ASYNCTASK - THREAD – HANDLER

1. THREAD VÀ MULTITHREADING

1.1. Thread

Trong lập trình ứng dụng cho di động, một trong những vấn đề quan trọng cần giải quyết đó là tài nguyên hạn hẹp của các thiết bị động. Các thiết bị di động hầu hết có tài nguyên rất hạn chế, ngày nay tuy đã được cải thiện rất nhiều (nâng cấp CPU, GPU, RAM…), nhưng nhìn chung lại, không thể chạy ứng dụng trên các thiết bị di động như trên PC, laptop… Do đó, việc xử lí các thuật toán cần phải tinh giản và thực hiện một cách nhanh nhất có thể để có thể tiết kiệm được chi phí tài nguyên.

Thread là đơn vị nhỏ nhất của tiến trình được định thời bởi hệ điều hành và được bao hàm trong các tiến trình thực thi của máy tính. Mỗi một thread có một callstack cho các phương thức, đối số và biến cục bộ của thread đó.

Android chạy trên một hệ điều hành nhân Linux. Các ứng dụng của Android được viết bằng ngôn ngữ Java, và chúng chạy trong một máy ảo, đó không phải máy ảo Java(JVM) mà là máy ảo Dalvik (Dalvik Virtual Machine). Mỗi một máy ảo Android khi chạy đều có ít nhất một thread chính khi khởi động và có thể còn vài thread khác dùng để quản lý các tiến trình đang chạy. Ứng dụng có thể tự khởi động thêm các thread phụ vào các mục đích cụ thể. Các thread trong cùng một máy ảo tác động qua lại và được đồng bộ hóa bằng cách sử dụng các đối tượng chia sẻ và các monitor liên quan đến các đối tượng đó.

- Mỗi Thread có một độ ưu tiên (số nguyên từ 1 - 10) dùng để xác định lượng thời gian CPU (CPU time) mà thread có thể sử dụng dựa vào phương thức setPriority(int).
- Các phương thức của Thread như sleep(), wait() , notify() và notifyAll()…
- Để tạo ra một thread khác ngoài thread chính trên, chúng ta có 2 cách:
    - Cách 1:
        - Tạo 1 lớp mới kế thừa từ lớp java.lang.Thread và ghi đè lại phương thức run() của lớp này:

```
ublic class MyThread extends Thread{
@Override
public void run() {
  //do something }}
```

- Sử dụng Thread:

```
yThread t = new MyThread();
 t.start();
```

  - Cách 2:
    - Tạo 1 lớp và cho lớp này implement Interface Runnable:

```
ublic class MyRunnable implements Runnable{
@Override
public void run() {
   //xử lý
}
```

- Sử dụng thread:

```
yRunnable b = new MyRunnable();
 MyThread t = new MyThread(b);
 t.start();
```

1.2. Multithreading

- Multithreading có nghĩa là đa luồng, nhiều thread. Với cơ chế đa luồng, các ứng dụng của bạn có thể thực thi đồng thời nhiều dòng lệnh cùng lúc. Có nghĩa là bạn có thể làm nhiều công việc đồng thời trong cùng một ứng dụng của bạn. Có thể hiểu một cách đơn giản: hệ điều hành với cơ chế đa nhiệm cho phép nhiều ứng dụng chạy cùng lúc thì ứng dụng với cơ chế đa luồng cho phép thực hiện nhiều công việc cùng lúc.
- Ưu điểm của việc sử dụng Multithreading:
  - Các thread chia sẻ tài nguyên của tiến trình nhưng vẫn có thể được thực thi một cách độc lập. Multi-thread sẽ rất hữu dụng khi dùng lập trình đa nhiệm.
  - Các ứng dụng Multi-thread chạy nhanh hơn trên các máy tính hỗ trợ xử lý đa luồng.
- Nhược điểm:
  - Khó khăn trong việc lập trình, việc sử dụng multithreading không hề dễ dàng, đòi hỏi lập trình viên có kinh nghiệm.
  - Khó khăn trong việc quản lí.
  - Có khả năng tiềm tàng xảy ra tình trang deadlock.

1.3. Handler

- Handler là đối tượng cho phép gửi, xử lí các thông điệp và các đối tương Runable có liên quan đến hàng đợi thông điệp. Mỗi Handler có thể liên kết với một thread và hàng đợi thông điệp của thread đó.
- Bạn có thể tạo ra một thread của riêng bạn và giao tiếp ngược với main thread của ứng dụng thông qua một Handler. Điều này được thực hiện bằng cách gọi sendMessage nhưng từ thread mới của bạn.
- Tạo một đối tượng Handler và ghi đè lại phương thức handleMessage để bắt và xử lý các thông điệp liên lạc.

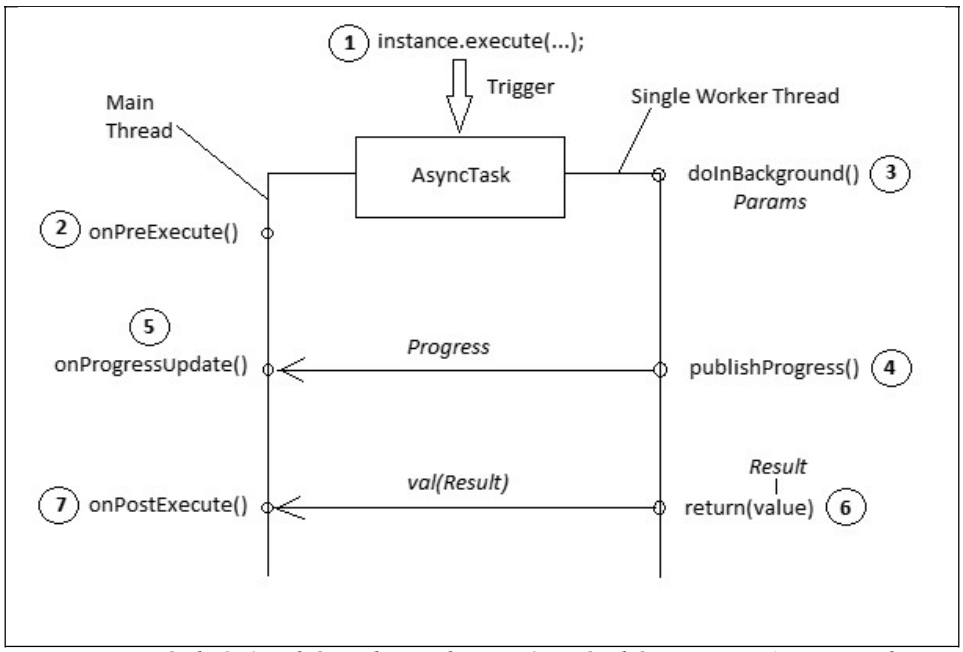
```
andler handler = new Handler(new Handler.Callback() {
      @Override
      public boolean handleMessage(Message msg) {
   //Kiểm tra message
      if(msg != null){
      //Xử lý message nhận được từ Thread

      }
      return true;
   }
```

- Thread sẽ gửi tin nhắn lên Handler và Handler đóng vai trò kiểm tra tin nhắn là gì và làm những việc tương ứng.
- Ta chú ý các phương thức sendMessage sau:
    - sendMessage(): gửi tin nhắn lên Handler ngay lập tức.
    - sendMessageAtFrontOfQueue(): gửi tin nhắn lên Handler và tin nhắn đó sẽ ưu tiên giải quyết trước.
    - sendMessageAtTime(): gửi tin nhắn vào thời điểm chỉ định trước.
    - sendMessageDelayed(): gửi tin nhắn vào Handler sau một khoảng thời gian nhất định.

```
hread t1 = new Thread(new Runnable() {
       @Override
       public void run() {
                Message mss = new Message();
                   mss.arg1 = 1;
                   handler.sendMessage(mss);
                 }
});
t1.start();
```

2. ASYNCTASK

- AsyncTask là đối tượng cho phép bạn thực hiện hành động xử lý phía background (Thread ngầm) và trả kết quả lên UI thread (Thread chính) mà không cần phải xử lý Thread hoặc Handler.
- Khi ứng dụng của bạn thực hiện một tác vụ dài, chiếm một khoảng thời gian nhất định. Ví dụ như khi click nút Login, ứng dụng phải gửi request lên server để xử lý và trả kết quả về. Nếu làm theo kiểu bình thường thì ứng dụng của bạn sẽ có cảm giác bị treo và chậm, gây ảnh hưởng đến trải nghiệm người dùng. Trong tình huống này, để giải quyết vấn đề chúng ta có thể sử dụng AsyncTask.
- Trong AsyncTask<Params, Progress, Result> có 3 đối số là các Generic Type:
    - Params: Là giá trị (biến) được truyền vào khi gọi thực thi tiến trình và nó sẽ được truyền vào doInBackground.
    - Progress: Là giá trị (biến) dùng để update giao diện diện lúc tiến trình thực thi, biến này sẽ được truyền vào hàm onProgressUpdate.
    - Result: Là biến dùng để lưu trữ kết quả trả về sau khi tiến trình thực hiện xong.
        - Những đối số nào không sử dụng trong quá trình thực thi tiến trình thì ta thay bằng kiểu Void.
- Thông thường trong một AsyncTask sẽ chứa 4 hàm, đó là :
    - onPreExecute(): Tự động được gọi đầu tiên khi tiến trình được kích hoạt.
    - doInBackground(): Được thực thi trong quá trình tiến trình chạy nền, thông qua hàm này để ta gọi hàm onProgressUpdate để cập nhật giao diện (gọi lệnh publishProgress). Ta không thể cập nhật giao diện trong hàm doInBackground().
    - onProgressUpdate(): Dùng để cập nhật giao diện lúc runtime.
    - onPostExecute(): Sau khi tiến trình kết thúc thì hàm này sẽ tự động xảy ra. Ta có thể lấy được kết quả trả về sau khi thực hiện tiến trình kết thúc ở đây.



Hình 8.1. Thứ tự hoạt động của các hàm trong AsyncTask.

Ví dụ:

```
public class MainActivity extends Activity {
  long startMillis;
  TextView txtHello;
           @Override
           protected void onCreate(Bundle savedInstanceState) {
               super.onCreate(savedInstanceState);
               setContentView(R.layout.activity_main);
                         txtHello          =          (TextView)
findViewById(R.id.txtHello);
                         MyAsyncTask     myAsyncTask      =     new
MyAsyncTask();
      myAsyncTask.execute();
             }
```

- Tạo lớp MyAsyncTask kế thừa từ lớp AsyncTask:

```
private class MyAsyncTask extends AsyncTask<String, Integer,
Void>{

        @Override
        protected void onPreExecute() {
    super.onPreExecute();
```

```
  startMillis = System.currentTimeMillis();
    Log.d("Test", "onPreExecute");
     }
      @Override
         protected void onProgressUpdate(Integer... values) {
         super.onProgressUpdate(values);
      Log.d("Test", "Công việc đang làm..." + values[0]);
       }
      @Override
         protected void onPostExecute(Void result) {
            super.onPostExecute(result);
            Log.d("Test", "Thời gian kết thúc: "
               + (System.currentTimeMillis()-startMillis)/1000
" giây");
         Log.d("Test", "Công việc hoàn thành.");
      }
     @Override
        protected Void doInBackground(String... arg0) {
          Log.d("Test", "doInBackground");
          for (int i = 0; i < 3; i++) {
               try {
       Thread.sleep(2000);
          publishProgress(i);
      } catch (InterruptedException e) {
          e.printStackTrace();
    }
         }
         return null;
      }
```

- *Lưu ý:*
    - Đối tượng của AsyncTask chỉ được dùng trong UI thread (Thread chính của ứng dụng).
    - Các phương thúc onPostExecute, onPreExecute, onProgressUpdate chỉ là tùy chọn.