Desarrollo Android

Arkaitz Garro

Master en desarrollo de Apps Móviles

Recursos de Internet

Conexión básica

Conectarse a Internet

Antes de conectarnos a Internet, debemos de solicitar los permisos para ello

```
<uses-permission android:name=”android.permission.INTERNET”/>
```

Conectarse a Internet

```
String myFeed = getString(R.string.my_feed);
try {
  URL url = new URL(myFeed);

  // Create a new HTTP URL connection
  URLConnection connection = url.openConnection();
  HttpURLConnection httpConnection = (HttpURLConnection)connection;

  int responseCode = httpConnection.getResponseCode();
  if (responseCode == HttpURLConnection.HTTP_OK) {
    InputStream in = httpConnection.getInputStream();
    processStream(in);
  }
}
catch (MalformedURLException e) {
  Log.d(TAG, "Malformed URL Exception.", e);
}
catch (IOException e) {
  Log.d(TAG, "IO Exception.", e);
}
```

Conectarse a Internet

Conectarse a Internet en el proceso principal de la UI, causará un `NetworkOnMainThreadException` en las últimas plataformas de Android

Tareas asíncronas

Utilizando Threads

Todos los componentes de Android, se ejecutan en el proceso principal, por lo tanto, cualquier proceso en cualquier componente puede bloquear el resto de componentes.

En Android, las actividades que no responden a un evento en entrada en 5 segundos, son consideradas como bloqueadas.

Es importante utilizar Threads para todas las operaciones que impliquen tiempo de procesado, para no interferir con la UI.

Tareas sencillas: AsyncTask

`AsyncTask` implementa uno de los mejores patrones para ejecutar Threads y sincronizarlos con la UI.

AsyncTask maneja la creación, gestión y sincronización, permitiendo ejecutar tareas y actualizar la UI cuando éstas se completan.

Es una solución perfecta para tareas cortas que tiene un reflejo en la UI.

Si una actividad es reiniciada, todas las AsyncTask son terminadas.

La tarea es definida por tres argumentos, `Params`, `Progress`, Result y por cuatro métodos `onPreExecute`, `doInBackground`, `onProgressUpdate` y `onPostExecute`.

Tareas sencillas: AsyncTask

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        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) {
        setProgressPercent(progress[0]);
    }

    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}

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

Recursos de Internet

Gestor de descargas

En Android 2.3 se introdujo el gestor de descargas con el fin de gestionar la descarga de ficheros

Es importante hacer uso del gestor, ya que asegura la descarga de ficheros entre sesiones de usuario, reinicios, fallos de conectividad...

```
String serviceString = Context.DOWNLOAD_SERVICE;
DownloadManager downloadManager;
downloadManager = (DownloadManager)getSystemService(serviceString);
```

Descarga de ficheros

```
String serviceString = Context.DOWNLOAD_SERVICE;
DownloadManager downloadManager;
downloadManager = (DownloadManager)getSystemService(serviceString);

Uri uri = Uri.parse("http://developer.android.com/shareables/
icon_templates-v4.0.zip");
DownloadManager.Request request = new Request(uri);
long reference = downloadManager.enqueue(request);
```

Descarga de ficheros

La referencia devuelta indica la descarga en marcha

Es posible especificar las restricciones bajo las que se va a realizar la descarga (Wi-Fi, 3G, Roaming...)

```
request.setAllowedNetworkTypes(Request.NETWORK_WIFI);
```

También podemos añadir nuevas cabeceras al objeto `Request`, o cambiar el `mimetype` devuelto por el servidor

Notificar la descarga

Para recibir la notificación de descarga completa, es necesario registrar un Receiver, que contendrá el ID del fichero descargado

```
IntentFilter filter = new IntentFilter(DownloadManager.ACTION_DOWNLOAD_COMPLETE);
```

```
BroadcastReceiver receiver = new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
    long reference = intent.getLongExtra(DownloadManager.EXTRA_DOWNLOAD_ID, -1);
    if (myDownloadReference == reference) {
        // TODO Do something with the file.
    }
  }
};

registerReceiver(receiver, filter);
```

Notificar la descarga

Es recomendable también crear un Receiver que se ejecute cuando el usuario ha seleccionado el fichero descargado se selecciona en la barra de notificaciones

```
IntentFilter filter = new
IntentFilter(DownloadManager.ACTION_NOTIFICATION_CLICKED);
```

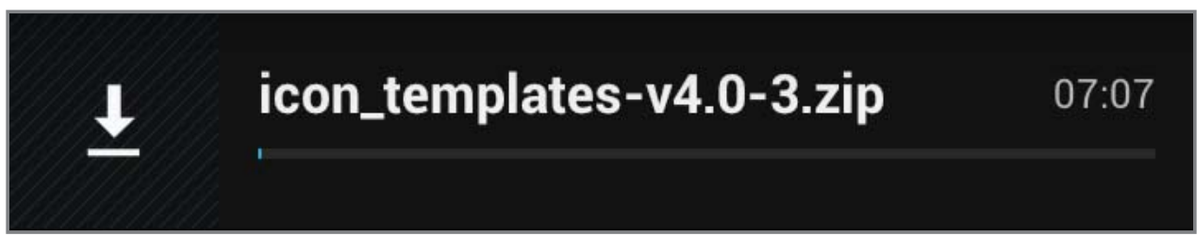
```
BroadcastReceiver receiver = new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
    String extraID = DownloadManager.EXTRA_NOTIFICATION_CLICK_DOWNLOAD_IDS;
    long[] references = intent.getLongArrayExtra(extraID);
    for (long reference : references)
      if (reference == myDownloadReference) {
        // Do something with downloading file.
      }
  }
};

registerReceiver(receiver, filter);
```

Personalizando las notificaciones

Por defecto, las descargas activas se muestran en la barra de notificaciones

El gestor de descargas nos permite personalizar la notificación (título y descripción) y su visibilidad



Personalizando las notificaciones

El método `setNotificationVisibility` permite modificar la visibilidad de las notificaciones

`Request.VISIBILITY_VISIBLE`: visible durante el proceso de descarga, desaparece al completarse.

`Request.VISIBILITY_VISIBLE_NOTIFY_COMPLETED`: visible durante el proceso de descarga, se mantiene al finalizar.

`Request.VISIBILITY_VISIBLE_ONLY_COMPLETION`: visible al finalizar la descarga.

`Request.VISIBILITY_HIDDEN`: no se muestra ninguna notificación.

Destino de descarga

Por defecto, todas las descargas son guardadas en el mismo directorio, utilizando nombres generados por el sistema

Para cada descarga, se puede especificar un destino, pero todas ellas deben ir al almacenamiento externo, además de tener el permiso apropiado

```
<uses-permission android:name=”android.permission.WRITE_EXTERNAL_STORAGE”/>

request.setDestinationUri(Uri.fromFile(f));
```

Lo normal, es almacenar los ficheros en el directorio propio de la aplicación

```
request.setDestinationInExternalFilesDir(this, Environment.DIRECTORY_DOWNLOADS,
                                         “Bugdroid.png”);
```

Cancelar descargas

El método remove del gestor de descargas permite cancelar descargas pendientes o en curso, o eliminar las ya descargadas

```
downloadManager.remove(REFERENCE_1, REFERENCE_2, REFERENCE_3);
```

Los ficheros asociados (completos o no) también son eliminados

Consultar el gestor de descargas

Es posible conocer el estado, progreso y detalles de las descargas gracias al método `query`

Se pueden consultar los siguientes datos:

`setFilterById`: obtener la descarga en función a una referencia.

`setFilterByStatus`: obtener las descargas en función de su estado; en ejecución, en pausa, fracasado o completado.

El cursor devuelto por estas consultas incluye columnas con los detalles de estas columnas

```
String serviceString = Context.DOWNLOAD_SERVICE;
DownloadManager downloadManager;
downloadManager = (DownloadManager)getSystemService(serviceString);

Query pausedDownloadQuery = new Query();
pausedDownloadQuery.setFilterByStatus(DownloadManager.STATUS_PAUSED);

Cursor pausedDownloads = downloadManager.query(pausedDownloadQuery);

int reasonIdx =
pausedDownloads.getColumnIndex(DownloadManager.COLUMN_REASON);
```

```
while (pausedDownloads.moveToNext()) {
    int reason = pausedDownloads.getInt(reasonIdx);
    String reasonString = "Unknown";
    switch (reason) {
        case DownloadManager.PAUSED_QUEUED_FOR_WIFI :
          reasonString = "Waiting to retry"; break;
        default : break;
    }
}

pausedDownloads.close();
```

Buenas prácticas

En general, las pequeñas y cortas pero frecuentes conexiones, son las que mayor impacto tienen sobre la batería

1. Descargar de una sola vez toda la información que sea posible.
2. Agrupar las conexiones y descargas, y enviarlas/recibirlas juntamente con otras conexiones.
3. Reutilizar las conexiones abiertas en lugar de iniciar unas nuevas. Esto mejora drásticamente el rendimiento y la latencia.
4. Evitar las actualizaciones programadas, y dejar en manos del usuario esta decisión.

Ejercicio