Lecture 15: Network

Jianjun Chen (Jianjun.Chen@xjtlu.edu.cn)

Classes for Connection

- Android includes multiple networking support classes, e.g.,
  - java.net - (Socket, URL)
  - org.apache - (HttpRequest, HttpResponse)
  - android.net - (URI, AndroidHttpClient, AudioStream)

- Apps need permission to access the Internet

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

HTTP Support

Using java.net

About HTTP

- HTTP is a protocol for communication between browsers and world wide web (WWW) servers.
  - Communication: HTTP documents are transferred.

- HTTP uses URL to locate resources:

```
protocol://hostname[:port]/path/[;parameters][?query]#fragment
```

  - Protocol examples: http, file, ftp, ed2k, mailto
  - Query: parameters for generating server-side dynamic web-pages. Used by PHP, JSP, ASP etc..

General Steps

- Create an `URL` object
- Obtain a new `HttpURLConnection` by calling `URL.openConnection()`
- Get input/output streams and communicate with the remote computer.
- Close the IO streams and `HttpURLConnection`.

Form Handling: Post VS GET

The difference is how forms are sent to the server:

- Using GET, key value pairs are encoded into the URL:

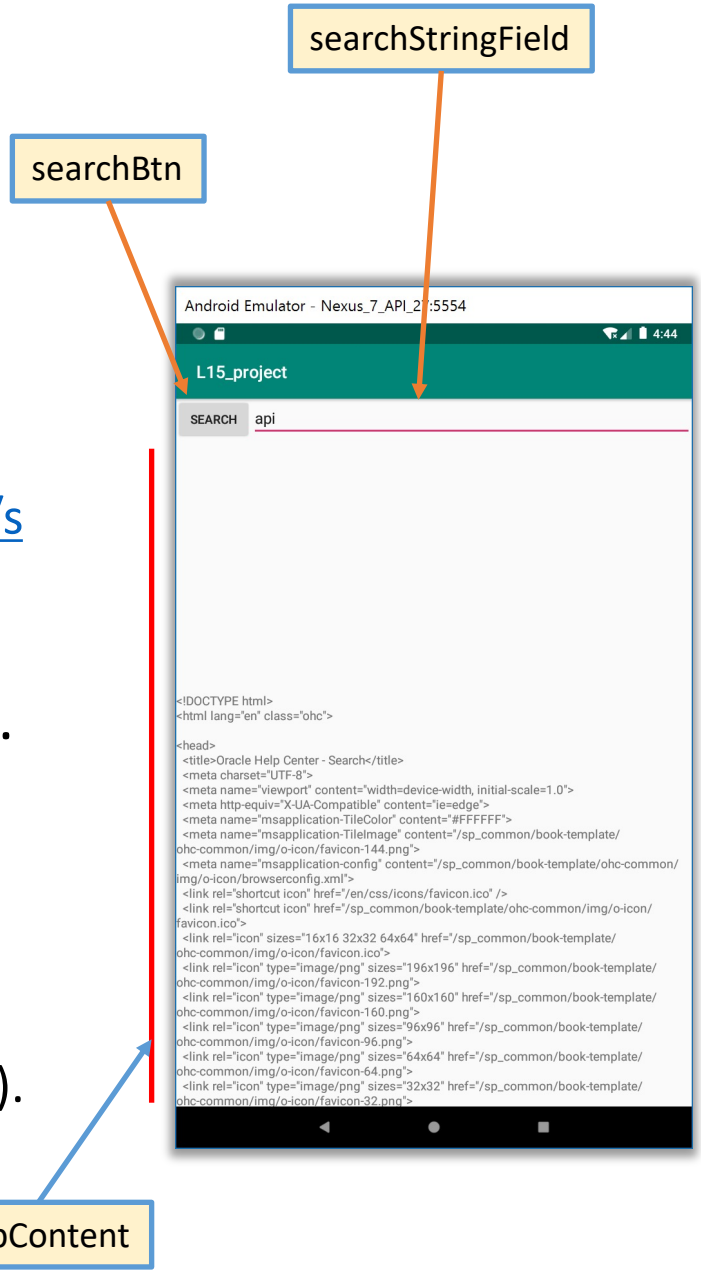
```
form_handler.php?name=Jianjun&email=jianjun.chen%40nuts.com
```

- Using POST, key value pairs are embedded within the body of the HTTP request. Not visible in the URL.
    - Safer, no data size limitation. But CANNOT be bookmarked.

- Check this link for more details (if the server uses PHP): https://www.w3schools.com/php/php_forms.asp

Example: HTTP GET

- In the next example, we will design an app that fetches query results from:
  - https://docs.oracle.com/apps/search/search.jsp
  - This page requires browsers to use GET to submit form queries.
- Search starts whenever the "search" button is pressed
  - The query string is provided by the EditText widget (right side of "search").



Step 1: Activity

```
protected void onCreate(Bundle savedInstanceState) {
    ...
    Button searchBtn = findViewById(R.id.searchOracleBtn);
    final EditText searchStringField = findViewById(R.id.searchStringField);
    searchBtn.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
            String httpURL = "https://docs.oracle.com/apps/search/search.jsp?q="
                    + searchStringField.getText();
            TextView httpContent = findViewById(R.id.httpContent);
            httpContent.setText("Searching...");
            try {
                URL url = new URL(httpURL);
                QueryTask task = new QueryTask();
                task.execute(url);
            } catch (MalformedURLException e) {
                System.out.println(e.getMessage());
            }
        }
    });
}
...
```

Our `AsyncTask`

Step 2.1: Fetch Data (AsyncTask)

```
class QueryTask extends AsyncTask<URL, Void, String> {
    protected String doInBackground(URL... urls) {
        if (urls.length == 0)
            return null;

        try {
            HttpURLConnection urlConn =
                  (HttpURLConnection) urls[0].openConnection();
            BufferedReader reader = new BufferedReader(
                    new InputStreamReader(urlConn.getInputStream()));
            String res = "", line = null;
            while ((line =  reader.readLine()) != null) {
                res += line + '\n';
            }
            reader.close();
            urlConn.disconnect();
            return res;
        } catch (IOException e) {
            return null;
        }
    }
...
}
```

We should not block the UI thread.
This example uses AsyncTask.

Step 2.2: Fetch Data (AsyncTask)

```
class QueryTask extends AsyncTask<URL, Void, String> {
    @Override
    protected String doInBackground(URL... urls) {
        ...
    }

    @Override
    protected void onPostExecute(String result) {
        TextView httpContent = findViewById(R.id.httpContent);
        if (result != null) {
            httpContent.setText(result);
        } else {
            httpContent.setText("Failed to fetch search result");
        }
    }
}
```

Parsing HTTP Response (XML)

- It is quite common for Web APIs to return XML or JSON documents as query results.
- Here, XML parsing using DOM is briefly introduced.
  - “Parses an XML document by loading the complete contents of the document and creating its complete hierarchical tree in memory.”
- For more information
  - https://www.tutorialspoint.com/java_xml/java_xml_parsers.htm

Example Data

http://api.geonames.org/earthquakes?north=44.1&south=-9.9&east=-22.4&west=55.2&username=demo

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<geonames>
    <earthquake>
        <src>us</src>
        <eqid>c0001xgp</eqid>
        <datetime>2011-03-11 04:46:23</datetime>
        <lat>38.322</lat>
        <lng>142.369</lng>
        <magnitude>8.8</magnitude>
        <depth>24.4</depth>
    </earthquake>
    <earthquake>
        <src>us</src>
        <eqid>c000905e</eqid>
        <datetime>2012-04-11 06:38:37</datetime>
        <lat>2.311</lat>
        <lng>93.0632</lng>
        <magnitude>8.6</magnitude>
        <depth>22.9</depth>
    </earthquake>
    <earthquake>
        <src>us</src>
        <eqid>2007hear</eqid>
        <datetime>2007-09-12 09:10:26</datetime>
```

Example Code

```
class XMLQueryTask extends AsyncTask<Void, Void, String> {
    @Override
    protected String doInBackground(Void... voids) {
        try {
            URL url = new URL("http://api.geonames.org/earthquakes?north=44.1&south=-9.9&east=-
22.4&west=55.2&username=demo");
            if (url != null) {
                HttpURLConnection urlConn = (HttpURLConnection) url.openConnection();
                DocumentBuilder dBuilder =
                                DocumentBuilderFactory.newInstance().newDocumentBuilder();
                Document doc = dBuilder.parse(urlConn.getInputStream());
                String str = "root node name is " + doc.getDocumentElement().getNodeName();
                urlConn.disconnect();
                return str;
            }
        } catch ...
        return null;
    }

    @Override
    protected void onPostExecute(String result) {
        TextView textView = findViewById(R.id.XMLDisplay);
        textView.setText(result);
    }
}
```

`Document.parse()` can also read from `file` and `String`

Socket

Also check this tutorial:

https://docs.oracle.com/javase/tutorial/networking/sockets/readingWriting.html

Definition from Wikipedia

- **"Berkeley sockets** is an application programming interface (API) for Internet sockets and Unix domain sockets, used for inter-process communication (IPC). It is commonly implemented as a library of linkable modules. It originated with the 4.2BSD Unix operating system, released in 1983."
- "A socket is an abstract representation (handle) for the local endpoint of a network communication path."

Socket

- Each socket contains five pieces of information:
  - Protocol used.
  - Local IP and local port.
  - Remote IP and remote port.
- `java.net` Classes:
  - `Socket`: client socket for TCP connections.
  - `DatagramSocket`: UDP connections.
  - `ServerSocket`:
    - A dedicated server socket waits for requests to come in over the network. (`ServerSocket.accept()`)

Socket Example

- The next example runs on PC. However, the code also works on any Android phones.

- Remember to use `AsyncTask` or a non-UI thread when processing network communication.

Client

Target IP & port

```
Socket socket = new Socket("127.0.0.1", 1024);
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));

out.println("Hello from the client");
System.out.println("echo: " + in.readLine());
```

Server

port number

```
ServerSocket server = new ServerSocket(1024);
Socket socket = server.accept();
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));

String line = in.readLine();
System.out.println("Received from the client: " + line);
out.print("Hello from the server");
```

Blocks the thread until a line is received

Server Sockets

- On a real server, you should create a new thread to handle each `Socket` returned from `accept()`.
- This allows your server to handle multiple client requests.

```
ServerSocket server = new ServerSocket(1024);
Socket socket = server.accept();
// create new thread and handle the connection.
```