

ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
Dipartimento di Informatica – Scienza e Ingegneria (DISI)
C.d.S. in Ingegneria e Scienze Informatiche, Campus di Cesena

Programmazione in Android

Processi, Thread e Task

Angelo Croatti
a.croatti@unibo.it

Sistemi Embedded e Internet of Things
A.A. 2019 – 2020

Outline

- 1 Processi in Android
- 2 Il Main Thread
- 3 Worker Thread
- 4 Handler
 - Android Looper
- 5 Async Task



Processi in Android

- Tutti i componenti di un'applicazione Android sono eseguiti nell'ambito dello stesso Processo (*Linux Process*)
 - ▶ Creato dal sistema all'avvio del primo componente dell'applicazione.
 - ▶ Con opportuni parametri è possibile richiedere che l'applicazione usi più processi.

"as long as possible" policy

- Android assicura che ciascun processo sia mantenuto attivo per il maggior tempo possibile, ma...
 - ▶ ...il sistema può decidere in qualunque momento di terminare qualunque processo attivo, ad eccezione di quello attualmente in foreground.
- A tutti i processi attivi è associato un ranking che consente di stabilire quale/i processo/i interrompere per rilasciare risorse necessarie o per questioni di performance.

Outline

- 1 Processi in Android
- 2 II Main Thread**
- 3 Worker Thread
- 4 Handler
 - Android Looper
- 5 Async Task



Il Main Thread (UI Thread)

- All'avvio di ciascuna applicazione il sistema crea un thread principale chiamato **Main Thread**
 - ▶ Tutti i componenti di un'applicazione sono istanziati nel Main Thread.
 - ▶ È deputato alla gestione di tutti gli eventi generati.
 - ▶ Tutte le chiamate di sistema e l'esecuzione di tutte le callback sono gestite dal Main Thread.
- Il Main Thread ha (necessariamente) alcune restrizioni in merito al codice che può eseguire.
 - ▶ In particolare, non può eseguire compiti “long-running”
 - ▶ Se resta bloccato per più di 5 secondi, il sistema interrompe istantaneamente l'intera applicazione (*ANR Message Dialog*).

Restrizioni per il Main Thread

1. **Il Main Thread è l'unico thread dell'applicazione che può eseguire operazioni sull'interfaccia utente.**
 - ▶ Es.: Creazione di componenti UI, aggiornamento dello stato, lettura/percezione dello stato, ...
 - ▶ **L'UI toolkit di android non è thread-safe...**
2. Non può eseguire compiti che potenzialmente potrebbero richiedere *molto* tempo. Tra cui:
 - ▶ Comunicazioni di rete (es. Richieste HTTP, Socket R/W, ...)
 - ▶ Accesso a DB, esecuzione di Query, ...
 - ▶ ...

Outline

- 1 Processi in Android
- 2 Il Main Thread
- 3 Worker Thread**
- 4 Handler
 - Android Looper
- 5 Async Task



Uso dei Worker Thread

- In Android è possibile sfruttare il supporto di Java al Multi-Threading e creare/eseguire nuovi thread a cui far eseguire compiti long-running.
 - ▶ `java.lang.Thread`, `java.lang.Runnable`,...
- Quindi è possibile creare ed eseguire thread come avviene in Java.

Approfondimenti/Richiami

- Oracle online Tutorial about Concurrency in Java
 - » docs.oracle.com/javase/tutorial/essential/concurrency/index.html
- Bruce Eckel, *Thinking in Java*, 4th Ed. (Concurrency Chapter)

Uso dei Worker Thread – Esempio I

Descrizione: *Alla pressione di un bottone, si vuole scaricare dalla rete un'immagine (specificandone l'URL) e associarla ad un componente grafico dedicato (imageView) presente sulla UI.*

Primo Tentativo

```
public void onClick(View v) {  
  
    Thread worker = new Thread(new Runnable(){  
        public void run() {  
            Bitmap b = downloadImage("http://site.com/img.png");  
            imageView.setImageBitmap(b);  
        }  
    });  
  
    worker.start();  
}  
  
private Bitmap downloadImage(String url){/* Long-Running Task */}
```

Uso dei Worker Thread – Esempio II

- Il codice dell'esempio sembra corretto ma... non funziona!
 - ▶ Si sta violando il principio per il quale solo il Main Thread può eseguire operazioni sull'interfaccia utente.
- Eccezione a run-time quando si tenta di eseguire l'istruzione `imageView.setImageBitmap(b)`; da un thread che non sia il Main Thread.
- L'esecuzione della precedente istruzione deve essere demandata al Main Thread.
 - ▶ Si può utilizzare il metodo `runOnUiThread(Runnable action)` fornito dalla classe `Activity`.
 - ▶ Se il metodo è invocato da un thread che non sia il Main Thread, la sua esecuzione è accodata nella coda degli eventi del Main Thread (diversamente è eseguito immediatamente).

Uso dei Worker Thread – Esempio III

Secondo Tentativo

```
public void onClick(View v) {

    Thread worker = new Thread(new Runnable(){
        public void run() {
            final Bitmap b = loadImage("http://site.com/img.png");

            runOnUiThread(new Runnable(){
                public void run() {
                    imageView.setImageBitmap(b);
                }
            });
        }
    });

    worker.start();
}

private Bitmap loadImage(String url){/* Long-Running Task */}
```

Uso dei Worker Thread – Esempio IV

- Il codice è thread-safe, quindi funziona ma...
 - ▶ L'uso di più thread innestati rende molto difficile la comprensione del codice scritto.
 - ▶ È chiaro come questo tipo di approccio possa diventare poco soddisfacente non appena la complessità del codice cresce.
- Per ovviare a queste problematiche Android propone due diversi approcci:
 - ▶ **Handler**, valido come approccio generale per la comunicazione tra Worker Thread e Main Thread;
 - ▶ **AsyncTask**, semplifica l'esecuzione dei worker thread quando devono essere eseguiti compiti che hanno come effetto azioni sull'interfaccia utente.

Outline

- 1 Processi in Android
- 2 Il Main Thread
- 3 Worker Thread
- 4 Handler**
 - Android Looper
- 5 Async Task



Handler

- Consente di accodare un messaggio alla coda dei messaggi di un Thread definito secondo il pattern **MessageLoop** (**EventLoop**)
 - ▶ **Il Main Thread di Android è di fatto un MessageLoop**
- Generalmente utilizzato per la comunicazione tra il Main Thread e uno o più Worker Thread
- Se non diversamente specificato, quando si instancia un Handler quest'ultimo è automaticamente associato al flusso di controllo thread che lo crea
 - ▶ Quindi un handler può essere istanziato in un MessageLoop Thread oppure deve essere definito esplicitamente il MessageLoop a cui deve fare riferimento!

Handler – Esempio I

- Ogni handler deve estendere da `android.os.Handler`
- Il metodo `handleMessage()` consente di intercettare i messaggi ricevuti dall'handler ed inviati da altri thread

Handler per ImageDownloaderTask

```
public class MyHandler extends Handler {  
  
    public static final int IMAGE_DOWNLOADED_MSG = 1;  
    public static final String NEW_IMG = "new-img";  
    private ImageView imageView;  
  
    public MyHandler(Looper looper, ImageView iv){  
        super(looper);  
        this.imageView = iv;  
    }  
}
```

Handler – Esempio II

```
@Override
public void handleMessage(Message msg) {
    switch(msg.what){
        case IMAGE_DOWNLOADED_MSG:
            byte[] blob = msg.getData().getByteArray(NEW_IMG);
            Bitmap img = BitmapFactory.decodeByteArray(blob, 0, blob.
                length);
            imageView.setImageBitmap(img);
            break;
        }
    }
}
```

Creazione dell'Handler

```
Handler h = new MyHandler(Looper.getMainLooper(),imageView);
```


Handler – Esempio III

WorkerThread che utilizza l'Handler

```
public void onClick(View v) {  
    new Thread(new Runnable(){  
        public void run() {  
            final Bitmap img = loadImage("http://site.com/img.png");  
            final byte[] blob = convertImage(img);  
  
            final Bundle b = new Bundle();  
            b.putByteArray(MyHandler.NEW_IMG, blob);  
  
            final Message msg = new Message();  
            msg.what = MyHandler.IMAGE_DOWNLOADED_MSG;  
            msg.setData(b);  
  
            h.sendMessage(msg);  
        }  
    }).start();  
}
```

Message e Bundle – Note

- Su un istanza di un handler è possibile invocare il metodo `sendMessage()` per trasmettere all'handler un messaggio
 - ▶ Il messaggio sarà processato dall'handler nel Looper scelto...
- Il messaggio trasmesso deve essere di tipo **Message**
 - ▶ Il campo **what** consente di specificare un valore intero per discriminare il messaggio ricevuto.
 - ▶ Esistono anche altri campi pubblici per specificare il contenuto del messaggio (`arg1`, `arg2`, `obj`, ...)
- Per definire il contenuto di un messaggio meglio utilizzare un oggetto di tipo **Bundle**
 - ▶ Consente di definire il contenuto secondo il pattern chiave-valore

Android Looper

- E' sempre possibile ottenere l'istanza del MessageLoop associato al Main Thread mediante il metodo `Looper.getMainLooper()`
- In generale, la classe Looper consente di inizializzare qualunque thread secondo il patter MessageLoop

LooperThread – Esempio

```
class LooperThread extends Thread {  
    public Handler handler;  
  
    public void run() {  
        Looper.prepare();  
  
        handler = new Handler() {  
            public void handleMessage(Message msg) { /* manage msgs */ }  
        };  
  
        Looper.loop();  
    }  
}
```

Outline

- 1 Processi in Android
- 2 Il Main Thread
- 3 Worker Thread
- 4 Handler
 - Android Looper
- 5 Async Task



Async Task

- Un Async Task consente di eseguire computazione *asincrona* sull'interfaccia utente.
 - ▶ Permette di specificare quale porzione di codice deve essere eseguita in background, da un worker thread, e quale invece deve essere demandata al Main Thread.
 - ▶ Non richiede di creare esplicitamente il worker thread.
- Un Async Task può essere creato estendendo la classe `android.os.AsyncTask`.
 - ▶ Deve essere implementato almeno il metodo `doInBackground()`, in cui deve essere inserito il codice la cui esecuzione deve essere demandata al worker thread.
- Può essere eseguito richiamando il metodo `execute()` sull'istanza dell'oggetto.

Async Task – Esempio

ImageDownloader con Async Task

```
class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {  
  
    protected Bitmap doInBackground(String... urls) {  
        Bitmap b = downloadImage(urls[0]);  
        return b;  
    }  
  
    protected void onPostExecute(Bitmap result) {  
        imageView.setImageBitmap(result);  
    }  
}
```

```
public void onClick(View v) {  
    DownloadImageTask task = new DownloadImageTask();  
    task.execute("http://site.com/img.png");  
}
```

Async Task – Dettagli I

- In un Async Task possono essere ridefiniti quattro diversi metodi: `onPreExecute()`, `doInBackground()`, `onProgressUpdate()` e `onPostExecute()`.
- Ciascuno ha associata una propria semantica d'esecuzione.

`void onPreExecute()`

- È il primo metodo che viene invocato successivamente alla richiesta di esecuzione del task stesso.
- La sua esecuzione è gestita dal Main Thread.
- Utilizzato generalmente per le impostazioni di setup del task.

Async Task – Dettagli II

`Result doInBackground(Params... params)`

- È Invocato subito dopo la terminazione di `onPreExecute()`.
- Viene eseguito in un worker thread opportunamente creato.
- Riceve in ingresso i parametri passati al metodo `execute()` richiamato sull'istanza del task.
- Il risultato ritornato dal metodo è passato come parametro di input al metodo `onPostExecute()`.
- Può richiamare il metodo `publishProgress()` per eseguire computazione minimale sul Main Thread.

`void onProgressUpdate(Progress... params)`

Async Task – Dettagli III

- Eseguito sul Main Thread ogni volta che nel metodo `doInBackground()` viene richiamato il metodo `publishProgress()`.
- Riceve in ingresso i parametri passati al metodo `publishProgress()`.

`void onPostExecute(Result res)`

- Eseguito sul Main Thread successivamente alla terminazione del metodo `doInBackground()`.
- Il risultato ritornato dalla `doInBackground()` è passato come parametro di input al metodo.
- La classe `AsyncTask<Params,Progress,Result>` è una classe parametrica che richiede tre parametri che rappresentano:

Async Task – Dettagli IV

- ▶ Il primo (**Params**), è il tipo associato ai parametri che possono essere passati al metodo `doInBackground`.
 - ▶ Il secondo (**Progress**), è il tipo associato ai parametri che possono essere passati al metodo `onProgressUpdate`.
 - ▶ Il terzo (**Result**), è il tipo associato al parametro di ritorno del metodo `doInBackground()` e quindi al parametro in ingresso del metodo `onPostExecute()`.
- Tali parametri possono essere specificati con qualunque tipo di dato, o con il tipo **Void** che rappresenta l'assenza di parametri.

Async Task – Esempio Completo I

Esempio - Download di n File

```
class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {  
    private int nFileDownloaded;  
  
    protected void onPreExecute(){  
        nFileDownloaded = 0;  
        setupDownloadManager();  
    }  
  
    protected Long doInBackground(URL... urls) {  
        long bytesDownloaded = 0;  
  
        for(int i = 0; i < urls.length; i++){  
            bytesDownloaded += downloadFile(urls[i]);  
            nFileDownloaded++;  
            publishProgress(nFileDownloaded);  
        }  
    }  
}
```

Async Task – Esempio Completo II

```
    return bytesDownloaded;
}

protected void onProgressUpdate(Integer n) {
    updateDownloadCounter(n);
}

protected void onPostExecute(Long bytes) {
    updateTotalBytesDownloadedCounter(bytes);
}
}

public void onClick(View v) {
    DownloadFilesTask task = new DownloadFilesTask();
    task.execute(url1, url2, url3);
}
```





Linee guida per l'uso di Async Task

- Tutti gli Async Task devono essere creati nel Main Thread.
 - ▶ Solo il Main Thread può richiamare il metodo `execute()` sull'istanza di un task.
- Su ciascuna istanza di Async Task, il metodo `execute()` può essere richiamato una sola volta.
 - ▶ Le successive invocazioni provocano un'eccezione.
- Non devono essere chiamati *manualmente* i metodi `onPreExecute()`, `doInBackground()`, `onProgressUpdate()` e `onPostExecute()` sull'istanza del task.

Async Task – Ordine di Esecuzione

- **Tutti gli Async Task creati ed eseguiti all'interno di una stessa applicazione fanno riferimento allo stesso worker thread!**
 - ▶ Esempio: *Se due diversi async task sono eseguiti all'interno della stessa applicazione, il metodo `doInBackground()` del secondo task potrà essere eseguito solo successivamente alla terminazione dell'esecuzione dello stesso metodo del primo task.*
 - ▶ Risulta quindi comunque necessario far attenzione ai compiti (potenzialmente bloccanti) che devono essere eseguiti nel metodo `doInBackground()`.
- Qualora si abbia la necessità di eseguire compiti in parallelo, devono essere sfruttati altri meccanismi. Ad esempio:
 - ▶ Eseguire il task richiamando la funzione `executeOnExecutor()` al posto della funzione `execute()`, specificando l'executor sul quale eseguire il task.
 - ▶ Avvalersi del componente **Service**.

Riferimenti - Risorse Online

-  **Android Developers - Guide**
» <https://developer.android.com/guide/>
-  **Android Developers - API Reference**
» <https://developer.android.com/reference/>
-  **Android Developers - Samples**
» <https://developer.android.com/samples/>
-  **Android Developers - Design & Quality**
» <https://developer.android.com/design/>

Riferimenti - Libri

-  Zigurd Mednieks, Laird Dornin, G. Blake Meike, Masumi Nakamura
Programming Android
O'Reilly, 2011
-  Chris Haseman, Kevin Grant
Beginning Android Programming: Develop and Design
Peachpit Press, 2013
-  Ronan Schwarz, Phil Dutson, James Steele, Nelson To
The Android Developer's Cookbook : Building Applications with the Android SDK
Addison-Wesley, 2013
-  Theresa Neil
Mobile Design Pattern Gallery: UI Patterns for Smartphone App
O'Reilly, Second Edition, 2014