gọi là luồng UI. Luồng UI này là cách mà ứng dụng của bạn tương tác với các thành phần từ bộ công cụ UI của Android.

Đôi khi, một ứng dụng cần thực hiện các tác vụ tốn tài nguyên như tải xuống tệp, thực hiện truy vấn cơ sở dữ liệu, phát media hoặc tính toán phân tích phức tạp. Loại công việc tốn tài nguyên này có thể làm chậm luồng UI khiến ứng dụng không phản hồi với đầu vào của người dùng hoặc không hiển thị trên màn hình. Người dùng có thể cảm thấy thất vọng và gỡ cài đặt ứng dụng của bạn.

Để giữ cho trải nghiệm người dùng (UX) diễn ra suôn sẻ, framework Android cung cấp một lớp trợ giúp gọi là AsyncTask, lớp này xử lý công việc ngoài luồng UI. Sử dụng AsyncTask để chuyển các xử lý tốn tài nguyên sang một luồng riêng biệt có nghĩa là luồng UI có thể giữ được tính phản hồi.

Vì luồng riêng biệt không được đồng bộ hóa với luồng gọi, nên nó được gọi là luồng bất đồng bộ.

### Những gì bạn cần biết

Bạn cần có thể:

- Tạo một Hoạt động.
- Thêm một TextView vào bố cục cho Hoạt động.
- Lập trình để lấy id cho TextView và đặt nội dung của nó.
- Sử dụng chế độ xem Nút và chức năng onClick của chúng.

Những gì bạn sẽ học

- Cách thêm AsyncTask vào ứng dụng của bạn để chạy tác vụ trong nền của ứng dụng.
- Những hạn chế khi sử dụng AsyncTask cho các tác vụ nền.

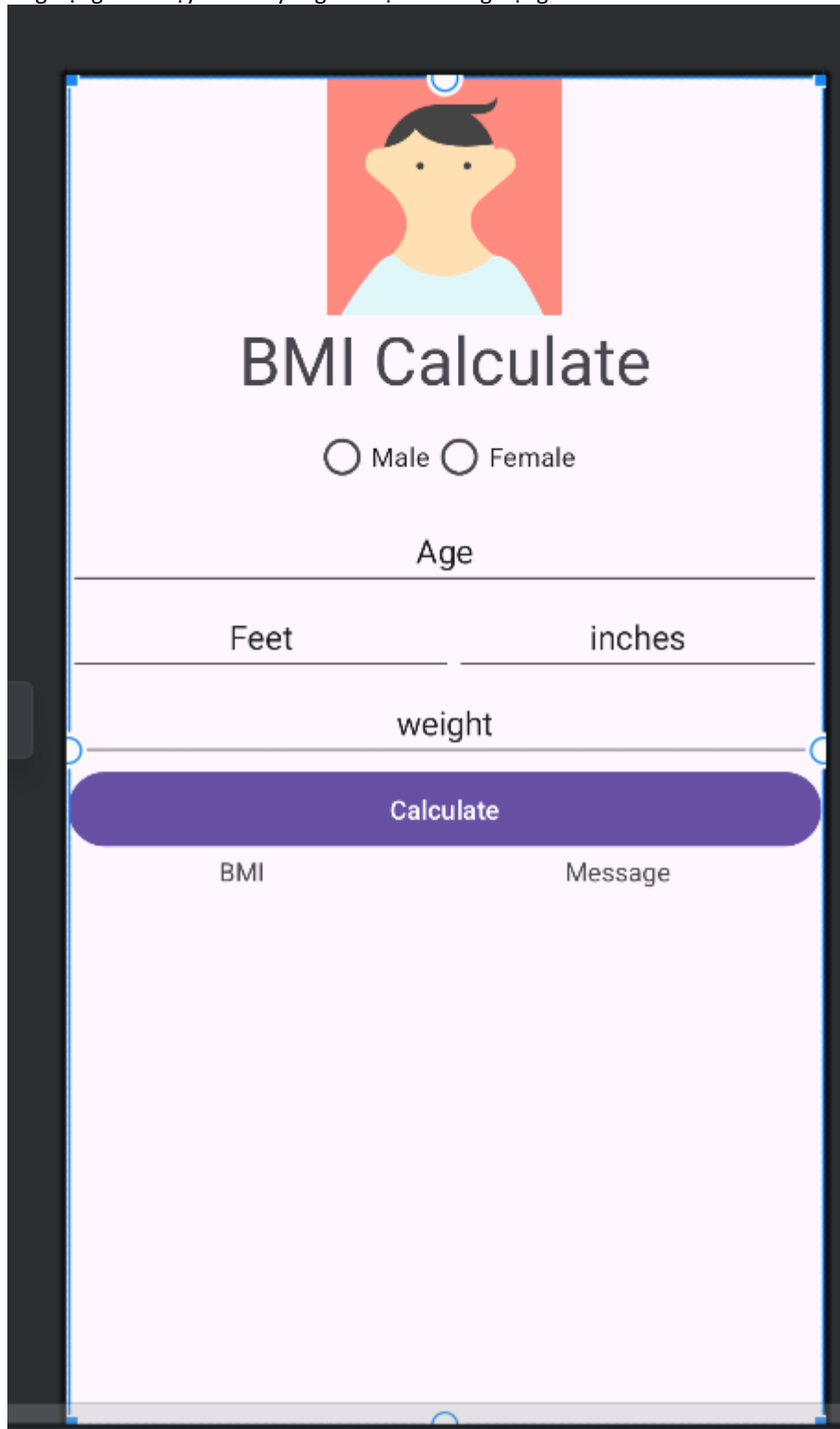
Những gì bạn sẽ làm

- Tạo một ứng dụng đơn giản thực thi tác vụ nền bằng AsyncTask.
- Chạy ứng dụng và xem điều gì xảy ra khi bạn xoay thiết bị.
- Triển khai trạng thái thể hiện hoạt động để giữ nguyên trạng thái của thông báo TextView.

### Tổng quan về ứng dụng

Bạn sẽ xây dựng một ứng dụng có một TextView và một Button. Khi người dùng nhấp vào Button, ứng dụng sẽ ngủ trong một khoảng thời gian ngẫu nhiên, sau đó hiển thị thông báo trong TextView khi

ứng dụng thức dậy. Sau đây là giao diện của ứng dụng đã hoàn thành:



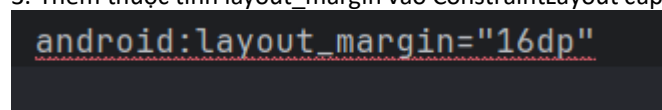
The screenshot shows a mobile application interface for a BMI calculator. At the top, there is a red square icon with a stylized yellow face. Below the icon, the title "BMI Calculate" is displayed in a large, bold, black font. Underneath the title, there are two radio buttons labeled "Male" and "Female". Below these, there is a text input field labeled "Age". Further down, there are two text input fields labeled "Feet" and "inches". Below these, there is a text input field labeled "weight". A large, rounded purple button labeled "Calculate" is positioned below the input fields. At the bottom of the interface, there are two labels: "BMI" and "Message". The entire interface is set against a light purple background and is framed by a dark grey border.

## Nhiệm vụ 1: Thiết lập dự án SimpleAsyncTask

Giao diện người dùng SimpleAsyncTask chứa một Nút khởi chạy AsyncTask và một TextView hiển thị trạng thái của ứng dụng.

### 1.1 Tạo dự án và bố cục

1. Tạo một dự án mới có tên là SimpleAsyncTask bằng mẫu Empty Activity. Chấp nhận các giá trị mặc định cho tất cả các tùy chọn khác.
2. Mở tệp bố cục activity\_main.xml. Nhấp vào tab Văn bản.
3. Thêm thuộc tính layout\_margin vào ConstraintLayout cấp cao nhất:



4. Thêm hoặc sửa đổi các thuộc tính sau của TextView "Hello World!" để có các giá trị này. Trích xuất chuỗi thành một tài nguyên.

Thuộc tính	Giá trị
Android: mã	"@+id/textView1"
Android: văn bản	"I am ready to start work!"
Android: độ lớn văn bản	"24sp"

5. Xóa thuộc tính app:layout\_constraintRight\_toRightOf và app:layout\_constraintTop\_toTopOf attributes.

6. Thêm một phần tử Button ngay dưới TextView, và gán cho nó những thuộc tính này. Trích xuất văn bản của nút vào một tài nguyên chuỗi.

Thuộc tính	Giá trị
Android: mã	"@+id/button"
Android: bố cục chiều rộng	"wrap_content"
Android: bố cục chiều cao	"wrap_content"
Android: văn bản	"start task"
Android: bố cục lề trên	"24dp"
Android: khi nhấp chuột	"startTask"
App: layout_constraintStart_toStartOf	"parent"
App: layout_constraintTop_toBottomOf	"@+id/textView1"

7. Thuộc tính onClick cho nút sẽ được làm nổi bật bằng màu vàng, vì phương thức startTask() chưa được triển khai trong MainActivity. Đặt con trỏ của bạn vào văn bản được làm nổi bật, nhấn Alt + Enter (Option + Enter trên Mac) và chọn Tạo 'startTask(View) trong MainActivity' để tạo stub phương thức trong MainActivity.

8. Mã giải pháp cho activity\_main.xml:

```

<?xml version="1.0" encoding="utf8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res/SHYauto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_margin="16dp"
tools:context=".MainActivity">
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/ready_to_start"
    android:textSize="24sp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="24dp"
    android:onClick="startTask"
    android:text="@string/start_task"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView1"/>
</android.support.constraint.ConstraintLayout>

```

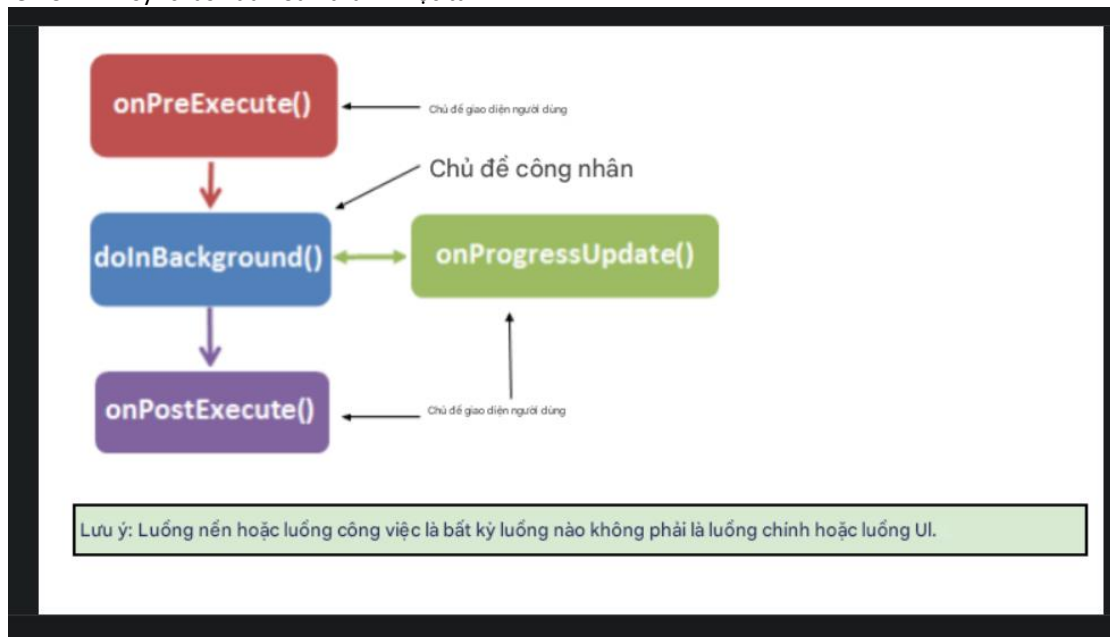
## Nhiệm vụ 2: Tạo lớp con AsyncTask

[AsyncTask](#) là một lớp trừu tượng, có nghĩa là bạn phải kế thừa nó để sử dụng. Trong ví dụ này, AsyncTask thực hiện một tác vụ nền rất đơn giản: nó ngủ trong một khoảng thời gian ngẫu nhiên. Trong một ứng dụng thực tế, tác vụ nền có thể thực hiện đủ loại công việc, từ truy vấn cơ sở dữ liệu, kết nối internet, đến tính toán nước đi tiếp theo trong trò chơi Go để đánh bại nhà vô địch Go hiện tại.

Một lớp con của AsyncTask có các phương thức sau để thực hiện công việc ngoài luồng chính:

- [onPreExecute\(\)](#): Phương thức này chạy trên luồng UI và được sử dụng để thiết lập tác vụ của bạn (như hiển thị thanh tiến trình).
- [doInBackground\(\)](#): Đây là nơi bạn triển khai mã để thực hiện công việc sẽ được thực hiện trên luồng riêng biệt.
- [onProgressUpdate\(\)](#): Phương thức này được gọi trên luồng UI và được sử dụng để cập nhật tiến trình trong UI (chẳng hạn như làm đầy thanh tiến trình).

- **onPostExecute()**: Một lần nữa trên luồng UI, phương thức này được sử dụng để cập nhật kết quả lên UI khi AsyncTask đã hoàn thành việc tải.



Khi bạn tạo một lớp con AsyncTask, bạn có thể cần cung cấp cho lớp con đó thông tin về công việc mà nó sẽ thực hiện, liệu có báo cáo tiến trình của nó hay không và báo cáo tiến trình đó như thế nào, và trả về kết quả theo hình thức nào.

Khi bạn tạo một lớp con AsyncTask, bạn có thể định cấu hình lớp con đó bằng các tham số sau:

- **Params:** Kiểu dữ liệu của các tham số được gửi đến tác vụ khi thực thi phương thức ghi đè `doInBackground()`.
- **Progress:** Kiểu dữ liệu của các đơn vị tiến trình được công bố bằng phương thức ghi đè `onProgressUpdate()`.
- **Result:** Kiểu dữ liệu của kết quả được cung cấp bởi phương thức ghi đè `onPostExecute()`.

Ví dụ: một lớp con AsyncTask có tên là `MyAsyncTask` với khai báo lớp sau có thể lấy các tham số sau:

- Một String làm tham số trong `doInBackground()`, để sử dụng trong truy vấn, chẳng hạn.
- Một số nguyên cho `onProgressUpdate()`, để biểu thị phần trăm công việc hoàn thành
- Một bitmap cho kết quả trong `onPostExecute()`, biểu thị kết quả truy vấn.

```
public class MyAsyncTask
    extends AsyncTask <String, Integer, Bitmap>{}
```

Trong tác vụ này, bạn sẽ sử dụng lớp con AsyncTask để xác định công việc sẽ chạy trong một luồng khác với luồng UI.

## 2.1 Phân lớp AsyncTask

Trong ứng dụng này, lớp con AsyncTask mà bạn tạo không yêu cầu tham số truy vấn hoặc công bố tiến trình của nó. Bạn sẽ chỉ sử dụng các phương thức `doInBackground()` và `onPostExecute()`.

1. Tạo một lớp Java mới có tên là `SimpleAsyncTask` mở rộng AsyncTask và sử dụng ba tham số kiểu chung.

Sử dụng Void cho các tham số, vì AsyncTask này không yêu cầu bất kỳ đầu vào nào. Sử dụng Void cho kiểu tiến trình, vì tiến trình không được công bố. Sử dụng String làm kiểu kết quả, vì bạn sẽ cập nhật TextView bằng một chuỗi khi AsyncTask đã hoàn tất thực thi.

```
public class SimpleAsyncTask extends AsyncTask <Void, Void, String>{}
```

Lưu ý: Phần khai báo lớp sẽ được gạch chân màu đỏ vì phương thức `doInBackground()` bắt buộc vẫn chưa được triển khai.

2. Ở đầu lớp, định nghĩa một biến thành viên mTextView có kiểu là WeakReference<TextView>:

```
private WeakReference<TextView> mTextView;
```

3. Triển khai hàm tạo cho AsyncTask get TextView làm tham số và tạo tham chiếu yếu mới cho TextView đó:

```
SimpleAsyncTask(TextView tv) {  
    mTextView = new WeakReference<>(tv);  
}
```

AsyncTask cần cập nhật TextView trong Activity sau khi hoàn tất trạng thái ngủ (trong phương thức onPostExecute()). Do đó, hàm tạo cho lớp sẽ cần tham chiếu đến TextView để được cập nhật.

Tham chiếu yếu (lớp WeakReference) dùng để làm gì? Nếu bạn truyền TextView vào hàm tạo AsyncTask rồi lưu trữ trong biến thành viên, tham chiếu đó đến TextView có nghĩa là Activity không bao giờ có thể được thu gom rác và do đó làm rò rỉ bộ nhớ, ngay cả khi Activity bị hủy và được tạo lại như trong thay đổi cấu hình thiết bị. Điều này được gọi là tạo ngử cảnh bị rò rỉ, và Android Studio sẽ cảnh báo bạn nếu bạn thử làm vậy.

Tham chiếu yếu ngăn chặn rò rỉ bộ nhớ bằng cách cho phép đối tượng được tham chiếu đó giữ được thu gom rác nếu cần.

## 2.2 Triển khai doInBackground()

Phương thức doInBackground() là bắt buộc đối với lớp con AsyncTask của bạn.

1. Đặt con trỏ vào khai báo lớp được tô sáng, nhấn Alt + Enter (Option + Enter trên máy Mac) và chọn Triển khai phương thức. Chọn doInBackground() và nhấp vào OK. Mẫu phương thức sau được thêm vào lớp của bạn:

```
@Override  
protected String doInBackground(Void... voids) {  
    return null;  
}
```

2. Thêm mã để tạo một số nguyên ngẫu nhiên trong khoảng từ 0 đến 10. Đây là số mili giây mà tác vụ sẽ tạm dừng. Đây không phải là thời gian dài để tạm dừng, vì vậy hãy nhân số đó với

200 để kéo dài thời gian đó.

```
Random r = new Random();  
int n = r.nextInt(11);  
int s = n * 200;
```

3. Thêm khối try/catch và đưa luồng vào trạng thái ngủ

```
try {
    Thread.sleep(s);
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

4. Thay thế câu lệnh return hiện tại để trả về Chuỗi "Awake at last after sleeping for xx mili giây", trong đó xx là số mili giây mà ứng dụng đã ngủ

```
return "Awake at last after sleeping for " + s + " milliseconds!";
```

Phương thức doInBackground() hoàn chỉnh trông như thế này:

```
@Override
protected String doInBackground(Void... voids) {
    // Generate a random number between 0 and 10
    Random r = new Random();
    int n = r.nextInt(11);
    // Make the task take long enough that we have
    // time to rotate the phone while it is running
    int s = n * 200;
    // Sleep for the random amount of time
    try {
        Thread.sleep(s);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    // Return a String result
    return "Awake at last after sleeping for " + s + " milliseconds!";
}
```

## 2.3 Triển khai onPostExecute()

Khi phương thức doInBackground() hoàn tất, giá trị trả về sẽ tự động được chuyển đến hàm gọi lại onPostExecute().

1. Triển khai onPostExecute() để lấy đối số String và hiển thị chuỗi đó trong TextView:

```
protected void onPostExecute(String result) {
    mTextView.get().setText(result);
}
```

Tham số String cho phương thức này là những gì bạn đã định nghĩa trong tham số thứ ba của định nghĩa lớp AsyncTask

của bạn và là những gì phương thức doInBackground() của bạn trả về.

Vì mTextView là tham chiếu yếu, bạn phải tham chiếu nó với phương thức get() để lấy đối tượng TextView

bên dưới và gọi setText() trên đó.

Lưu ý: Bạn có thể cập nhật UI trong `onPostExecute()` vì phương thức đó được chạy trên luồng chính. Bạn không thể cập nhật `TextView` bằng chuỗi mới trong phương thức `doInBackground()` vì phương thức đó được thực thi trên một luồng riêng biệt.

## Nhiệm vụ 3: Triển khai các bước cuối cùng

### 3.1 Triển khai phương thức khởi động `AsyncTask`

Ứng dụng của bạn hiện có lớp `AsyncTask` thực hiện công việc ở chế độ nền (hoặc sẽ thực hiện nếu gọi `sleep()` là công việc được mô phỏng). Bây giờ bạn có thể triển khai phương thức `onClick` cho nút "Start Task"

để kích hoạt tác vụ nền.

1. Trong tệp `MainActivity.java`, hãy thêm một biến thành viên để lưu trữ `TextView`

```
private TextView mTextView;
```

2. Trong phương thức `onCreate()`, khởi tạo `mTextView` thành `TextView` trong bố cục

```
mTextView = findViewById(R.id.textView1);
```

3. Trong phương thức `startTask()`, hãy cập nhật `TextView` để hiển thị văn bản "Napping...". Trích xuất thông điệp đó thành một tài nguyên chuỗi.

```
mTextView.setText(R.string.napping);
```

4. Tạo một thể hiện của `SimpleAsyncTask`, truyền `TextView mTextView` vào constructor. Gọi `execute()` trên thể hiện `SimpleAsyncTask` đó.

```
new SimpleAsyncTask(mTextView).execute();
```

Lưu ý: Phương thức `execute()` là nơi bạn truyền các tham số được phân tách bằng dấu phẩy sau đó được hệ thống truyền vào `doInBackground()`. Vì `AsyncTask` này không có tham số nên bạn để trống.