Network Connection

CS443 – Mobile Applications
Instructor: Bo Sheng

Outline

• Connecting to the Internet
• Perform network operations
• Manage network usage
• Parsing data

Outline

• Connecting to the Internet
• Perform network operations
• Manage network usage
• Parsing data

Connect to the Internet

• Permissions in the manifest file
  - <uses-permission android:name="android.permission.INTERNET"/>
  - <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>

• Create an HTTP client
  - Common approach for Android apps to send and receive data
  - HttpURLConnection: support HTTPS, streaming data, configurable timeouts, IPV6, ...

  URL url = new URL("http://www.android.com/");
  HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();

Connect to the Internet

• Check the network connection
  - Before your app attempts to connect to the network
  - Using isConnected() and getActiveNetworkInfo()

  ConnectivityManager connMgr = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
  NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
  if (networkInfo != null && networkInfo.isConnected()) {
    // fetch data
  } else {
    // display error
  }

Perform Network Operations

• Start a separate thread
  - Network operations can involve unpredictable delays
  - E.g., using AsyncTask
    - doInBackground: network connection and operation
    - onPostExecute: update the UI

1. The app parses the specified URL to the AsyncTask
2. The doInBackground method doInBackground() calls the downloadFile() method.
3. The downloadFile() method takes a URL string as a parameter and uses it to create a URL object
4. The URL object is used to establish an HttpURLConnection
5. The HttpURLConnection object fetches the web page content as an InputStream
6. The InputStream is passed to the read() method, which converts the stream to a string
7. Finally, the onPostExecute() method displays the string in the main activity's UI.

Perform Network Operations

• Use HttpURLConnection
  - Obtain a new HttpURLConnection by calling URL.openConnection()
  - Prepare the request: set up the parameters of the HttpURLConnection, e.g.,
    - Timeout parameters
      - conn.setReadTimeout(10000 /* milliseconds */);
      - conn.setConnectTimeout(15000 /* milliseconds */);
    - Request method: GET, POST, PUT, ...

  HttpURLConnection conn = (HttpURLConnection) url.openConnection();
  conn.setRequestMethod("GET");
  if (networkInfo != null && networkInfo.isConnected()) {
    InputStream inputStream = conn.getInputStream();
Perform Network Operations

- Use HttpURLConnection
  - Connection is initiated when HttpURLConnection.connect() is called
  - Apps are not always required to explicitly call connect method
  - Some operations such as getInputStream will implicitly perform the connection when necessary.

```java
URL url = new URL("http://www.android.com/");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
ConnectionManager connMgr = (ConnectionManager)
MemoryManager.getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
if (networkInfo != null && networkInfo.isConnected()) {
    urlConnection.setReadTimeout(10000 /* milliseconds */);
    urlConnection.setConnectTimeout(15000 /* milliseconds */);
    urlConnection.connect();
    InputStream in = new BufferedInputStream(urlConnection.getInputStream());
    readStream(in);
} else {
    // display error
}
```

- Response header from the server: content type/length, encoding
  - An int representing the three digit HTTP Status-Code: 1xx: Information, 2xx: Success, 3xx: Redirection, 4xx: Client Error, 5xx: Server Error

```
InputStream is = new BufferedInputStream(urlConnection.getInputStream());
readStream(is);
```

- Disconnect: Once the response body has been read, the HttpURLConnection should be closed by calling disconnect(). Disconnecting releases the resources held by a connection so they may be closed or reused.

```
urlConnection.setDoOutput(true);
OutputStream out = new BufferedOutputStream(urlConnection.getOutputStream());
writeStream(out);
InputStream in = new BufferedInputStream(urlConnection.getInputStream());
readStream(in);
```

Manage Network Usage

- Fine-grained control of network resources
  - User specific settings, such as
    - How often your app syncs data?
    - Perform network operations only when on WiFi?
    - Transfer data while roaming?
Manage Network Usage

- **Network connection**
  - Types of network connection (WiFi or cellular)
  - Check the network state before performing network operations
  - ConnectivityManager and NetworkInfo
    - ConnectivityManager: Answers queries about the state of network connectivity. Notify applications when network connectivity changes
      - Monitor network connections (Wi-Fi, GPRS, UMTS, etc.)
      - Send broadcast intents when network connectivity changes
      - Attempt to "fail over" to another network when connectivity to a network is lost
      - Provide an API that allows applications to query the coarse-grained or fine-grained state of the available networks
      - Provide an API that allows applications to request and select networks for their data traffic
  - NetworkInfo: Describes the status of a network interface of a given type (currently either Mobile or Wi-Fi)

```java
ConnectivityManager connMgr = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
boolean isWifiConn = networkInfo.isConnected();
Log.d(DEBUG_TAG, "Wifi connected: " + isWifiConn);
Log.d(DEBUG_TAG, "Mobile connected: " + isMobileConn);
```

- **Preference of network usage**
  - For example: allow users to upload videos only when the device is connected to a Wi-Fi network; sync (or not) depending on specific criteria such as network availability, time interval, ...
  - Required permissions
    - android.permission.INTERNET — Allows applications to open network socket
    - android.permission.ACCESS_NETWORK_STATE — Allows applications to access information about networks

```java
public boolean isOnline() {
    ConnectivityManager connMgr = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
    return (networkInfo != null && networkInfo.isConnected());
}
```
Manage Network Usage

- Settings
  - A Preference object represents a single setting
  - Key-value pairs are stored in a default
    SharedPreferences file
  - PreferenceActivity or PreferenceFragment
  - Different types of preferences
    - CheckBoxPreference
    - ListPreference
    - EditTextPreference
- Define Preferences in XML

Manage Network Usage

- Preference of network usage
  - Implement a preferences activity
  - A subclass of PreferenceActivity
  - Implement OnSharedPreferenceChangeListener

@Override
public void onSharedPreferenceChanged(...)
{
// Sets refreshDisplay to true so that when the user returns to the main
// activity, the display refreshes to reflect the new settings.
NetworkActivity.refreshDisplay = true;
}

HW2

Thread 1

while(true){
  sleep(3000);
  if(count<4){
    randomly pick a tile to place "X";
    count++;
    updateUI;
  }
}

Thread 2

while(true){
  move to the next tile;
  updateUI();
  if passed a "X":
    clear the tile;
    count--;
  pause;
}

Manage Network Usage

- Preference of network usage
  - Respond to preference changes
  - Get the current network state

public void updateConnectedFlags() {
  ConnectivityManager connMgr = (ConnectivityManager)
    getSystemService(Context.CONNECTIVITY_SERVICE);
  NetworkInfo activeInfo = connMgr.getActiveNetworkInfo();
  if (activeInfo != null && activeInfo.isConnected()) {
    wifiConnected = activeInfo.getType() == ConnectivityManager.TYPE_WIFI;
    mobileConnected = activeInfo.getType() == ConnectivityManager.TYPE_MOBILE;
  } else {
    wifiConnected = false;
    mobileConnected = false;
  }
}

Manage Network Usage

- Preference of network usage
  - Respond to preference changes
  - Check the network preference

public void loadPage() {
  if (((sPref.equals(ANY)) && (wifiConnected || mobileConnected)) ||
      ((sPref.equals(WIFI)) && (wifiConnected))) {
    // AsyncTask subclass
    new DownloadXmlTask().execute(URL);
  } else {
    showErrorPage();
  }
}
Manage Network Usage

- Parsing data (XML)
  - Choose a parser (XmlPullParser)
  - via `Xml.newPullParser`
  - Analyze the feed

```java
public List parse(InputStream in) throws XmlPullParserException, IOException {
    try {
        XmlPullParser parser = Xml.newPullParser();
        parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, false);
        parser.setInput(in, null);
        parser.nextTag();
        return readFeed(parser);
    } finally {
        in.close();
    }
}
```

```java
private List readFeed(XmlPullParser parser) throws XmlPullParserException, IOException {
    List entries = new ArrayList();
    parser.require(XmlPullParser.START_TAG, ns, "feed");
    while (parser.next() != XmlPullParser.END_TAG) {
        if (parser.getEventType() != XmlPullParser.START_TAG) {
            continue;
        }
        String name = parser.getName();
        if (name.equals("entry")) {
            entries.add(readEntry(parser));
        } else {
            skip(parser);
        }
    }
    return entries;
}
```

```java
private Entry readEntry(XmlPullParser parser) throws XmlPullParserException, IOException {
    parser.require(XmlPullParser.START_TAG, ns, "entry");
    String title = null;
    String summary = null;
    String link = null;
    while (parser.next() != XmlPullParser.END_TAG) {
        if (parser.getEventType() != XmlPullParser.START_TAG) {
            continue;
        }
        String name = parser.getName();
        if (name.equals("title")) {
            title = readTitle(parser);
        } else if (name.equals("summary")) {
            summary = readSummary(parser);
        } else if (name.equals("link")) {
            link = readLink(parser);
        } else {
            skip(parser);
        }
    }
    return new Entry(title, summary, link);
}
```

- Parse each entry
  - A "read" method for each tag you're interested in. For example, `readEntry()`, `readTitle()`, and so on.

HW3

1. Read a user input (city's name or zip code)
2. Display the temperature
3. Display the weather icon
4. Show the map of the city

• **openweathermap.org**
  - References
    - [https://openweathermap.org/current](https://openweathermap.org/current)
    - [http://openweathermap.org/weather-conditions](http://openweathermap.org/weather-conditions)

- Query examples
  - [http://api.openweathermap.org/data/2.5/weather?q=boston,us&APPID=cc4ae6e545dee0a295a471824c9fdbda](http://api.openweathermap.org/data/2.5/weather?q=boston,us&APPID=cc4ae6e545dee0a295a471824c9fdbda)
  - [http://api.openweathermap.org/data/2.5/weather?zip=02125,us&APPID=cc4ae6e545dee0a295a471824c9fdbda](http://api.openweathermap.org/data/2.5/weather?zip=02125,us&APPID=cc4ae6e545dee0a295a471824c9fdbda)
HW3

• Main steps
  – Get the user input and form the http URL
  – Then, download the data from the server
  – Analyze/parse the returned data
    • Temperature
    • Icon file name
    • Latitude, longitude
  – Download the icon file

• Parse JSON data
  
  ```
  JSONObject jobj = new JSONObject(StringInput);
  jobj.getString("base").getJSONObject("coord");
  
  ```

Manage Network Usage

• Settings
  
  ```
  public static class SettingsFragment extends PreferenceFragment {
    @Override
    public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // Load the preferences from an XML resource
      addPreferencesFromResource(R.xml.preferences);
    }
    ...
  }
  public class SettingsActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // Display the fragment as the main content.
      getFragmentManager().beginTransaction().replace(android.R.id.content, new SettingsFragment()).commit();
    }
  }
  ```