Android Connectivity

Lecture 4

Security of Mobile Devices

2019

Outline

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`

AsyncTask

- 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);
```

Outline

Sources of Location

- GPS
  - Most accurate
  - Works only outdoors
  - Energy consuming
  - Responds slower
- WiFi/mobile
  - Less accurate
  - Works both indoors and outdoors
  - Less energy consuming
  - Respons faster
- Use both

Request Location Permissions

- For GPS_PROVIDER:
  - `ACCESS_FINE_LOCATION`
  - `android.hardware.location.gps` hardware feature
- For NETWORK_PROVIDER (WiFi/mobile):
  - `ACCESS_COARSE_LOCATION`
  - `android.hardware.location.network` hardware feature
- When using both:
  - Request only `ACCESS_FINE_LOCATION`
  - Request both hardware features

Request Location Permissions - Examples

```
<manifest ... >
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    ...
    <!-- Needed only if your app targets Android 5.0 (API level 21) or higher. -->
    <uses-feature android:name="android.hardware.location.gps" />
    ...
</manifest>
```

```
<manifest ... >
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    ...
    <!-- Needed only if your app targets Android 5.0 (API level 21) or higher. -->
    <uses-feature android:name="android.hardware.location.network" />
    ...
</manifest>
```

Listen to Location Updates

- Receive location updates from LocationManager
  - Get a reference to the LocationManager (system service)
- Using a LocationListener
  - Implement a LocationListener with callbacks
  - Callbacks will be called by the LocationManager
  - Register listener with LocationManager to receive updates

SMD

Listen to Location Updates

```
LocationManager locationManager = (LocationManager) this.getSystemService(
                                                        Context.LOCATION_SERVICE);
[..]
LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
      makeUseOfNewLocation(location);
    }

    public void onStatusChanged(String provider, int status, Bundle extras) {}

    public void onProviderEnabled(String provider) {}

    public void onProviderDisabled(String provider) {}
  };
[..]
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
                                                        0, 0, locationListener);
```

Operations with LocationManager

Specify location provider

```
String locationProvider = LocationManager.NETWORK_PROVIDER;
// Or, use GPS location data:
// String locationProvider = LocationManager.GPS_PROVIDER;

locationManager.requestLocationUpdates(locationProvider, 0, 0, locationListener);
```

Obtain last known location (cached)

```
String locationProvider = LocationManager.NETWORK_PROVIDER;
// Or use LocationManager.GPS_PROVIDER

Location lastKnownLocation = locationManager.getLastKnownLocation(locationProvider);
```

Stop listening for updates

```
// Remove the listener you previously added
locationManager.removeUpdates(locationListener);
```

Outline

WifiManager

- WiFi connectivity service
- Configure, manage and monitor WiFi connections
- Scan for available networks
- Needed permissions:
  - `ACCESS_WIFI_STATE`
  - `CHANGE_WIFI_STATE`

Enable and Disable WiFi

- Obtain a reference to WifiManager
- Call `setWifiEnabled` to enable or disable WiFi

```
WifiManager wifiManager = (WifiManager) this.getSystemService(Context.WIFI_SERVICE);
[..]
wifiManager.setWifiEnabled(true);
[..]
wifiManager.setWifiEnabled(false);
```

Scan WiFi Networks

- Obtain a reference to WiFiManager
- Implement a broadcast receiver to obtain scanning results
  - Request scanning results from WiFiManager when receiver is called
- Register this receiver for action `WifiManager.SCAN_RESULTS_AVAILABLE_ACTION`
- Start scanning

Scan WiFi Networks

```
WifiManager wifiManager = (WifiManager) this.getSystemService(Context.WIFI_SERVICE);
[..]
class WifiScanReceiver extends BroadcastReceiver {
   public void onReceive(Context c, Intent intent) {
        List<ScanResult> wifiScanList = wifiManager.getScanResults();
        String data = wifiScanList.get(0).toString();
   }
}
[..]
WifiScanReceiver wifiReceiver = new WifiScanReceiver();
registerReceiver(wifiReceiver, new IntentFilter(
                                    WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));
wifiManager.startScan();
```

SMD

Current Connection Information

- Obtain WifiInfo object from WifiManager
  - Information about the active WiFi connection
- Obtain DhcpInfo object from WifiManager
  - IP, mask, gateway, DNS servers

```
WifiManager wifiManager = (WifiManager) this.getSystemService(Context.WIFI_SERVICE);

WifiInfo wifiInfo = wifiManager.getConnectionInfo();
Log.v(LOG_TAG, "SSID: " + wifiInfo.getSSID() + ", "
               "Frequency: " + wifiInfo.getFrequency() + ", "
               "Link sped: " + wifiInfo.getLinkSpeed());


DhcpInfo dhcpInfo = wifiManager.getDhcpInfo();
Log.v(LOG_TAG, "DHCP Info: "+dhcpInfo.toString());
```

Outline

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()` of BluetoothAdapter
    - List of paired devices
    - BluetoothDevice objects
  - Query device information (name, address, class, pairing state)
  - Connect to the remote device by requesting a `BluetoothSocket`

BluetootAdapter and BluetoothDevice Example

```
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) {
    // Device doesn't support Bluetooth
}
[..]
if (!mBluetoothAdapter.isEnabled()) {
    Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
[..]
Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
if (pairedDevices.size() > 0) {
    for (BluetoothDevice device : pairedDevices) {
        String deviceName = device.getName();
        String deviceHardwareAddress = device.getAddress(); // MAC address
    }
}
```

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

Bluetooth API

- `BluetoothServerSocket`
  - Obtained from BluetoothAdapter
  - 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
  - Implement `BluetoothAdapter.LeScanCallback`
  - Override `onLeScan()` method of `BluetoothAdapter.LeScanCallback`
- Scan record contains:
  - RSSI - approximate proximity to sender
  - Device type (unique per manufacturer)
  - Device identifier
  - Attributes

LeScanCallback Example

```java
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();
            }
        });
    }
};
```

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

Outline

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)

NFC modes of operation

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

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

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
- https://developer.android.com/guide/topics/location/strategies.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/

Keywords

- Threads
- AsyncTask
- LocationManager
- WiFiManager
- Bluetooth
- Bluetooth Low Energy
- NFC