1. **Giới thiệu**

Mỗi ứng dụng Android khi sử dụng đều được hệ điều hành cấp cho một process, và trên đó có một thread mặc định. Đó là main UI thread. Android xử lý tất cả các sự kiện/tác vụ trên một thread duy nhất gọi là main UI thread. Main UI thread không xử lý các hoạt động đồng thời vì nó chỉ xử lý một sự kiện/task tại một thời điểm.

Do vậy, nếu thực hiện một tác vụ gì đó mà tốn nhiều thời gian trên main UI thread sẽ gây ra hiện tượng treo ứng dụng hay còn gọi là ANR(Application Not Responding).

Để xử lý các tác vụ cần nhiều thời gian như: Tải file từ internet, nén hoặc giải nén… thì chúng ta phải tách tác vụ đó khỏi main UI thread(gọi là xử lý đa nhiệm). Android cung cấp một số công cụ để có thể làm được điều đó như:

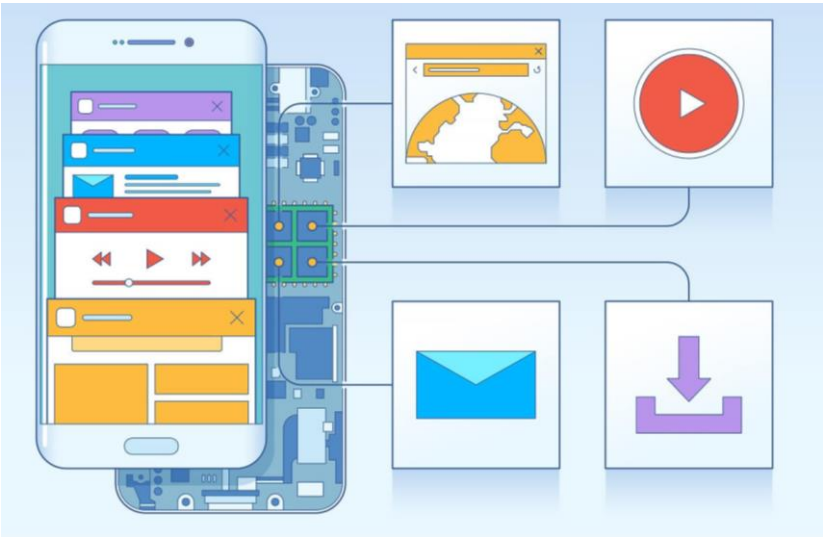
- Sử dụng Service – IntentService
- Sử dụng Thread – một khái niệm của Java
- Loader trong Android
- Hoặc sử dụng AsyncTask trong Android…

2. **Xử lý đa nhiệm trong Android**

Nếu các sự kiện hoặc một task nào đó không được xử lý đồng thời. Thì toàn bộ mã của ứng dụng Android sẽ chạy trên luồng chính và code sẽ được thực hiện tuần tự từng dòng một.

Khi thực hiện một công việc/ tác vụ cần thời gian xử lý như tải nhạc từ Internet, ứng dụng sẽ hiển thị trạng thái treo cho đến khi tải xong.

Để mang lại trải nghiệm người dùng tốt, tất cả tác vụ có khả năng chạy chậm đều phải chạy không đồng bộ.



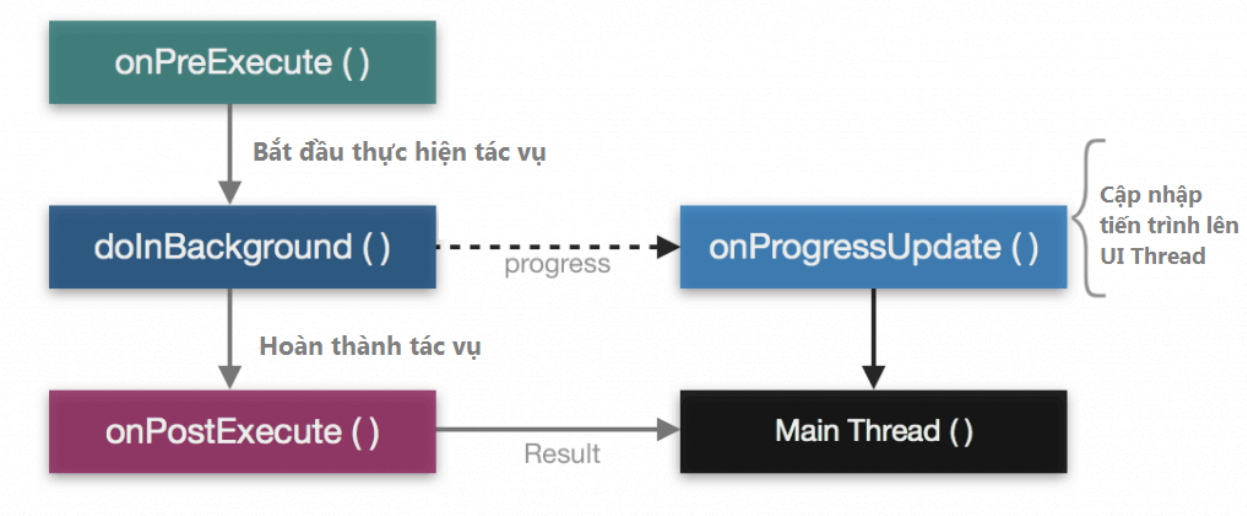
Một số tác vụ cần thời gian xử lý như:

- Truy cập tài nguyên (như MP3, JSON, Hình ảnh) từ Internet.
- Thao tác với cơ sở dữ liệu.
- Tương tác với webService như RESTful, SOAP…
- Các Logic phức tạp mất khá nhiều thời gian như: Nén/giải nén file, sao chép/di chuyển file trong bộ nhớ…

3. AsyncTask

AsyncTask là một abstract Android class, giúp ứng dụng Android xử lý main UI thread hiệu quả hơn. AsyncTask trong Android cho phép chúng ta thực hiện những tác vụ dài mà không ảnh hưởng đến main thread.

```
private class MyTask extends AsyncTask<URL, Integer, Long> {
 protected Long doInBackground(URL... urls) {
      //Yet to code
     }
 protected void onProgressUpdate(Integer... progress) {
     //Yet to code
     }
 protected void onPostExecute(Long result) {
     //Yet to code
     }
}
```



Bước 1: onPreExecute()

Được thực hiện trước khi bắt đầu thực hiện tác vụ. Hàm được gọi trước phương thức doInBackground() và được gọi trên UI thread.

Thông thường, hàm này được dùng để hiển thị thanh progressbar thông báo cho người dùng biết tác vụ bắt đầu thực hiện

Bước 2: doInBackground()

Tất cả code mà cần thời gian thực hiện sẽ được đặt trong hàm này. Vì hàm này được thực hiện ở một thread hoàn toàn riêng biệt với UI thread nên không được phép cập nhật giao diện ở đây.

Để có thể cập nhập giao diện khi tác vụ đang thực hiện. Ví dụ như cập nhập trạng thái % file đã download được, chúng ta sẽ phải sử dụng đến hàm bên dưới onProgressUpdate()

Bước 3: onProgressUpdate()

Hàm này được gọi khi trong hàm doInBackground() gọi đến hàm publishProgress()

Bước 4: onPostExecute()

Hàm này được gọi khi doInBackground hàm thành công việc. Kết quả của doInBackground() sẽ được trả cho hàm này để hiển thị lên giao diện người dùng.

Trong quá trình Asynctask thực hiện tác vụ, hoàn toàn có thể tạm dừng bất kể lúc nào mà không cần phải đợi AsyncTask làm xong sử dụng hàm cancel(boolean).

4. Ứng dụng mô phỏng

Ứng dụng sẽ tạp 1 AsyncTask để cập nhật liên tục TextView các giá trị từ 0% đến 100% và dừng lại khi đạt 100%. Giao diện ứng dụng gồm 1 TextView và 1 nút nhấn để chạy AsyncTask

Bước 1: Tạo lợp MyAsyncTask kế thừa từ AsyncTask với các phương thức override đã đề cập ở trên.

```
package com.example.exercise01;
import android.os.AsyncTask;
public class MyAsyncTask extends AsyncTask<Void, Void, String> {
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
    }
    @Override
    protected void onPostExecute(String s) {
        super.onPostExecute(s);
    }
    @Override
    protected void onProgressUpdate(Void... values) {
        super.onProgressUpdate(values);
    }
    @Override
    protected String doInBackground(Void... voids) {
        return null;
    }
}
```

Bước 2: Tạo các thuộc tính giá trị phần trăm và thuộc tính TextView truy xuất thay đổi TextView ở giao diện chính. Phương thức khởi tạo MyAsyncTask truyền vào TextView

```
private WeakReference<TextView> mTextView;
    private static int percent;
    MyAsyncTask(TextView tv) {
        percent=0;
        mTextView = new WeakReference<>(tv);
    }
```

Bước 3: Xử lý hiển thị TextView ngoài giao diện với MyAsyncTask

Phương thức xử lý tác vụ chạy ngầm:

```
protected String doInBackground(Void... voids) {
        try {
                percent++;
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return Integer.toString(percent);
    }
```

Phương thức cập nhật TextView sau khi tác vụ chạy xong

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

    }
```

Gọi thực thi AsyncTask

```
public class MainActivity extends AppCompatActivity {
    private TextView mTextView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mTextView = findViewById(R.id.textView);
    }
    public void startTask(View view) {
        new MyAsyncTask(mTextView).execute();
    }
}
```