# UI Thread & Traitement long

Rémi Forax

# UI Thread

Toutes les méthodes **on**BlahBlah() d'une activité (ou d'un Fragment) sont appelées par l'UI Thread

C'est la **seule thread** qui a le droit de faire les changements graphiques et d'appeler les listeners, etc

Par défaut, il n'est donc pas nécessaire de gérer la concurrence car il n'y en a pas !

# Traitement long

Pour les traitements long, il n'est pas possible de faire ceux-ci dans l'UI car cela freeze l'application !

Il faut démarrer une autre thread (worker thread)

```
Thread workerThread = new Thread(...);
workerThread.start()
```

mais celle-ci ne peut pas faire de changement graphique

Si une autre thread fait des changement graphiques au mieux, on a une exception, souvent cela ne fait rien, des fois ça crash

# Thread de traitement et rafraichissement graphique

Comme un worker thread ne peut pas faire de traitement graphique

- L'idée consiste dans la worker thread à poster un évènenement contenant un Runnable dans la queue d'évènement
- Lorsque cet évèvement spécial est traité par la queue d'évènement, la méthode run du Runnable est appelée par l'UI Thread

# Poster un Runnable

La classe Activity possède une méthode runOnUiThread

```
final Button button = ...
runOnUiThread(new Runnable() {
  public void run() {
    button.setEnabled(false);
  }
});
```

Si la thread courante est l'UI Thread appel run() directement, sinon post un évènement dans la queue d'évènement

# Poster un Runnable

android.os.Handler possède une API plus complète

- postAtFrontOfQueue(Runnable)
  - Poster en début de queue
- postAtTime(Runnable, long time)
  - Exécute le Runnable après la date “time”
    - voir SystemClock.uptimeMillis
- postDelayed(Runnable, long delay)
  - Le Runnable ne peut s'exécuter que le delai passé

Un Handler sur l'UI Thread s'obtient avec new Handler() dans l'UI Thread

# Problèmes liés au post de Runnable

Il est possible de noyer l'UI Thread avec plein des Runnable qui doivent s'exécuter ce qui ralenti le traitement des autres évènements

- Pour l'utilisateur, l'application devient lente

Problème producteur/consommateur !

- On a besoin d'un mécanisme qui ralenti la worker thread ou qui agglomère les changements graphiques

# AsyncTask

Permet de faire un traitement en tâche de fond avec un affichage de résultat intermédiaire

- Comme mettre à jour une progresse bar
- Afficher des logs graphiques

AsyncTask est paramétrée par

- Params : type des paramètre d'entrée
- Progress : résultat intermédiaire
- Result : résultat final

  donc ne suis pas la convention de code Java , grrr !

# AsyncTask

doInBackground(Params...),

- Exécuté dans la worker thread après onPreExecute(),  doit tester isCancelled(), peut appeler publishProgress() pour des résultats intermédiaire, renvoie un Result,

cancel()

- Arrête l'asyncTask, isCancelled() passe à vrai

onPreExecute()

- Exécuté dans la thread UI avant que la tâche s'exécute

onProgressUpdate(Progress...)

- Exécuté dans la thread UI suite à un appel à publishProgress(Progress...)

onPostExecute(Result)

- Exécuté dans la thread UI après doInBackGround()

# Tâche longue & changement de géométrie

Lorsqu'une tâche longue s'exécute, il est possible que la géométrie change :((

Dans ce cas l'Activity et toutes les View vont disparaitre et une nouvelle activité avec de nouvelles views vont être créés

Il ne faut donc pas stocker une référence à l'Activity (ni au Context, ni aux View) dans l'AsyncTask !!

les vues accédées **ne doivent pas** être des champs de l'activité, et lors d'un changement d'orientation, il faut

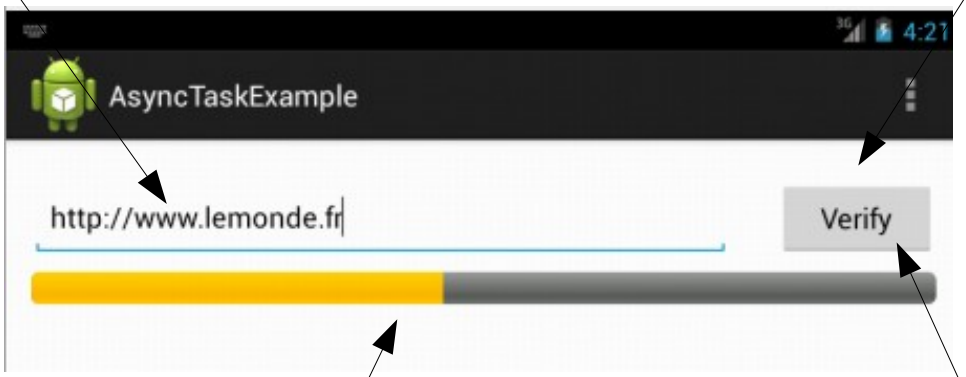
- Détacher l'AsyncTask de l'activiter
- Envoyer l'AsyncTask dans onRetainNonConfigurationInstance
- Récupérer l'AsyncTask dans la nouvelle Activity
- Puis ré-attacher la nouvelle Activity à l'AsyncTask

# Exemple

## Chargement d'une page Web en tâche de fond

1. On entre l'URL ici

2. On demande la vérification




3. Pendant la recherche DNS, le bouton est grisé

4. on affiche une barre de progression pendant le download (qui peut être interrompu)

# Exemple : Code sans UI

```
HttpURLConnection connection = (HttpURLConnection)url.openConnection();
int responseCode = connection.getResponseCode();
if (responseCode < 100 || responseCode >= 400) {
  ...
  return ...
}
int contentLength = httpURLConnection.getContentLength();
//TODO update UI
int numberOfBytes = 0;
byte[] buffer = new byte[8192];
InputStream stream = connection.getInputStream();
try {
  int read;
  while ((read = stream.read(buffer)) != -1) {
    numberOfBytes += read;
    //TODO update UI
    if (/* test user cancellation */) {
      break;
    }
  }
} finally {
  stream.close();
}
```

# Code de l'activité

```
public class MainActivity extends Activity {
  private AsyncDownload asyncTask;

  @Override protected void onCreate(Bundle savedBundle) {
    super.onCreate(savedBundle);
    setContentView(R.layout.activity_main);

    final EditText edit = (EditText)findViewById(R.id.edit);
    ProgressBar progressBar =
        (ProgressBar)findViewById(R.id.progress_bar);
    Button verify = (Button)findViewById(R.id.verify);

    verify.setOnClickListener(new View.OnClickListener() {
        ...
    });
    ...
```

# Code de l'activité (2)

```
public class MainActivity extends Activity {
 ...
  @Override protected void onCreate(Bundle savedInstanceState) {
    ...
    verify.setOnClickListener(new View.OnClickListener() {
        @Override public void onClick(View v) {
          if (asyncTask != null) {      // cancel pending task if necessary
            asyncTask.cancel(true);
            asyncTask = null;
          }
          URL url;
          try {
            url = new URL(edit.getText().toString());
          } catch (MalformedURLException e) {
             Toast.makeText(MainActivity.this, "malformed URL " + e.getMessage(),...).show();
             return;
          }
          verify.setEnabled(false);
          asyncTask = new AsyncDownload(MainActivity.this);
          asyncTask.execute(url);
        }
    });
    ...
```

# Code de l'activité (suite)

```
public class MainActivity extends Activity {
  private AsyncDownload asyncTask;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    ...
    asyncTask = (AsyncDownload)getLastNonConfigurationInstance();
    if (asyncTask != null) {
      asyncTask.activity = this;  // volatile write
    }
  }
  @Override
  public Object onRetainNonConfigurationInstance() {
    if (asyncTask == null) {
      return null;
    }
    asyncTask.activity = null;  // not linked to the activity anymore
    return asyncTask;
  }
```

# AsyncTask.doInBackground

```
static class AsyncDownload extends AsyncTask<URL, Integer, IOException> {
    volatile MainActivity activity;
    public AsyncDownload(MainActivity activity) {
        this.activity = activity;
    }
    protected IOException doInBackground(URL... params) {
      try {
        URL url = params[0];
        HttpURLConnection connection = url ...
        ...
        final int contentLength = connection.getContentLength();
        final MainActivity mainActivity = this.activity;   // volatile read
        if (mainActivity != null) {
          runOnUiThread(new Runnable() {
              public void run() {
                Button verify = (Button)mainActivity.findViewById(R.id.verify);
                ProgressBar progressBar = (ProgressBar)mainActivity.findViewById(R.id.progress_bar);
                verify.setEnabled(true);
                if (contentLength == -1) {
                  progressBar.setIndeterminate(true);
                } else {
                  progressBar.setMax(contentLength);
                } } });
        }
```

# AsyncTask.doInBackground

```
    try {
      ...
      int numberOfBytes = 0;
      byte[] buffer = new byte[8192];
      InputStream stream = connection.getInputStream();
      try {
        int read;
        while ((read = stream.read(buffer)) != -1) {
          numberOfBytes += read;
          publishProgress(numberOfBytes);
          if (isCancelled()) {
            break;   // user cancellation
          }
        }
        return null;
      } finally {
        stream.close();
      }
    } catch(IOException e) {
      return e;
    }
  }
}
```

# AsyncTask : code de l'UI

```
@Override protected void onPreExecute() {
  ProgressBar progressBar = ...;
  progressBar.setProgress(0);
}
@Override protected void onProgressUpdate(Integer... values) {
  ProgressBar progressBar = ...;
  progressBar.setProgress(values[values.length – 1]);
}
@Override protected void onCancelled() {
  onPostExecute(null);
}
@Override protected void onCancelled(IOException result) {
  onPostExecute(result);
}
@Override protected void onPostExecute(IOException result) {
  ...
}
```

# AsyncTask : code de l'UI (2)

```
@Override protected void onPostExecute(IOException result) {
  MainActivity activity = this.activity;
  if (activity != null) {
    activity.asyncTask = null;
    Button verify = ...; ProgressBar progressBar = ...;
    verify.setEnabled(true);
    progressBar.setIndeterminate(false);
    progressBar.setProgress(0);
  }
  if (result != null) {
    Toast.makeText(activity, result.getMessage(),
                           Toast.LENGTH_SHORT).show();
  }
}
```

# En résumé

**Tous les traitements graphiques doivent être fait dans l'UI Thread**

- Activity.runOnUiThread(Runnable) permet de poster du code qui sera exécuté dans l'UI Thread
- AsyncTask permet de séparer la worker thread de la thread UI
  - Il faut attacher/détacher l'activité à l'AsyncTask
  - NE PAS Les Views à metre à jour doivent être en champs et pas en variable locale dans onCreate()
  - Si dans doOnBackground on souhaite accéder à des Views, il faut que l'Activity dans AsyncTask soit déclarée volatile