20CSI605 - Mobile Application Development

Background Tasks

Background Tasks

There are several ways to do background processing in Android.

Two of those ways are:

1. You can do background processing directly, using the **AsyncTask** class.
2. You can do background processing indirectly, using the **Loader framework** and then the AsyncTaskLoader class.

Background Tasks

The UI thread

- When an Android app starts, it creates the *main thread*, which is often called the *UI thread*.
- The UI thread dispatches events to the appropriate user interface (UI) widgets. The UI thread is where your app interacts with components from the Android UI toolkit

Android's thread model has two rules:

- ✔ Do not block the UI thread.
- ✔ Do UI work only on the UI thread.

Background Tasks

The UI thread

- The UI thread needs to give its attention to drawing the UI and keeping the app responsive to user input.
- If everything happened on the UI thread, long operations such as network access or database queries could block the whole UI.
- From the user's perspective, the app would appear to hang.
- Even worse, if the UI thread were blocked for more than a few seconds (about 5 seconds currently) the user would be presented with the "application not responding" (ANR) dialog.

Background Tasks

The UI thread

To make sure your app doesn't block the UI thread:

- Complete all work in less than 16 ms for each UI screen.
- Don't run asynchronous tasks and other long-running tasks on the UI thread. Instead, implement tasks on a background thread using AsyncTask (for short or interruptible tasks) or AsyncTaskLoader (for tasks that are high-priority, or tasks that need to report back to the user or UI).

Background Tasks

AsyncTask

- A worker thread is any thread which is not the main or UI thread.
- Use the AsyncTask class to implement an asynchronous, long-running task on a worker thread.
- AsyncTask allows you to perform background operations on a worker thread and publish results on the UI thread without needing to directly manipulate threads or handlers.

When AsyncTask is executed, it goes through four steps:

1.onPreExecute()

2. doInBackground(Params...)

3. onProgressUpdate(Progress...)

4. onPostExecute(Result)

Background Tasks

AsyncTask Usage

- To use the AsyncTask class, define a subclass of AsyncTask that overrides the doInBackground(Params...) method (and usually the onPostExecute(Result) method as well).

AsyncTask Parameters

- "Params" specifies the type of parameters passed to doInBackground() as an array.
- "Progress" specifies the type of parameters passed to publishProgress() on the background thread.
- "Result" specifies the type of parameter that doInBackground() returns.

Background Tasks

Executing an AsyncTask

- After you define a subclass of AsyncTask, instantiate it on the UI thread. Then call execute() on the instance, passing in any number of parameters.

  For example, to execute the DownloadFilesTask task defined above, use the following line of code:

```
new DownloadFilesTask().execute(url1, url2, url3);
```

Background Tasks

Cancelling an AsyncTask

You can cancel a task at any time, from any thread, by invoking the cancel() method.

- The cancel() method returns false if the task could not be cancelled, typically because it has already completed normally.
- To find out whether a task has been cancelled, check the return value of isCancelled() periodically from doInBackground(Object[])
- After an AsyncTask task is cancelled, onPostExecute() will not be invoked after doInBackground() returns.
- By default, in-process tasks are allowed to complete. To allow cancel() to interrupt the thread that's executing the task, pass true for the value of mayInterruptIfRunning.

Background Tasks

Limitations of AsyncTask

AsyncTask is impractical for some use cases:

- Changes to device configuration cause problems.

When device configuration changes while an AsyncTask is running, for example if the user changes the screen orientation, the activity that created the AsyncTask is destroyed and re-created.

- Old AsyncTask objects stay around, and your app may run out of memory or crash.

If the activity that created the AsyncTask is destroyed, the AsyncTask is not destroyed along with it. For example, if your user exits the app after the AsyncTask has started, the AsyncTask keeps using resources unless you call cancel().

Background Tasks

When to use AsyncTask:

- ✔Short or interruptible tasks.
- ✔Tasks that don't need to report back to UI or user.
- ✔Low-priority tasks that can be left unfinished.

Background Tasks

Internet connection

- Most Android apps engage the user with useful data.
- That data might be news articles, weather information, contacts, game statistics, and more. Often, data is provided over the network by a web API.

Network security

- Network transactions are inherently risky, because they involve transmitting data that could be private to the user.
- People are increasingly aware of these risks, especially when their devices perform network transactions, so it's very important that your app implement best practices for keeping user data secure at all times.

Background Tasks

Security best practices for network operations include:

- Use appropriate protocols for sensitive data. For example for secure web traffic, use the HttpsURLConnection subclass of HttpURLConnection.
- Use HTTPS instead of HTTP anywhere that HTTPS is supported on the server, because mobile devices frequently connect on insecure networks such as public Wi-Fi hotspots.
- Don't use localhost network ports to handle sensitive interprocess communication (IPC), because other apps on the device can access these local ports.
- Don't trust data downloaded from HTTP or other insecure protocols. Validate input that's entered into a WebView and responses to intents that you issue against HTTP.

Background Tasks

Connecting and downloading data

- In the worker thread that performs your network transactions, for example within your override of the doInBackground() method in an AsyncTask, use the HttpURLConnection class to perform an HTTP GET request and download the data your app needs

1.To obtain a new HttpURLConnection, call URL.openConnection() using the URI that you've built.

2.Set optional parameters.

3.Open an input stream using the getInputStream() method, then read the response and convert it into a string.

4.Call the disconnect() method to close the connection.

Background Tasks

Uploading data

- If you're uploading (posting) data to a web server, you need to upload a *request body*, which holds the data to be posted.

  To do this:

- ✔ Configure the connection so that output is possible by calling setDoOutput(true). By default, HttpURLConnection uses HTTP GET requests.When setDoOutput is true, HttpURLConnection uses HTTP POST requests instead.
- ✔ Open an output stream by calling the getOutputStream() method.

Background Tasks

Managing the network state

- Making network calls can be expensive and slow, especially if the device has little connectivity. Being aware of the network connection state can prevent your app from attempting to make network calls when the network isn't available.

To check the network connection, use the following classes:

- ✔ **ConnectivityManager** answers queries about the state of network connectivity. It also notifies apps when network connectivity changes.
- ✔ **NetworkInfo** describes the status of a network interface of a given type (currently either mobile or Wi-Fi).

Background Tasks

Broadcasts

- *Broadcasts* are messages that the Android system and Android apps send when events occur that might affect the functionality of other apps.
- In general, broadcasts are messaging components used for communicating across apps when events of interest occur.

There are two types of broadcasts:

1. *System broadcasts* are delivered by the system.
2. *Custom broadcasts* are delivered by your app.

Background Tasks

Service

- A *service* is an app component that performs long-running operations, usually in the background. Unlike an Activity, a service doesn't provide a user interface (UI).
- Services are defined by the Service class or one of its subclasses.

A service can be *started*, *bound*, or both:

- A *started service* is a service that an app component starts by calling startService().
- Use started services for tasks that run in the background to perform long-running operations. Also use started services for tasks that perform work for remote processes.
- A *bound service* is a service that an app component binds to itself by calling bindService().

Background Tasks

How a service starts:

- An app component such as an Activity calls startService() and passes in an Intent. The Intent specifies the service and includes any data for the service to use.
- The system calls the service's onCreate() method and any other appropriate callbacks on the main thread. It's up to the service to implement these callbacks with the appropriate behavior, such as creating a secondary thread in which to work.
- The system calls the service's onStartCommand() method, passing in the Intent supplied by the client in step 1.

THANK YOU