LẬP TRÌNH ANDROID CĂN BẢN

Bài 7: Background

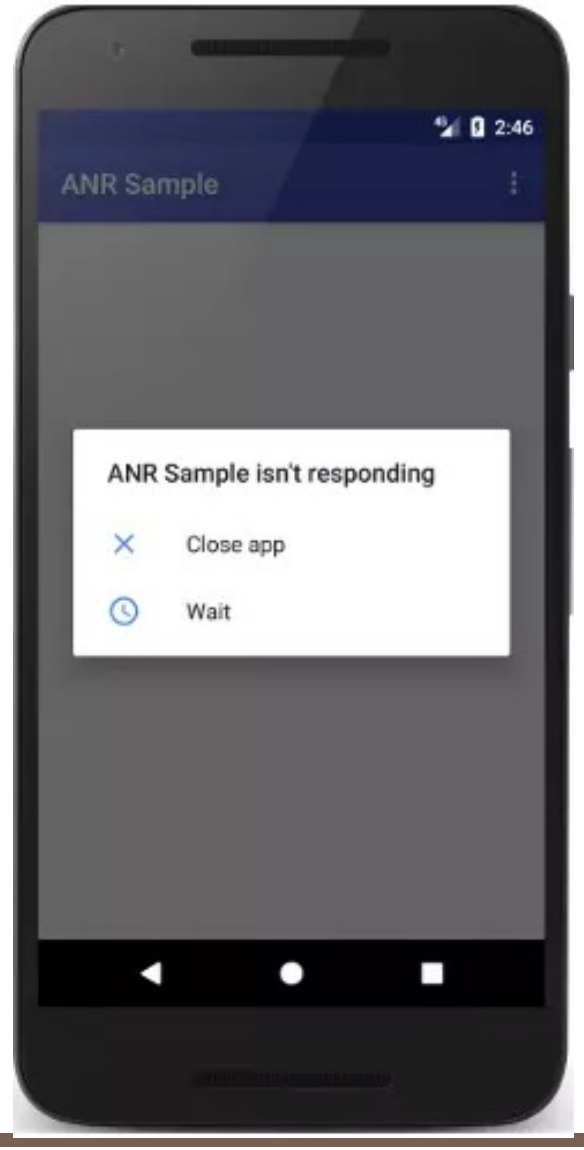
Ths. Trần Xuân Thanh Phúc | Trường Đại học Công Nghiệp Thực Phẩm

Nội dung

- ANR
- Thread, Handler, AsyncTask
- Service
- Broadcast Receiver

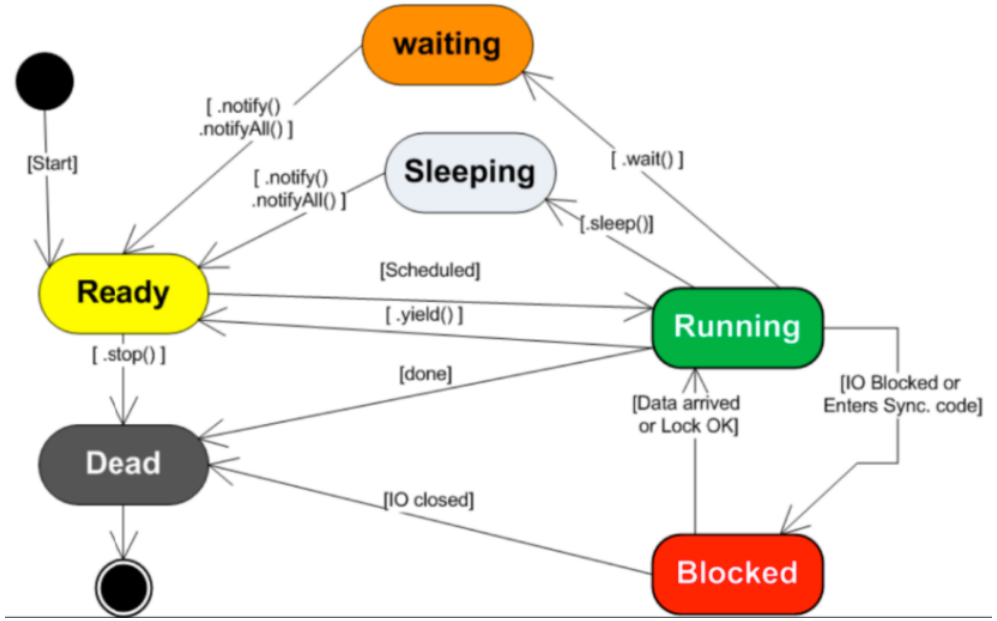
1. ANR (Application Not Responding)

- ANR là lỗi thường gặp trong lập trình luồng. Theo mặc định, hệ thống sẽ hủy chương trình khi UI thread bị chặn trong hơn 5 giây.
- Để không xảy ra hiện tượng trên thì Android đã đề ra 2 rules sau và buộc lập trình viên phải tuân theo:
  - Không được block UI thread → sử dụng đa luồng
  - Không được kết nối với Android UI từ một thread không phải là UI thread → Handler, AsyncTask



2. Đa luồng

- Thread là một luồng của một chương trình đang chạy
- Một chương trình (process) đang chạy thường sẽ có 1 luồng chính (main thread / UI thread) và có thể tạo ra được nhiều các luồng khác chạy xung quanh để phục vụ cho luồng chính đó. Các luồng này có thể chạy song song với nhau.
- Các luồng có thể dùng chung tài nguyên của tiến trình để thực hiện các thao tác cần thiết.



2.1 Thread

- Có 2 cách để tạo và thực thi 1 Java Thread:
  - Tạo một lớp thread mới kế thừa từ lớp Thread
  - Tạo một biến mới của lớp Thread với đối tượng Runnable

```
public class NewThread extends Thread{
    public void run()
    {
        print("Hello")
    }
}
```

```
Thread background = new Thread(new Runnable() {
    public void run() {
        print("Hello")
    }
}
```

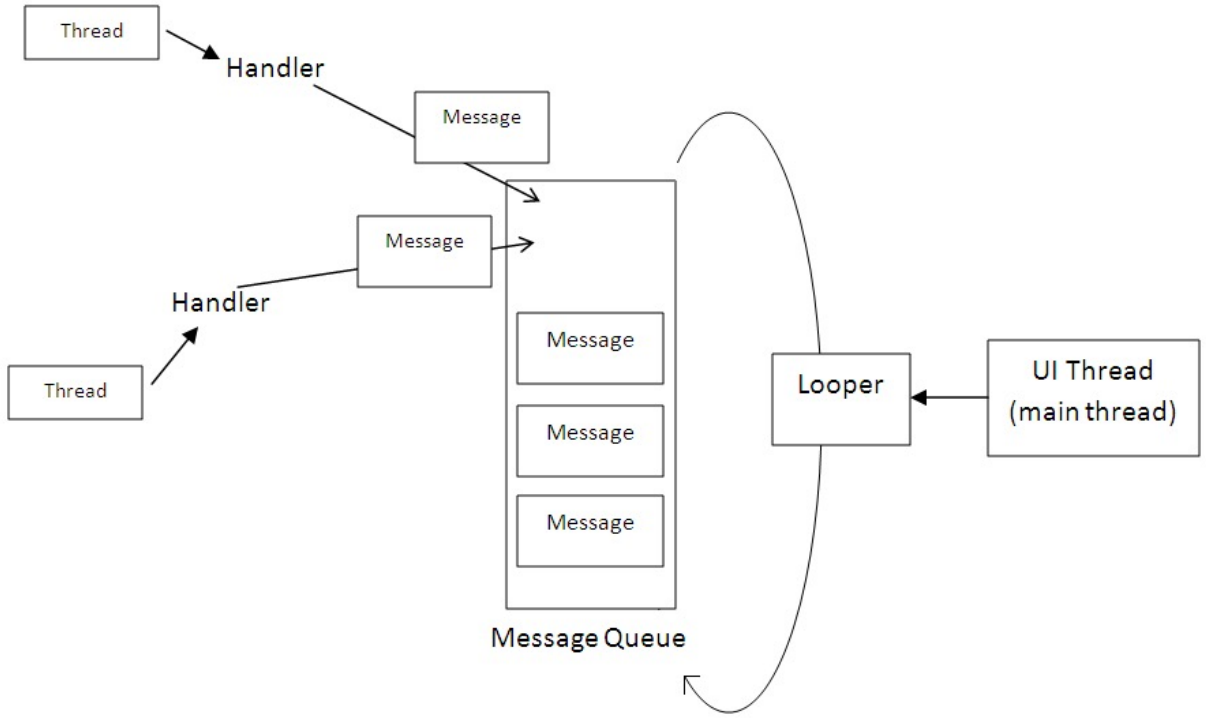
2.2 Handler

- Trong android, các thread tương tác với nhau thông qua các đối tượng được gọi là Handler. Khi một tiến trình được tạo cho một ứng dụng, luồng chính của nó được dành riêng để chạy một message queue, queue này quản lý các đối tượng bậc cao của ứng dụng (activity, intent receiver, ..).
- Khi ta tạo một Handler mới, nó được gắn với message queue của thread tạo ra nó – từ đó trở đi, nó sẽ gửi các message và các runnable tới message queue đó và thực thi chúng khi chúng ra khỏi message queue.

2.2 Handler

- Nhiệm vụ chính của Handler là: xếp lịch cho các message và runnable cần được thực thi vào thời điểm nào đó trong tương tai. Mỗi luồng phụ cần liên lạc với luồng chính thì phải yêu cầu một message token bằng cách dùng phương thức obtainMessage().
- Khi đã lấy được token, thread phụ đó dùng phương thức sendMessage() để gắn nó vào message queue của Handler.
- Handler liên tục xử lý các message mới được gửi tới thread chính bằng phương thức handleMessage(), hoặc yêu cầu chạy các đối tượng runnable qua phương thức post().

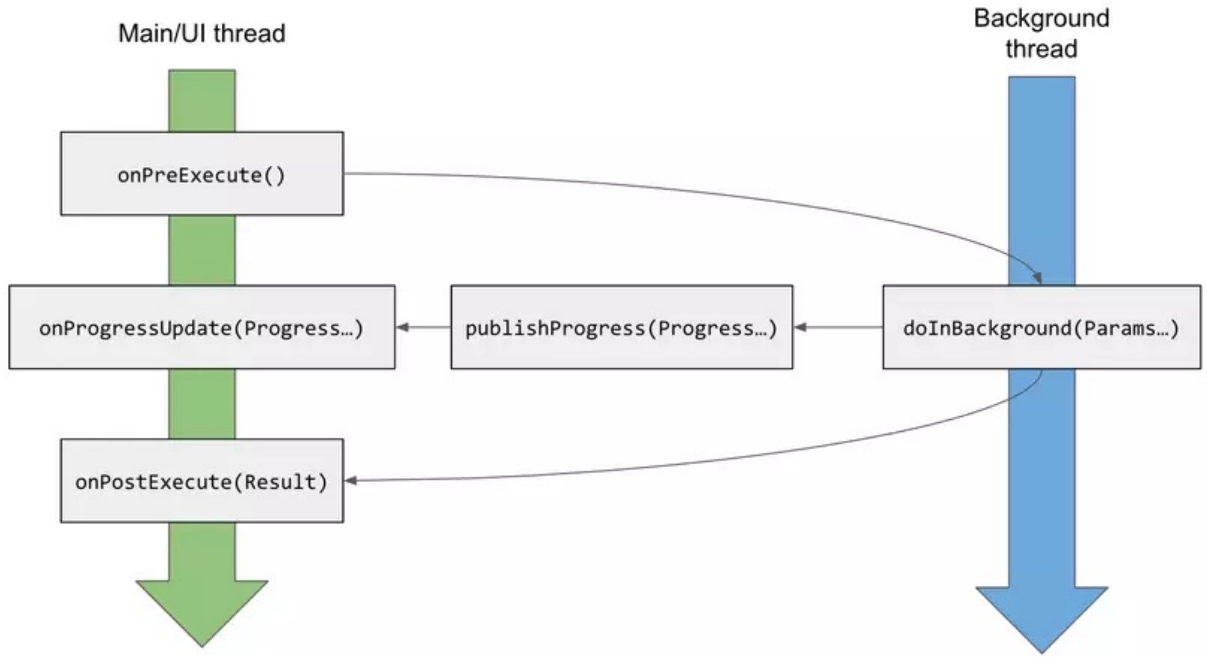
2.2 Handler




```
TextView tv = findViewById(R.id.labelDemo);
Button btn = findViewById(R.id.btnClick);
Handler handler = new Handler(new Handler.Callback() {
    @Override
    public boolean handleMessage(@NonNull Message message) {
        int i = message.getData().getInt( key: "step");
        tv.setText(String.valueOf(i));
        return true;
    }
});
btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Thread t = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 0; i < 10; i++) {
                    try {
                        Message m = handler.obtainMessage();
                        Intent intent = new Intent();
                        intent.putExtra( name: "step", i);
                        m.setData(intent.getExtras());
                        handler.sendMessage(m);
                        Thread.sleep( millis: 1000);
                    } catch (InterruptedException e) {
                        throw new RuntimeException(e);
                    }
                    Log.e( tag: "LOG COUNT", String.valueOf(i));
                }
            }
        });
        t.start();
    }
});
```

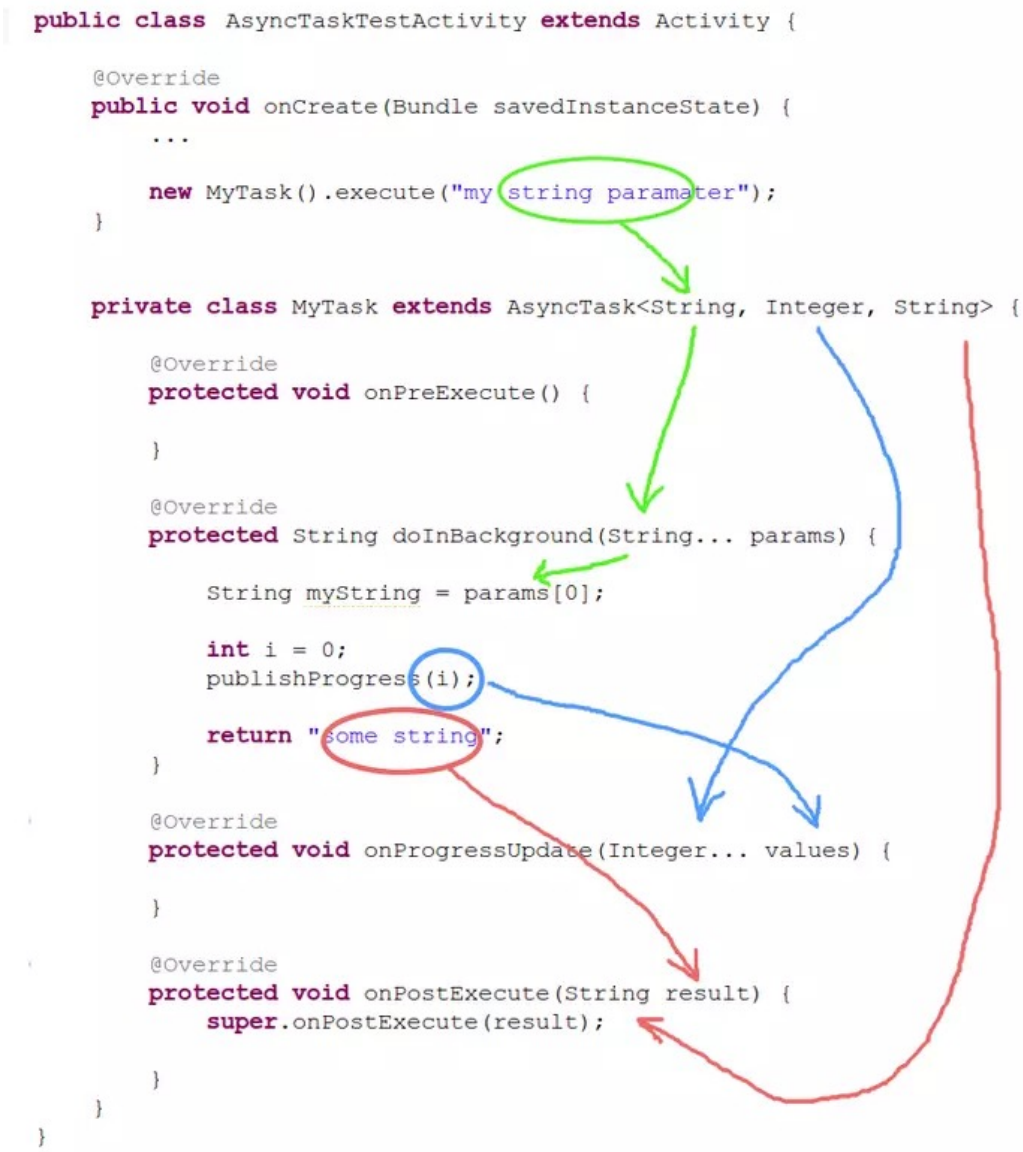
2.3 AsyncTask

- AsyncTask là một đối tượng thích hợp dùng để giao tiếp giữa một Thread với UI Thread
- Nó cho phép thực hiện công việc ở background và sau đó cập nhật giao diện ở UI Thread.
- AsyncTask = Thread + Handler.



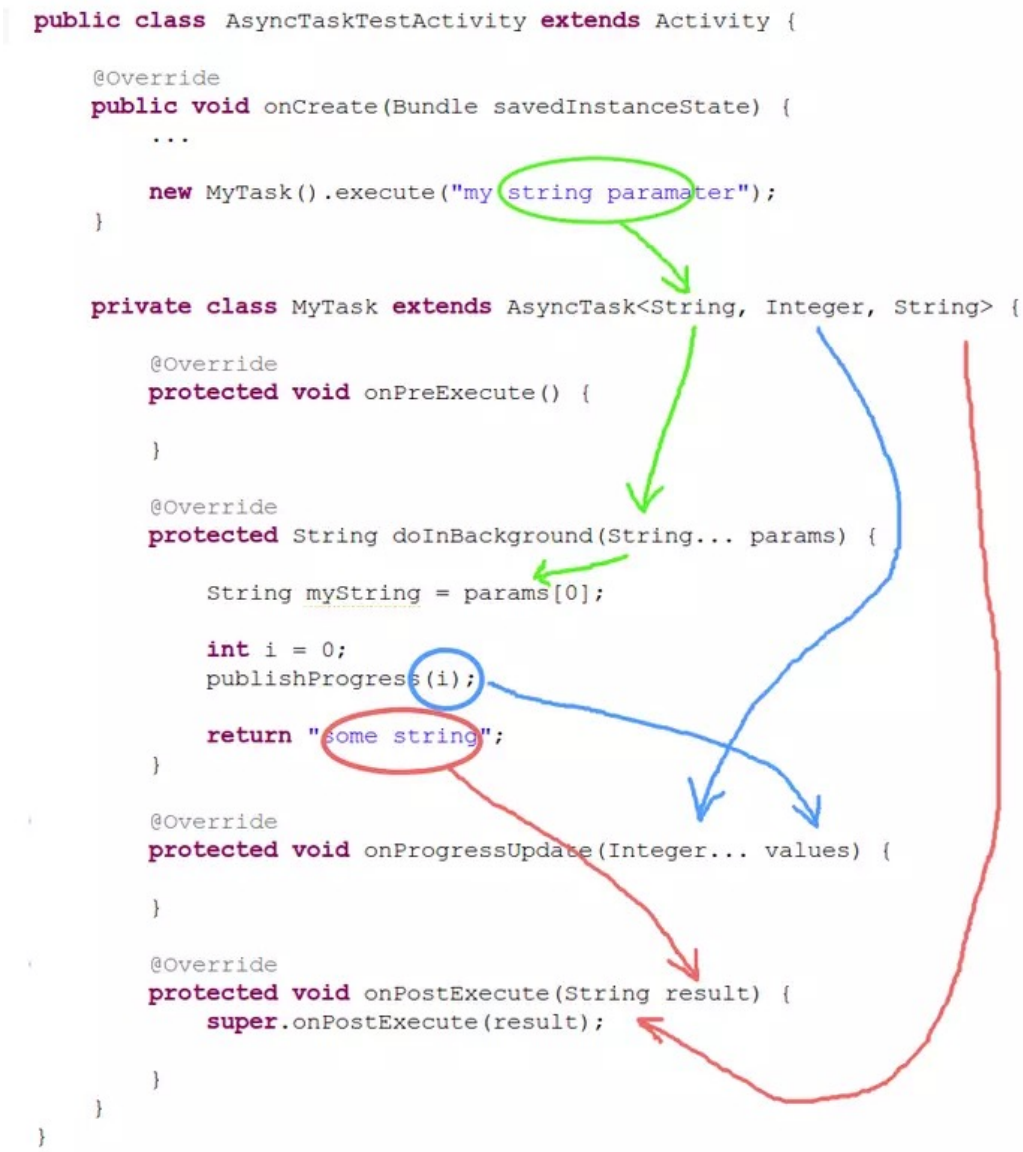
2.3 AsyncTask

- onPreExecute(): được gọi từ UI thread ngay sau khi thực thi task. Bước này thường dùng để setup task, vd: hiển thị một progress bar lên giao diện
- doInBackground(Params...): được gọi ở background thread ngay sau onPreExecute() hoàn tất. Dùng để thực hiện các xử lý tốn nhiều thời gian. Bước này còn gọi đến publishProgress(Progress...) để gửi thông tin về progress, các giá trị này được gửi đến UI thread trong phương thức onProgressUpdate(Progress...)



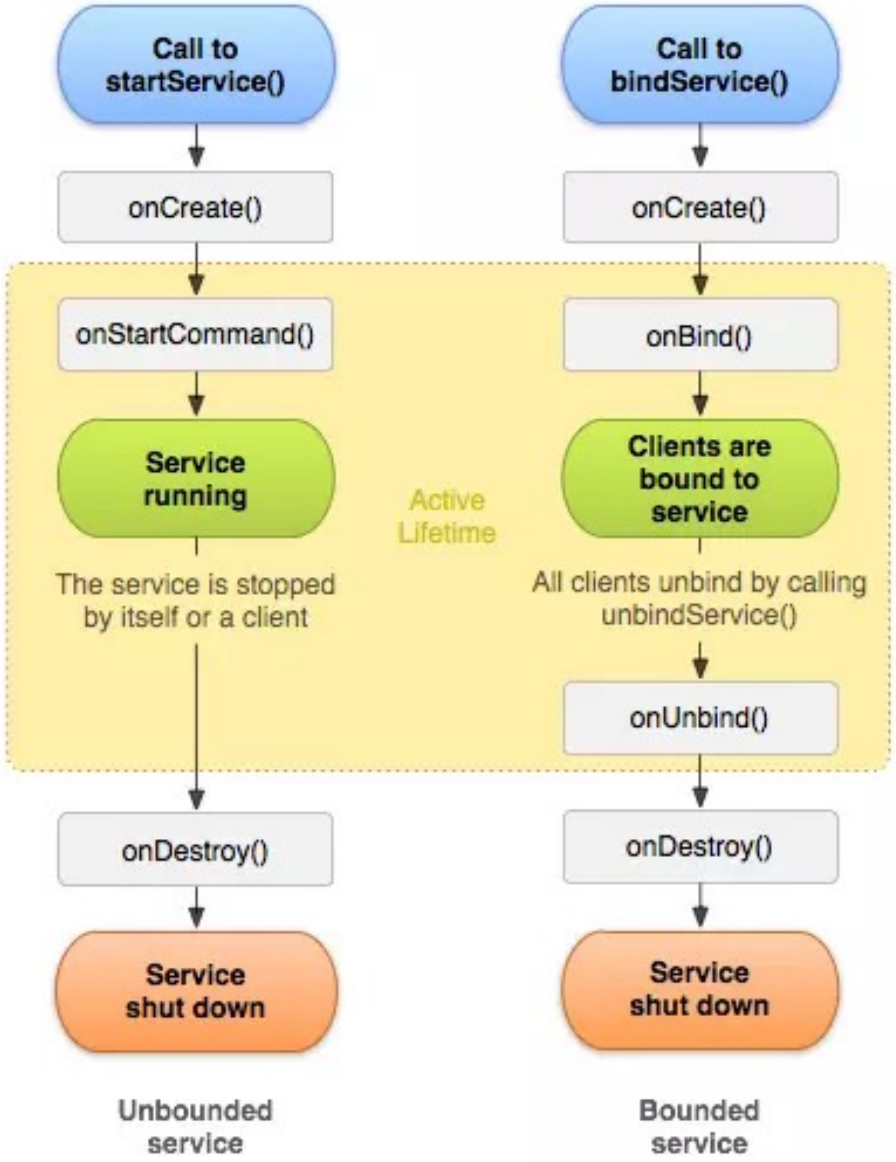
2.3 AsyncTask

- onProgressUpdate(Progress...): được gọi ở UI thread sau khi gọi publishProgress(Progress...). Phương thức này dùng để thông tin đến người dùng về progress trong lúc background đang xử lý, tính toán.
- onPostExecute(Result): được gọi ở UI thread sau khi quá trình xử lý ở background kết thúc. Kết quả của background được truyền tới bước này chính là tham số của phương thức. Ở bước này, UI thread thực hiện cập nhật giao diện với kết quả trả về.



3. Service

- Service là một trong bốn thành phần cơ bản của một ứng dụng Android, được sử dụng để thực hiện các tiến trình ngầm mà không ảnh hưởng đến hoạt động của người dùng, không cần tương tác với người dùng thậm chí ngay cả khi ứng dụng đã tắt.
- Service giống như Activity thực hiện theo chu kỳ thời gian để quản lý sự thay đổi của trạng thái.

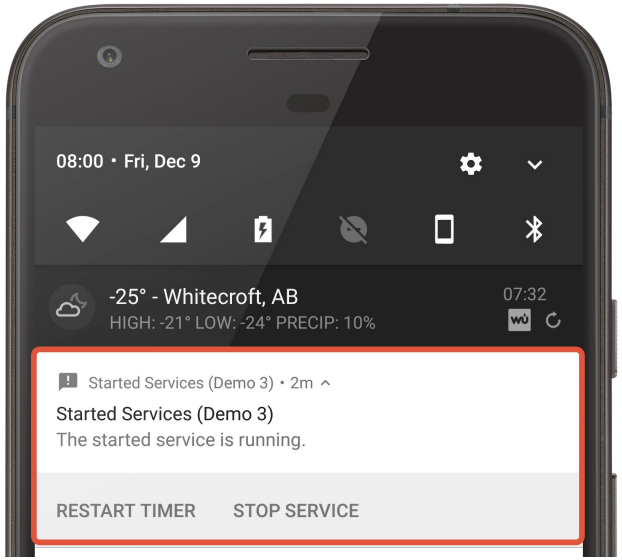


3. Service

- Có 3 loại
  - **Started**: một service khi nó được gọi bằng phương thức startService(). Khi đó service có thể chạy ngầm vô thời hạn, ngay cả khi component đã gọi nó ngừng hoạt động. Khi service thực hiện xong các tác vụ nó đảm nhận hoặc được gọi bởi phương thức stopService() thì service sẽ ngừng hoạt động.
  - **Bound**: một service khi nó được liên kết với một thành phần của ứng dụng qua phương thức bindService(), Bound Service sẽ cung cấp một giao diện client-server cho phép các thành phần có thể tương tác với service. Nhiều component có thể bind đến service cùng 1 lúc. Bound Service sẽ ngừng hoạt động khi tất cả các component đó được unbind.

3. Service

– **Foreground**: Một Foreground Service thực hiện một số thao tác mà người dùng chú ý, có thể thấy rõ ràng. (thường đi kèm với Notification)



3. Service – độ ưu tiên

- Hệ thống Android bắt buộc phải dừng một service khi bộ nhớ ít và phải khôi phục tài nguyên hệ thống cho Activity đang được sử dụng. Nếu Service được ràng buộc với một Activity đang sử dụng, hoặc được khai báo và chạy ở chế độ Foreground thì khó bị dừng hơn. Nếu Service là Started và chạy lâu dài, hệ thống sẽ làm giảm vị trí ưu tiên của nó
- Bound > Foreground > Background
- Tùy thuộc vào giá trị trả về trong onStartCommand() mà Service có thể được khởi động lại

3. Service - Các giá trị trả về trong onStartCommand()

- **START_NOT_STICKY**: Nếu hệ thống kill service thì service này không được khởi động lại trừ khi có một Intent đang được chờ ở onStartCommand(). Đây là lựa chọn an toàn nhất để tránh chạy Service khi không cần thiết và khi ứng dụng có thể khởi động lại một cách đơn giản các công việc chưa hoàn thành.
- **START_STICKY**: nếu service bị hệ thống kill và onStartCommand không có một intent nào chờ nó nữa thì Service sẽ được hệ thống khởi động lại với một Intent null.

3. Service - Các giá trị trả về trong onStartCommand()

- **START_REDELEVER_INTENT**: Nếu Service bị kill thì nó sẽ được khởi động lại với một Intent là Intent cuối cùng mà Service được nhận → thích hợp với các service đang thực hiện công việc muốn tiếp tục ngay tức thì như download file
- **START_STICKY_COMPATIBILITY**: Giá trị này cũng giống như START_STICKY nhưng nó không chắc chắn, đảm bảo khởi động lại service.

3. Service

```
// Đăng ký service với Manifest
<service android:name=". DemoService"/>
```

```
// Start service
Intent intent = new Intent(
            this,  // Context
            DemoService.class
);
startService(intent);
```

```
public class DemoService extends IntentService {

    public DemoService()
    {
        super("Demo Service");  // Tên service, có ý nghĩa trong việc debug
    }

    @Override
    public int onStartCommand(@Nullable Intent intent, int flags, int startId) {
        onStart(intent, startId);
        return START_STICKY;
    }

    @Override
    protected void onHandleIntent(@Nullable Intent intent) {
        // Long code
    }
}
```

4. Broadcast Receiver

- Broadcast Receiver là một trong 4 component lớn trong Android, với mục đích là lắng nghe các sự kiện, trạng thái của hệ thống phát ra thông qua Intent nhờ đó mà có thể xử lí được các sự kiện hệ thống ở bên trong ứng dụng.
- Broadcast Receiver có thể hoạt động được cả khi ứng dụng bị tắt đi, vì vậy nó thường được sử dụng với service.
- Có 2 cách để đăng ký đối tượng của Broadcast receiver:
  - Context.registerReceiver()
  - ~~Manifest với tag receiver~~

4. Broadcast Receiver

- Có 2 thành phần chính của Broadcast có thể nhận Intent
  - **Normal broadcasts**: thực hiện gửi theo cơ chế không đồng bộ, các Receiver thực thi không theo thứ tự nhất định. Sử dụng phương thức sendBroadcast().
  - **Ordered broadcasts**: Vào mỗi thời điểm chỉ gửi đến một broadcast và sử dụng phương thức sendOrderedBroadcast().
- Chỉ Broadcast receiver được cung cấp quyền mới có thể nhận những intent gửi đi.

4. Broadcast Receiver

```
public class WifiReceiver extends BroadcastReceiver {
   @Override
   public void onReceive(Context context, Intent intent) {
      Log.e(getClass().getCanonicalName(), "Wifi thay doi");
   }
}
```

```
// Đăng ký BR
IntentFilter intentFilter = new
IntentFilter(WifiManager.WIFI_STATE_CHANGED_ACTION);
registerReceiver(new WifiReceiver(), intentFilter);
```

```
// BR as Service
WifiReceiver wifiReceiver;
@Override
public void onCreate() {
   super.onCreate();
   wifiReceiver = new WifiReceiver();
   IntentFilter intentFilter = new
IntentFilter(WifiManager.WIFI_STATE_CHANGED_ACTION);
   registerReceiver(wifiReceiver, intentFilter);
}
@Override
public void onDestroy() {
   super.onDestroy();
   unregisterReceiver(wifiReceiver); // Destroy BR
}
@Override
public int onStartCommand(@Nullable Intent intent, int flags, int startId) {
   onStart(intent, startId);
   return START_STICKY;
}
@Override
protected void onHandleIntent(@Nullable Intent intent) {
   while(true);
}
```

LẬP TRÌNH ANDROID CĂN BẢN

Kết thúc ☺️

Ths. Trần Xuân Thanh Phúc | Trường Đại học Công Nghiệp Thực Phẩm