Android Connectivity & Google APIs

Lecture 5

Operating Systems Practical

2 November 2016

Multithreading

Accessing Online Content

Bluetooth Communication

Near Field Communication

Google APIs

Bibliography

Multithreading

Accessing Online Content

Bluetooth Communication

Near Field Communication

Google APIs

Bibliography

The Main UI Thread

- The UI thread is the main thread of an Android app
- Responsible for handling UI events
- The only one who can update UI elements
  - `CalledFromWrongThreadException` if other thread tries to do it
- BroadcastReceivers and Services (by default) run on UI thread

The Main UI Thread

- Computationally intensive and potentially blocking operations on the main UI thread
  - Block the thread
  - Prevent UI events from being dispatched
  - Prevent the user from interacting with the app
  - Generate ANR
- 2 rules:
  - No CPU intensive and blocking operations on the UI thread
  - UI toolkit API only from the UI thread

Using Java Thread

- Create worker thread for CPU intensive or blocking operations
- Create a new `Thread` instance and call `start()`
- Or implement the `Runnable` interface
- Manually send data back to the UI thread
- `Thread` and `Runnable`, the basis of:
  - `AsyncTask`
  - `IntentService`
  - `HandlerThread`
  - `ThreadPoolExecutor`

- Designed to execute asynchronous operations on a separate thread
  - Run operations on worker thread
  - Publish results to UI thread
- One class method that runs on the worker thread
- Several class methods that run on the UI thread

AsyncTask API

- `doInBackground()` method invoked on a worker thread
- `onPreExecute()`, `onPostExecute()`, and `onProgressUpdate()` invoked on the UI thread
- The value returned by `doInBackground()` is sent to `onPostExecute()`
- Call `publishProgress()` at any time from `doInBackground()` to execute `onProgressUpdate()`
- Launch: `execute()`
- Cancel at any time, from any thread - `cancel()`

AsyncTask Example

```
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));
            if (isCancelled()) break;
        }
        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);
```

Multithreading

Accessing Online Content

Bluetooth Communication

Near Field Communication

Google APIs

Bibliography

Connecting to the Network

- Permissions required
  - `ACCESS_NETWORK_STATE` to check the state of the network
  - `INTERNET` to access remote resources over the Internet

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

- Perform all network operations on a separate Thread
  - E.g. `AsyncTask`

Connecting to the Network

- Use `ConnectivityManager` to check network connections
  - `getActiveNetworkInfo()` returns a `NetworkInfo` object
  - `isConnected()` method checks for connectivity

```
public void myClickHandler(View view) {
    [...]
    ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
    if (networkInfo != null && networkInfo.isConnected()) {
        // fetch data
    } else {
        // display error
    }
    [...]
}
```

Retrieving Content

- Using Java sockets
  - Suitable if using a very simple protocol
  - Manage application layer messages yourself
- Using Android `URLConnection`
  - Connect to an URL for reading or writing
  - Automatic handling of different protocols (`file://`, `ftp://`, `http://`, `https://`)
  - Steps:
    - Build an URL object (`new URL("ftp://example.com")`)
    - Calling `URL.openConnection()` returns a `URLConnection`

HTTP Connection

- `AndroidHttpClient` deprecated starting with Android 5.1
- Use `HttpURLConnection` / `HttpsURLConnection`
    - GET operations
    - Transparent support for IPv6

HttpURLConnection

- Call `URL.openConnection()` and cast result to `HttpURLConnection`
- Read data using `getInputStream()`,
- Write data using `getOutputStream()`
- Use `CookieManager` and `HttpCookie` to handle cookies
- Use `setDoOutput(true)` to use the `HTTP POST` method

HttpURLConnection Example

```
URL url = new URL("http://www.android.com/");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
try {
        InputStream in = new BufferedInputStream(urlConnection.getInputStream());
        readStream(in);
} finally {
        urlConnection.disconnect();
}
```

Outline

Multithreading

Accessing Online Content

Bluetooth Communication

Near Field Communication

Google APIs

Bibliography

Bluetooth Operations

- Android can provide an app control over the Bluetooth adapter
  - Turn the adapter on/off
  - Make the device discoverable
  - Scan for discoverable devices
  - Device pairing
  - Transfer data to/from devices
  - Manage multiple connections

Bluetooth Permissions

- `android.permission.BLUETOOTH`
  - Connect to paired devices
  - Transfer data to / from
- `android.permission.BLUETOOTH_ADMIN`
  - Set adapter state (off, on, discoverable)
  - Discover devices
  - Pair with discovered devices with user confirmation
- `android.permission.BLUETOOTH_PRIVILEGED`
  - Pair with devices without user interaction
  - Not available to third-party applications

Bluetooth API

- BluetoothAdapter
  - Local Bluetooth adapter (radio)
  - Obtained using the static method getDefaultAdapter()
  - Entry-point for all operations
    - Discover devices
    - List paired devices
    - Instantiate a BluetoothDevice using a known MAC address
  - isEnabled()
    - Send Intent to enable Bluetooth
  - Create a BluetoothServerSocket

Bluetooth API

- `BluetoothDevice`
  - Represents a remote device
  - `getBondedDevices()` - list of paired devices
  - Query device information (name, address, class, pairing state, etc.)
  - Connect to the remote device by requesting a `BluetoothSocket`

Bluetooth API

- `BluetoothSocket`
  - Similar to a TCP socket
  - Connection point to a remote device
    - `connect()`
  - Exchange data via InputStream or OutputStream
    - `getInputStream()`
    - `getOutputStream()`

Bluetooth API

- `BluetoothServerSocket`
  - Listen for incoming connections (similar to a TCP server socket)
  - Calling the `accept()` method blocks, waiting for incoming connections
  - Return a `BluetoothSocket` when a new connection is accepted

Android BLE API

- Consume less energy
- Making an app available only to devices which support BLE:
  - Entry in the AndroidManifest: `<uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>`
- Checking for BLE feature at runtime:
  - `getPackageManager().hasSystemFeature( PackageManager.FEATURE_BLUETOOTH_LE)`

Android BLE API

- Finding BLE devices:
  - `BluetoothAdapter.startLeScan()`
  - `BluetoothAdapter.LeScanCallback` as parameter
  - Override `onLeScan()` method of `BluetoothAdapter.LeScanCallback`
- RSSI can be used to approximate proximity to sender
- Scan record contains:
  - Device type (unique per manufacturer)
  - Device identifier
  - Attributes

LeScanCallback Example

```
private LeDeviceListAdapter mLeDeviceListAdapter;
...
// Device scan callback.
private BluetoothAdapter.LeScanCallback mLeScanCallback =
        new BluetoothAdapter.LeScanCallback() {
    @Override
    public void onLeScan(final BluetoothDevice device, int rssi,
            byte[] scanRecord) {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                mLeDeviceListAdapter.addDevice(device);
                mLeDeviceListAdapter.notifyDataSetChanged();
            }
        });
    }
};
```

```
mBluetoothAdapter.startLeScan(mLeScanCallback);
...
mBluetoothAdapter.stopLeScan(mLeScanCallback);
```

Outline

Multithreading

Accessing Online Content

Bluetooth Communication

Near Field Communication

Google APIs

Bibliography

Near Field Communication

- Short-range wireless technology (distance 4cm)
- Share small data payloads between an NFC tag and an Android-powered device or two devices
- Data usually kept as NDEF (NFC Data Exchange Format)
- Android NFC devices have 3 modes of operation:
  - Reader/writer mode - read/write passive NFC tags
  - P2P mode - exchange data with another device (E.g. Android Beam)
  - Card emulation mode - device acts like an NFC card (E.g. use phone at an NFC POS terminal)

Using NFC in an Android Application

- Request permission to NFC API:
  - `<uses-permission android:name="android.permission.NFC" />`
- Set minimum SDK to API level 10
  - `<uses-sdk android:minSdkVersion="10"/>`

Using NFC in an Android Application

- Making an app available only to devices which have NFC hardware:
  - Entry in the AndroidManifest: `<uses-feature android:name="android.hardware.nfc" android:required="true" />`
  - At runtime, by checking if `NfcManager.getDefaultAdapter()` returns null

Discovering NFC Devices

- Receive an Intent when an NFC tag is discovered by adding an Intent filter with:
  - Action `android.nfc.action.NDEF_DISCOVERED`
- Check if Intent action is `NfcAdapter.ACTION_NDEF_DISCOVERED`
- Retrieve message from `intent.getParcelableArrayExtra( NfcAdapter.EXTRA_NDEF_MESSAGES)`

NFC Retrieve Message Example

```
@Override
protected void onNewIntent(Intent intent) {
    super.onNewIntent(intent);
    ...
    if (intent != null &&
        NfcAdapter.ACTION_NDEF_DISCOVERED.equals(intent.getAction())){
        Parcelable[] rawMessages =
            intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
        if (rawMessages != null) {
            NdefMessage[] messages = new NdefMessage[rawMessages.length];
            for (int i = 0; i < rawMessages.length; i++) {
                messages[i] = (NdefMessage) rawMessages[i];
            }
            // Process the messages array.
            ...
        }
    }
}
```

Sending NFC Messages

- Have an Activity that implements:
  - `NfcAdapter.CreateNdefMessageCallback`
  - `NfcAdapter.OnNdefPushCompleteCallback`
- In `onCreate()` get an instance of the `NfcAdapter`
- Set the Activity as responsible for handling the adapter's relevant callbacks:
  - `NfcAdapter.setNdefPushMessageCallback()`
  - `NfcAdapter.setOnNdefPushCompleteCallback()`

Sending NFC Messages

- Override `createNdefMessage()` callback
  - Will be called by the system when a new NFC tag is discovered
  - Create the actual message
- Use `onNdefPushComplete()` callback - notify the UI of the message being sent

NFC Send Message Example

```
public class Beam extends Activity implements CreateNdefMessageCallback {
    NfcAdapter mNfcAdapter;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        [...]
        mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
        if (mNfcAdapter == null) {
            finish();
            return;
        }
        mNfcAdapter.setNdefPushMessageCallback(this, this);
    }

    @Override
    public NdefMessage createNdefMessage(NfcEvent event) {
        String text = ("Beam me up, Android!\n\n" +
                "Beam Time: " + System.currentTimeMillis());
        NdefMessage msg = new NdefMessage(
                new NdefRecord[] { createMime(
                        "application/vnd.com.example.android.beam", text.getBytes())
        });
        return msg;
    }
[...]
```

Outline

Multithreading

Accessing Online Content

Bluetooth Communication

Near Field Communication

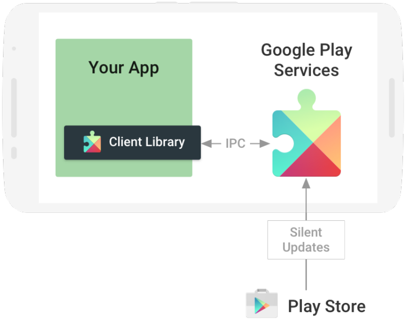
Google APIs

Bibliography

- Google-related services are not available within the AOSP
- Google Play Services APK and other proprietary libraries
- Provide access to a series of services:
  - Google Maps
  - Google+
  - Google Drive
  - Google Cloud Messaging (GCM)
  - etc.
- Through Google Play
- Provided as library projects within the SDK

Google Play Services

- The client library (included in apps) relies on the Play Services APK for communicating with external services
- The Play Services APK is updated directly through the Play Store



Source: http://developer.android.com

Google Cloud Messaging (GCM)

- New version: Firebase Cloud Messaging (FCM)
- Free service for sending messages between client apps and servers
- Two types of messages:
  - Downstream messages - server to client (push notifications)
  - Upstream messages - client to server

Android GCM client

- Add required permissions to the AndroidManifest:
  - `android.permission.INTERNET`
  - `com.google.android.c2dm.permission.RECEIVE`
  - `applicationPackage + ".permission.C2D_MESSAGE"` - prevent other apps from receiving your messages
- Set minimum SDK to API level 8
- Declare a `com.google.android.gms.gcm.GcmReceiver` broadcast receiver
  - Sender needs to have `com.google.android.c2dm.permission.SEND`

- Add a service that extends `GcmListenerService`
  - `onMessageReceived()` called when receiving downstream messages
- Extend InstanceIDListenerService - handle registration tokens
  - Obtain a registration token using the `InstanceID` API
- Send messages using `GoogleCloudMessaging.send()`

Google Maps Android API

- Add maps based on Google Maps to a third-party application
- API automatically handles:
  - Access to Google Maps servers
  - Map data download
  - Map display
  - User interaction with map
- Allows adding custom data to a map:
  - Markers
  - Polylines or polygons
  - Overlays

Using Maps within an Application

- Google Maps API key required
  - Register app to Google API Console
  - Add key to the AndroidManifest as a `meta-data` component
- Add the required permissions to the AndroidManifest:
  - `android.permission.INTERNET` - download map tiles from Google servers
  - `android.permission.ACCESS_NETWORK_STATE` - check connection status to see if data can be downloaded

Using Maps within an Application

- Add the required permissions to the AndroidManifest:
  - `android.permission.WRITE_EXTERNAL_STORAGE` - cache map tile data on phone external storage
  - `android.permission.ACCESS_COARSE_LOCATION` (recommended) - use WiFi and / or mobile data to determine the device's location
  - `android.permission.ACCESS_FINE_LOCATION` (recommended) - determine a precise location using GPS, WiFi and / or mobile data

Using Maps within an Application

- Add a fragment to the layout of the Activity that will show the map
  - Set the fragment's `android:name` attribute to `com.google.android.gms.maps.MapFragment`
- In the Activity, get an instance to the fragment, and cast it to `MapFragment` class
- Render the map:
  - Implement `OnMapReadyCallback` interface
  - Call `MapFragment.getMapAsync(OnMapReadyCallback)`

Outline

Multithreading

Accessing Online Content

Bluetooth Communication

Near Field Communication

Google APIs

Bibliography

Bibliography

- https://developer.android.com/guide/components/processes-and-threads.html
- http://developer.android.com/training/multiple-threads/index.html
- http://developer.android.com/training/basics/network-ops/connecting.html
- http://developer.android.com/reference/android/os/AsyncTask.html
- https://developer.android.com/training/multiple-threads/communicate-ui.html
- http://developer.android.com/guide/topics/connectivity/bluetooth.html
- http://developer.android.com/guide/topics/connectivity/bluetooth-le.html
- http://developer.android.com/guide/topics/connectivity/nfc/index.html
- https://developers.google.com/maps/documentation/android/
- https://developers.google.com/cloud-messaging/android/client

Keywords

- Threads
- AsyncTask
- URLConnection
- Bluetooth
- Bluetooth Low Energy
- NFC
- Google Play Services
- GCM
- Google Maps