Android Networking

Phong Nguyen Quoc

Google

Introduction

- Your role
- Your background and experience in the subject:
  - Java.
  - Basic Android.
- What do you want from this course

Course Objectives

- At the end of the course, you will have acquired sufficient knowledge to:
  - **Connecting to the Network.**
  - **Managing Network Usage.**
  - **Parsing XML Data.**
  - **Parsing JSON Data.**

Agenda

I. Connecting to the Network
II. Managing Network Usage
III. Parsing XML Data
IV. Parsing JSON Data
V. Assignment

Assessment Disciplines

- Class Participation: 40%
- Assignment: 60%
- Final Exam: 0%
- Passing Scores: 70%

Set Up Environment

- To complete the course, your PC must install:
  - Eclipse with Android plugins
  - Android SDK

Course Administration

- In order to complete the course you must:
  - Sign in the Class Attendance List
  - Participate in the course
  - Provide your feedback in the End of Course Evaluation

Connecting to the Network

Connecting to the Network

- Choose a HTTP client.
- Check the Network Connection.
- Perform Network Operations on a Separate Thread.
- Connect and Download Data.
- Convert the InputStream to a String.
- Convert the InputStream to an Image.

Connecting to the Network (cont'd)

- Choose a HTTP client:
  - HttpURLConnection
  - Apache HTTP Client

Connecting to the Network (cont'd)

- HttpURLConnection:
  - Simplifies connections to HTTP servers
    - Same as with desktop Java programming

Connecting to the Network (cont'd)

- HttpURLConnection: Reading data from a URL
  - **Getting a connection from a URL**
    - **URL url = new URL("http://…")**;
    - **HttpURLConnection urlConnection = (HttpURLConnection)url.openConnection()**;
  - **Reading data**
    - **BufferedReader in = new BufferedReader(new InputStreamReader (urlConnection.getInputStream()))**;
    - **while ((line = in.readLine()) != null) {**

      // doSomethingWith(line);

      }
  - **Other methods**
    - **disconnect, getResponseCode, getHeaderField**
      - Call disconnect when done

Connecting to the Network (cont'd)

- Apache HTTP Client:
  - Simplest way to read an entire URL (via GET) into String
  - Moderately simple way to send POST parameters, then read entire result into a String

Connecting to the Network (cont'd)

- Apache HTTP Client: Reading Result of GET Request
  - Make a default client
    - **HttpClient client = new DefaultHttpClient();**
  - Make an HttpGet with address
    - **HttpGet httpGet = new HttpGet(address);**
  - Make a String ResponseHandler
    - **ResponseHandler<String> handler = new BasicResponseHandler();**
  - Call client.execute
    - **String content = client.execute(httpGet, handler);**

Connecting to the Network (cont'd)

- Apache HTTP Client: Reading Result of POST Request
  - **Make a default client** (same as GET Request)
  - **Make an HttpPost with address**
    - **HttpPost httpPost = new HttpPost(address);**
  - **Make a List of name/value pairs**
    - **List<NameValuePair> params = new ArrayList<NameValuePair>()**;
    - **params.add(new BasicNameValuePair(paramName1, paramValue1))**;
    - **params.add(…)**; *// More names and values. NOT URL-encoded*
  - **Attach POST data**
    - **UrlEncodedFormEntity entity = new UrlEncodedFormEntity(params, "UTF-8")**;
    - **httpPost.setEntity(entity)**;
  - **Make a String ResponseHandler**
    - **ResponseHandler<String> handler = new BasicResponseHandler()**;
  - **Call client.execute**
    - **String content = client.execute(httpPost, handler)**;

Connecting to the Network (cont'd)

- Check the Network Connection:
  - Before your app attempts to connect to the network, it should check to see whether a network connection is available using **getActiveNetworkInfo()** and **isConnected()**.
  - Example:

```
public boolean isConnectingToInternet(){
    ConnectivityManager connectivity =
            (ConnectivityManager)getSystemService(Context.CONNECTIVITY_SERVICE);
      if (connectivity != null)
      {
          NetworkInfo activeNetwork = connectivity.getActiveNetworkInfo();
          if (activeNetwork != null && activeNetwork.isConnected())
              return true;
      }
      return false;
}
```

Connecting to the Network (cont'd)

- Perform Network Operations on a Separate Thread:
  - Always perform network operations on a separate thread from the UI.
  - The AsyncTask class provides one of the simplest ways to fire off a new task from the UI thread.
  - Example:

```
private class DownloadTask extends AsyncTask<String, Void, String>  {

    @Override
    protected String doInBackground(String... params) {
        // Connect to Server and download data
        return downloadFromURL(params[0]);
    }

    @Override
    protected void onPostExecute(String result) {
        // Update UI
    }
}
```

Connecting to the Network (cont'd)

- Connect and Download Data:
  - Example:

```
private String downloadFromURL(String url) {
    HttpClient httpClient = new DefaultHttpClient();
    HttpPost httpPost = new HttpPost(url);

    String result = null;
    InputStream is = null;

    try {
        HttpResponse httpResponse = httpClient.execute(httpPost);
        HttpEntity httpEntity = httpResponse.getEntity();
        is = httpEntity.getContent();
        result = convertStreamToString(is);
    }
    catch (ClientProtocolException e) { }
    catch (IOException e) { }
    finally {
        try {
            if (is != null)
                is.close();
        }
        catch (IOException e) { }
    }

    return result;
}
```

Connecting to the Network (cont'd)

- Convert the InputStream to a String:
  - Example:

```
private String convertStreamToString(InputStream is) {
    BufferedReader reader = new BufferedReader(new InputStreamReader(is));
    StringBuilder sb = new StringBuilder();

    String line = null;
    try {
        while ((line = reader.readLine()) != null) {
            sb.append(line);
        }
    }
    catch (IOException e) { }
    finally {
        try {
            is.close();
        }
        catch (IOException e) { }
    }
    return sb.toString();
}
```

Connecting to the Network (cont'd)

- Convert the InputStream to an Image:
  - Example:

```
private String storeDownloadedImage(InputStream is, String fileId) {
    String tmpFilePath = null;
    try {
        // Getting Caching directory
        File cacheDirectory = getCacheDir();
        // Temporary file to store the downloaded image
        File tmpFile = new File(cacheDirectory.getPath() + "/csc_" + fileId + ".jpg");

        FileOutputStream fOutStream = new FileOutputStream(tmpFile);

        // Creating a bitmap from the downloaded inputstream
        Bitmap b = BitmapFactory.decodeStream(is);
        // Writing the bitmap to the temporary file as png file
        b.compress(Bitmap.CompressFormat.JPEG, 50, fOutStream);

        fOutStream.flush();
        fOutStream.close();

        tmpFilePath = tmpFile.getPath();
    }
    catch (FileNotFoundException e) { }
    catch (IOException e) { }
    finally {
        try {
            is.close();
        }
        catch (IOException e) { }
    }

    return tmpFilePath;
}
```

Managing Network Usage

Managing Network Usage

- Check a Device's Network Connection
- Manage Network Usage
- Implement a Preferences Activity
- Respond to Preference Changes
- Detect Connection Changes

Managing Network Usage (cont'd)

- Check a Device's Network Connection:
  - Wi-Fi is typically faster. Also, mobile data is often metered, which can get expensive.
  - Only fetch large data if a Wi-Fi network is available.
  - Example:

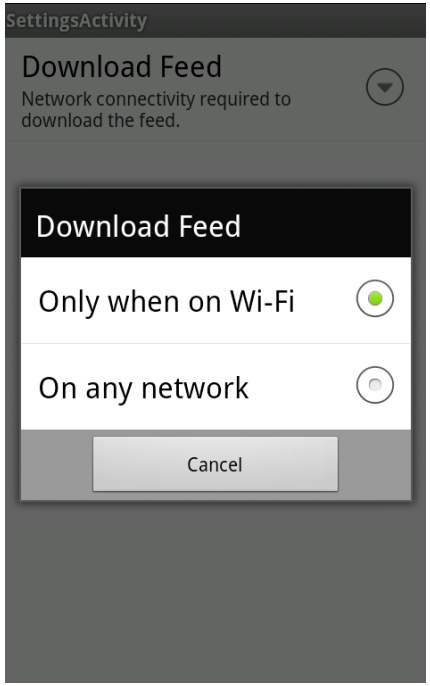
```
public boolean isWifiOn(){
    ConnectivityManager connectivity =
            (ConnectivityManager)getSystemService(Context.CONNECTIVITY_SERVICE);
      if (connectivity != null)
      {
          NetworkInfo wifiNetwork = connectivity.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
          if (wifiNetwork != null && wifiNetwork.isConnected())
              return true;
      }
      return false;
}
```

Managing Network Usage (cont'd)

- Manage Network Usage:
  - To write an app that supports network access and managing network usage, your manifest must have the right permissions and intent filters.
  - The manifest excerpted below includes the following permissions:
    - **android.permission.INTERNET** - Allows applications to open network sockets.
    - **android.permission.ACCESS_NETWORK_STATE** - Allows applications to access information about networks.

Managing Network Usage (cont'd)

- Implement a Preferences Activity:
  - SettingsActivity has an intent filter for the **ACTION_MANAGE_NETWORK_USAGE** action.
  - SettingsActivity is a subclass of **PreferenceActivity**.
  - SettingsActivity implements **OnSharedPreferenceChangeListener**




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

<application ... >

    ...

    <activity
        android:name="com.csc.demo.networking.SettingsActivity"
        android:label="SettingsActivity" >
        <intent-filter>
            <action android:name="android.intent.action.MANAGE_NETWORK_USAGE" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </activity>
</application>
```

Managing Network Usage (cont'd)

- Implement a Preferences Activity: (cont'd)
  - Example:

```
import android.content.SharedPreferences;

public class SettingsActivity extends PreferenceActivity
                                        implements OnSharedPreferenceChangeListener {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.preferences);
    }

    @Override
    protected void onResume() {
        super.onResume();
        // Registers a callback to be invoked whenever a user changes a
        // preference.
        getPreferenceScreen().getSharedPreferences()
                                        .registerOnSharedPreferenceChangeListener(this);
    }

    @Override
    public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String key) {
        // Refresh UI
    }

}
```

Managing Network Usage (cont'd)

- Detect Connection Changes:
  - NetworkReceiver is a subclass of BroadcastReceiver
  - NetworkReceiver intercepts the action **CONNECTIVITY_ACTION**, determines what the network connection status is, and sets the flags **wifiConnected** and **mobileConnected** to true/false accordingly.

Managing Network Usage (cont'd)

- Detect Connection Changes: (cont'd)
  - Example:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Register BroadcastReceiver to track connection changes.
    IntentFilter filter = new IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION);
    receiver = new NetworkReceiver();
    this.registerReceiver(receiver, filter);
}

@Override
public void onStart() {
    super.onStart();
    SharedPreferences sharedPrefs = PreferenceManager.getDefaultSharedPreferences(this);
    sPref = sharedPrefs.getString("listPref", "Wi-Fi");

    if (refreshDisplay) { /* Reload UI */ }
}

public class NetworkReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        ConnectivityManager connMgr =
                (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();

        // Checks the user prefs and the network connection. Based on the result,
        // decides whether to refresh the display or keep the current display.
        if (WIFI.equals(sPref) && networkInfo != null
                && networkInfo.getType() == ConnectivityManager.TYPE_WIFI) {
            refreshDisplay = true;
        }
        else if (ANY.equals(sPref) && networkInfo != null) {
            refreshDisplay = true;
        }
        else {
            refreshDisplay = false;
        }
    }
}
```

Parsing XML Data

Parsing XML Data

- Choose a Parser
- Analyze the Feed
- Instantiate the Parser
- Read the Feed
- Parse XML

Parsing XML Data (cont'd)

- Choose a Parser:
  - Recommend **XmlPullParser**

Parsing XML Data (cont'd)

- Analyze the Feed:
  - Example:

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:creativeCommons="http://backend.userland.com/creativeCommonsRssModule"
      ...>
<title type="text">newest questions tagged android - Stack Overflow</title>
    ...
    <entry>
    ...
    </entry>
    <entry>
        <id>http://stackoverflow.com/q/9439999</id>
        <re:rank scheme="http://stackoverflow.com">0</re:rank>
        <title type="text">Where is my data file?</title>
        <category term="android"
            scheme="http://stackoverflow.com/feeds/tag?tagnames=android&sort=newest/tags" />
        <category term="file"
            scheme="http://stackoverflow.com/feeds/tag?tagnames=android&sort=newest/tags" />
        <author>
            <name>cliff2310</name>
            <uri>http://stackoverflow.com/users/1128925</uri>
        </author>
        <link rel="alternate"
            href="http://stackoverflow.com/questions/9439999/where-is-my-data-file" />
        <published>2012-02-25T00:30:54Z</published>
        <updated>2012-02-25T00:30:54Z</updated>
        <summary type="html">
        <p>I have an Application that requires a data file...</p>
        </summary>
    </entry>
    <entry>
    ...
    </entry>
    ...
</feed>
```

Parsing XML Data (cont'd)

- Instantiate the Parser:
  - Call to **nextTag()** and invokes the **readFeed()** method, which extracts and processes the data the app is interested in.
  - Example:

```
// This class represents a single entry (post) in the XML feed.
// It includes the data members "title," "link," and "summary."
public static class Entry {
    public final String title;
    public final String link;
    public final String summary;

    private Entry(String title, String summary, String link) {
        this.title = title;
        this.summary = summary;
        this.link = link;
    }
}

public List<Entry> 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();
        }
}
```

Parsing XML Data (cont'd)

- Read the Feed:
  - The **readFeed()** method looks for elements tagged "entry" as a starting point for recursively processing the feed. If a tag isn't an entry tag, it skips it.
  - Example:

```
private List<Entry> readFeed(XmlPullParser parser)
                                        throws XmlPullParserException, IOException {
    List<Entry> entries = new ArrayList<Entry>();

    parser.require(XmlPullParser.START_TAG, null, "feed");
    while (parser.next() != XmlPullParser.END_TAG) {
        if (parser.getEventType() != XmlPullParser.START_TAG) {
            continue;
        }

        String name = parser.getName();
        // Starts by looking for the entry tag
        if (name.equals("entry"))
            entries.add(readEntry(parser));
        else
            skip(parser);
    }
    return entries;
}
```

Parsing XML Data (cont'd)

- Parse XML:
  - Identify the tags you want to include in your app.
  - Create the following methods:
    - A **"read"** method for each tag you're interested in. For example, **readEntry()**, **readTitle()**, and so on.
    - Methods to extract data for each different type of tag and to advance the parser to the next tag. For example:
      - For the title, the parser calls **readText()**. This method extracts data for these tags by calling **parser.getText()**.
      - For the link tag, the parser extracts data for links by first determining if the link is the kind it's interested in. Then it uses **parser.getAttributeValue()** to extract the link's value.
      - For the entry tag, the parser calls **readEntry()**. This method parses the entry's nested tags and returns an Entry object with the data members title, link,...
    - Use **skip()** method to skip tags it's not interested in.

Parsing XML Data (cont'd)

- Parse XML: (cont'd)
  - Example:

```
// Parses the contents of an entry.
private Entry readEntry(XmlPullParser parser) throws XmlPullParserException, IOException {
    parser.require(XmlPullParser.START_TAG, null, "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);
}

// Processes title tags in the feed.
private String readTitle(XmlPullParser parser) throws IOException, XmlPullParserException {
    parser.require(XmlPullParser.START_TAG, null, "title");
    String title = readText(parser);
    parser.require(XmlPullParser.END_TAG, null, "title");
    return title;
}
```

Parsing XML Data (cont'd)

- Parse XML: (cont'd)
  - Example:

```
// Processes link tags in the feed.
private String readLink(XmlPullParser parser) throws IOException, XmlPullParserException {
    String link = "";
    parser.require(XmlPullParser.START_TAG, null, "link");
    String tag = parser.getName();
    String relType = parser.getAttributeValue(null, "rel");
    if (tag.equals("link")) {
        if (relType.equals("alternate")) {
            link = parser.getAttributeValue(null, "href");
            parser.nextTag();
        }
    }
    parser.require(XmlPullParser.END_TAG, null, "link");
    return link;
}

// Processes summary tags in the feed.
private String readSummary(XmlPullParser parser) throws IOException, XmlPullParserException {
    parser.require(XmlPullParser.START_TAG, null, "summary");
    String summary = readText(parser);
    parser.require(XmlPullParser.END_TAG, null, "summary");
    return summary;
}

// For the tags title and summary, extracts their text values.
private String readText(XmlPullParser parser) throws IOException, XmlPullParserException {
    String result = "";
    if (parser.next() == XmlPullParser.TEXT) {
        result = parser.getText();
        parser.nextTag();
    }
    return result;
}
```

Parsing XML Data (cont'd)

- Parse XML: (cont'd)
  - Example:

```
// Skips tags the parser isn't interested in. Uses depth to handle nested tags. i.e.,
// if the next tag after a START_TAG isn't a matching END_TAG, it keeps going until it
// finds the matching END_TAG (as indicated by the value of "depth" being 0).
private void skip(XmlPullParser parser) throws XmlPullParserException, IOException {
    if (parser.getEventType() != XmlPullParser.START_TAG) {
        throw new IllegalStateException();
    }
    int depth = 1;
    while (depth != 0) {
        switch (parser.next()) {
        case XmlPullParser.END_TAG:
                depth--;
                break;
        case XmlPullParser.START_TAG:
                depth++;
                break;
        }
    }
}
```

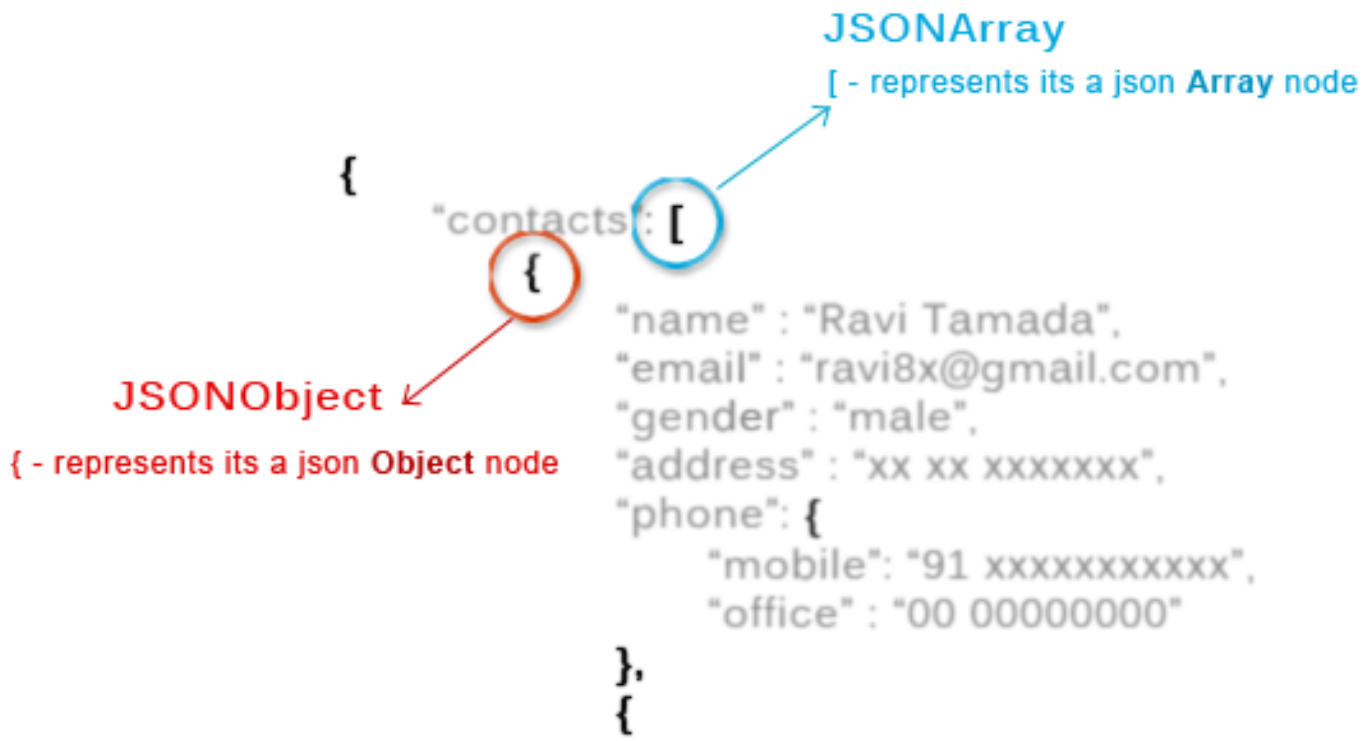
Parsing JSON Data

Parsing JSON Data

- The JSON Structure
- Parsing JSON

Parsing JSON Data (cont'd)

- The JSON Structure



Parsing JSON Data

- Parsing JSON
  - Example:

```
public List<Contact> parse(String data) {
    List<Contact> contacts = new ArrayList<Contact>();
    try {
        JSONObject jObject = new JSONObject(data);
        JSONArray jArray = jObject.getJSONArray("contacts");

        for(int i = 0; i < jArray.length(); i++){
            JSONObject c = jArray.getJSONObject(i);

            // Storing each json item in variable
            String id = c.getString("id");
            String name = c.getString("name");
            String email = c.getString("email");
            String address = c.getString("address");
            String gender = c.getString("gender");

            // Phone number is agin JSON Object
            JSONObject phone = c.getJSONObject("phone");
            String mobile = phone.getString("mobile");
            String home = phone.getString("home");
            String office = phone.getString("office");

            Contact contact = new Contact(id, name, email, address, gender,
                                                                mobile, home, office);

            contacts.add(contact);
        }

    } catch (JSONException e) { }

    return contacts;
}
```

Assignment

Assignment

- Develop an application has a ListView with dynamic data retrieved from http://api.androidhive.info/music/music.xml

Refferences

- Performing Network Operations - http://developer.android.com/training/basics/network-ops/index.html
- Connecting to the Network - http://developer.android.com/training/basics/network-ops/connecting.html
- Managing Network Usage - http://developer.android.com/training/basics/network-ops/managing.html
- Parsing XML Data - http://developer.android.com/training/basics/network-ops/xml.html
- Image loader - http://www.androidhive.info/2012/02/android-custom-listview-with-image-and-text/


Q&A

Thank You


CSC
BUSINESS SOLUTIONS
TECHNOLOGY
OUTSOURCING