Android: Event Handler Blocking, Android Inter-Thread, Process Communications

10/11/2012

Y. Richard Yang
Outline

- Admin
- Android
  - Basic concepts
    - Activity, View, External Resources, Listener
  - Inter-thread communications
    - Handler, ASyncTask
  - Inter-process communications
    - Intent
Admin.

- HW2
  - Due: Friday @ 11:55 pm
Recap: Mobile GUI App Workflow

App lifecycle callbacks/custom:
- start
- pause
- ...

App

Display

Composite

Display

Composite

Display

Composite

Display

Composite

Display

Composite

Display

Composite

Display

Composite

Display

Composite

Display

Composite

Display

Composite

Display

Composite

Event Handler

Data/Model

Event Handler

Data/Model
Recap: Android UI App Basic Concepts

- Activity

- View/ViewGroup
  - External definition of views in XML
  - findViewById() to reduce coupling

- Link view events to event handlers
  - set...Listener()
Example: TipCalc

Set listener:

```java
public class TipCalcActivity extends Activity
    implements OnClickListener {

    ...     
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tip_calc);

        Button calc = (Button) findViewById(R.id.calculate);
        calc.setOnClickListener(this);
    }
    ...
}
```
Example: TipCalc Event Handler

Handler:

```java
@Override
public void onClick(View arg0) {
    EditText amountText =
        (EditText) findViewById(R.id.amount_value);

    // get input from UI
    double amt = Double.parseDouble(amountText.getText().toString);

    // compute output
    double tipD = amount * 0.15;

    // update UI
    String tipT = String.format("%.2f", tipD);
    TextView tipText = (TextView) findViewById(R.id.tip_value);
    tipText.setText(tipT);
}
```
Event Handler Execution

- Event handler executed by the main/UI thread

UI events

Looper

message

message

system events

UI (main) thread

http://www.java2s.com/Open-Source/Android/android-core/platform-frameworks-base/android/os/Looper.java.htm
Event Handler and Responsiveness

- Event handler blocks events in the msg queue from being processed => slow running handler leads to no UI response

Responsiveness: Numbers (Nexus One)

- ~5-25 ms - uncached flash reading a byte
- ~5-200+(!) ms - uncached flash *writing tiny amount*
- 100-200 ms - human perception of slow action
- 108/350/500/800 ms - ping over 3G. varies!
- ~1-6+ seconds - TCP setup + HTTP fetch of 6k over 3G
Event Handler and ANR

- Android system detects no response
  - Main thread ("event"/UI) does not respond to input in 5 sec
Example

- play_music
Discussion

- What are some design options if an event may take a while to be processed
  - Time consuming loading process, e.g., slow `onCreate`
  - Heavy computation, e.g., voice recognition, update map display
  - Networking access
  - ...

Typical Design Guidelines

- Notify user
  - E.g., progress bar, progress dialog
  - A splash screen

- If possible, non-blocking, incremental update UI
  - E.g., gradual add items to map

- Whenever possible, release UI thread ASAP
  - Keep event handler simple
  - Post heavy processing off the UI thread
Example: Background Thread

- Use a background thread to do the task
- Background thread behavior controlled by state
- State controlled by event handler

- See PlayMusicActivity
Service: Working in Background

Why Android Service:
- Faceless components that typically run in the background
  - the user is not directly interacting with the application.
  - NO GUI, higher priority than inactive Activities
- The system to schedule work for the service, to be run until the service or someone else explicitly stop it.

Note
- A Service is not a separate process. The Service object itself does not imply it is running in its own process; unless otherwise specified, it runs in the same process as the application it is part of.
- A Service is not a thread. It is not a means itself to do work off of the main thread (to avoid Application Not Responding errors).
- We will cover Service more in next class.
Background Thread vs UI Thread

- Problem:
  - Background thread and UI thread are running concurrently and may have race conditions if they modify UI simultaneously (e.g., UI switches to a different orientation)
  - A major sin of programming: concurrency bugs

- Example: LoadingScreen
Example: LoadingScreen

```java
public class LoadingScreen extends Activity implements Runnable {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.loading);

        // start a new thread to load
        Thread thread = new Thread(this);
        thread.start();
    }

    public void run() {
        longRunningTask();
        setContentView(R.layout.main);
    }

    ...
}
```

Conflict with UI thread
Solution

- Background thread does not directly modify UI: send msg to UI thread, who processes the msg
Android Handler

- Android’s mechanism to send and process `Message` and Runnable objects associated with a thread’s `MessageQueue`.
- Each Handler instance is associated with a single thread and that thread's message queue
  - A handler is bound to the thread / message queue of the thread that creates it
  - From that point on, it will deliver messages and runnables to that message queue
  - That thread processes msgs
Android Handler

Diagram showing the relationship between threads, handlers, messages, and the message queue.
Using Handler: Examples

- There are two main uses for a Handler

  - to schedule messages and runnables to be executed as some point in the future
    - postDelayed(Runnable, delayMillis)

  - to enqueue an action to be performed on a different thread than your own.
    - post(Runnable)
public class MyActivity extends Activity {

    // Need handler for callbacks to the UI thread
    final Handler mHandler = new Handler();

    // Create runnable task to give to UI thread
    final Runnable mUpdateResultsTask = new Runnable() {
        public void run() {
            updateResultsInUi();
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // [ . . . ]
    }
}
protected void startLongRunningOperation() {

    // Fire off a thread to do some work that we shouldn't do directly in the UI thread
    Thread t = new Thread() {
        public void run() {
            mResults = doSomethingExpensive();
            mHandler.post(mUpdateResultsTask);
        }
    };
    t.start();
}

private void updateResultsInUi() {

    // Back in the UI thread -- update our UI elements based on the data in mResults
    [ . . . ]
}
public class LoadingScreen extends Activity implements Runnable {
    private Handler mHandler = new Handler(); // UI handler

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.loading);

        // start a new thread to load
        Thread thread = new Thread(this);
        thread.start();
    }

    public void run() {
        longTask();
        mHandler.post(mSetFinalViewTask);
    }

    private Runnable mSetFinalViewTask = new Runnable() {
        public void run() {
            setContentView(R.layout.main);
        }
    };
}
Example: BackgroundTimer
private Handler mHandler = new Handler(); // UI handler
private Runnable longTask = new Runnable() {

    // processing thread
    public void run() {
        while (notFinished) {
            // doSomething

            mHandler.post(taskToUpdateProgress);
        }
    }

    // mHandler.post(taskToUpdateFinalResult)

};

Thread thread = new Thread(longTask);
thread.start();
AsyncTask as Abstraction

```java
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) { // on some background thread
        int count = urls.length; long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
        }
        return totalSize;
    }
    protected void onProgressUpdate(Integer... progress) { // on UI thread!
        setProgressPercent(progress[0]);
    }
    protected void onPostExecute(Long result) { // on UI thread!
        showDialog("Downloaded " + result + " bytes");
    }
}

new DownloadFilesTask().execute(url1, url2, url3); // call from UI thread!
```

Outline

- Admin
- Android
  - Basic concepts
    - Activity, View, External Resources, Listener
  - Inter-thread communications
    - Handler, ASyncTask
  - Inter-process communications
    - Intent, Broadcast, BroadcastReceiver, Service
Inter-Process Communications (IPC)

- Inter-thread communications are for one activity
- Inter-process communication is designed to promote the development of complex applications, by allowing developers to reuse existing data and services from other applications.
- One may also use IPC for intra-app communications (e.g., between Activities of the same app)
Discussion: IPC Use Cases

- One component of Android sends messages to another component of Android
- An IPC message in Android is called Intent
Target: Activity

- startActivity() or startActivityForResult() to launch an activity or get an existing activity to do something new.
Target: Service

- `startService()` to initiate a service or deliver new instructions to an ongoing service.
- `bindService()` to establish a connection between the calling component and a target service. It can optionally initiate the service if it's not already running.
Target: **BroadcastReceiver**

- `broadcastIntent()` to send messages to all interested broadcast receivers.
  - Many kinds of broadcasts originate in system code, e.g., boot, battery low
Target: Data Provider

- `startActivityForResult()` may target to a data provider (e.g., Contacts)
Example: A SocialApp
Android Application Component: Gang of Four
Outline

- Admin
- Android
  - Basic concepts
    - Activity, View, External Resources, Listener
  - Inter-thread communications
    - Handler, ASyncTask
  - Inter-process communications
    - Intent
Intent

- An intent is an abstract description of an operation to be performed.
  - Indicate operations from your own or others

http://developer.android.com/reference/android/content/Intent.html
Intent Data Structure

- Primary pieces of info in an Intent
  - Action: The general action to be performed
    - ACTION_VIEW, ACTION_DIAL, ACTION_EDIT, ...
    - Your own definition of strings
  - Data: a URI
    - tel:123
    - content://contacts/people/1
    - http://zoo.cs.yale.edu/classes/cs434
    - hotel://name/Omni_New_Haven

- Other attributes
  - Category
  - Type (MIME type)
  - Component (class name)
  - Extras (key-value store)
Intent Resolution: Explicit Intent

 Explicit Intent: specifies the exact class to run

```java
public class IntentController extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.intentcontroller);

        // launch tip cal button
        Button tipBtn = (Button) findViewById(R.id.tipButton);
        tipBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent tipIntent = new Intent(IntentController.this, TipCal.class);
                startActivity(tipIntent);
            }
        });
    }
}
```
Explicit Intent and Manifest

Make sure AndroidManifest.xml announces activities to be started

```xml
<application
    android:icon="@drawable/icon"
    android:label="@string/app_name" >
    <activity
        android:name=".IntentController"
        android:label="IntentController" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <activity
        android:name=".TipCal"
        android:label="TipCal" >
    </activity>
</application>
```

Shown in Launcher

Announce class

See IntentController
Explicit Intent

Yelp

To: MapActivity

Name: MapActivity

Map App

Only the specified destination receives this message