Http Connection Parsing JSON

HTTP Connection

Android Networking

Steps to connect to the Internet

1. Add permissions to Android Manifest
2. Check Network Connection
3. Create Coroutines
4. Implement background task
    a. Create URI
    b. Make HTTP Connection
    c. Connect and GET Data
5. Process results
    a. Parse Results

Android Networking

Network Prerequisites

- The **<uses-permission>** element must be included in the AndroidManifest.xml resource so as to allow the application to connect to the network
- Permissions are used to ask the operating system to access any privileged resource
- The **<uses-permission>** tag causes the application to request to use an Android resource that must be authorized
  - The tag must be an immediate child of **<manifest>**

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

Android Networking

Getting network information for API >= 29

- ConnectivityManager
  - Answers queries about the state of network connectivity
  - Notifies applications when network connectivity changes
- NetworkCapabilities
  - Describes capability of a network interface of a given transport type
  - Cellular or Wi-Fi

Getting network information for API >= 29

```
val connectivityManager = getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager
val networkCapabilities = connectivityManager.getNetworkCapabilities(connectivityManager.activeNetwork)
if (networkCapabilities == null) {
    Log.v("TAG", "No connection")
} else {
    if (networkCapabilities.hasTransport(NetworkCapabilities.TRANSPORT_WIFI)) {
        Log.v("TAG", "Has WIFI connection")
    }
    if (networkCapabilities.hasTransport(NetworkCapabilities.TRANSPORT_CELLULAR)) {
        Log.v("TAG", "Has cellular connection")
    }
}
```

Android Networking

Getting network information for API < 29

- ConnectivityManager
  - Answers queries about the state of network connectivity
  - Notifies applications when network connectivity changes
- NetworkInfo
  - Describes status of a network interface of a given type
  - Mobile or Wi-Fi

Check if network is available

```
ConnectivityManager cm = (ConnectivityManager)
getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo networkInfo = cm.getActiveNetworkInfo();
if (networkInfo != null && networkInfo.isConnected())
    Log.v("TAG", "Connected");
else
    Log.v("TAG", "No network connection");
```

Check for Wifi and mobile

```
NetworkInfo networkInfo =
    connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
boolean isWifiConn = networkInfo.isConnected();

networkInfo =
    connMgr.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
boolean isMobileConn = networkInfo.isConnected();
```

Protocols

- Android supports several different network protocols.
  - TCP / IP (through the `Socket` class)
  - SMTP (through the `GMailSender` class)
  - HTTP
  - And others
- In this lesson, you will work with HTTP

HTTP

- Hyper Text Transfer Protocol
- In our case the transfer protocol is HTTP
  - We connect the client device to a server and get data
  - We then process that data somehow
    - We might render a Web page
    - We might parse and process XML
    - Or any other message

HTTP

- Two HTTP clients
  - `HttpClient`
  - `HttpURLConnection`
- Both support HTTPS and IPV6
- Use `HttpURLConnection` for post Lollipop devices (version 5.x)

The URL class

- The `java.net.URL` class represents a url
  - Convert strings to URLs
  - Convert URLs to strings
- http://developer.android.com/reference/java/net/URL.html

The URL class

Protocol	http
Authority	username:password@host:8080
User Info	username:password
Host	host
Port	8080
File	/directory/file?query
Path	/directory/file
Query	query
Ref	ref

Opening a connection

- The `URL.openConnection()` method establishes a connection to a resource
- Over this connection, you make the request and get the response
- We will use HTTP here but other protocols are supported

Opening a connection

- The `setReadTimeout()` mutator defines the time to wait for data
- The `setConnectTimeout()` mutator the time to wait before establishing a connection
- The `setRequestMethod()` defines whether the request will be a GET or a POST

Opening a connection

- `getResponseCode()` gets the HTTP response code from the server
  - -1 if there is no response code.
  - Such as 404 not found?
- `getInputStream()` gets the input stream from which you can read data
  - Works just like an open file

Sending GET request

```
lifecycleScope.launch(Dispatchers.IO) {
    val url = URL("https://httpbin.org/get")
    val conn = url.openConnection() as HttpURLConnection
    Log.v("TAG", "Response code: ${conn.responseCode}")

    val reader = conn.inputStream.reader()
    val content = reader.readText()
    reader.close()

    // TODO: Process result
}
```

Download file

```
lifecycleScope.launch(Dispatchers.IO) {
    val url = URL("https://lebavui.github.io/videos/ecard.mp4")
    val conn = url.openConnection() as HttpURLConnection
    Log.v("TAG", "Response code: ${conn.responseCode}")

    val reader = conn.inputStream
    val writer = openFileOutput("download.mp3", MODE_PRIVATE)

    val buffer = ByteArray(2048)

    while (true) {
        val len = reader.read(buffer)
        if (len <= 0)
            break
        writer.write(buffer, 0, len)
    }

    writer.close()
    reader.close()
}
```

Sending POST request

```
lifecycleScope.launch(Dispatchers.IO) {
    val url = URL("https://httpbin.org/post")
    val conn = url.openConnection() as HttpURLConnection

    conn.requestMethod = "POST"
    val params = "user=admin&password=123456"
    conn.doOutput = true
    val writer = conn.outputStream.writer()
    writer.write(params)
    writer.flush()
    writer.close()

    Log.v("TAG", "Response code: ${conn.responseCode}")

    val reader = conn.inputStream.reader()
    val content = reader.readText()
    reader.close()

    // TODO: Process result
}
```

Android Networking

3rd party libraries

Http Client:

- Volley https://github.com/google/volley
- OKHttp http://square.github.io/okhttp/
- Retrofit http://square.github.io/retrofit/

Image Loader:

- Picasso http://square.github.io/picasso/
- Universal Image Loader https://github.com/nostra13/Android-Universal-Image-Loader

Parsing JSON

Parsing JSON

What is JSON

- **J**ava**S**cript **O**bject **N**otation
- JSON is a syntax for storing and exchanging data
- JSON is text, written with JavaScript object notation
- Completely language independent
- Easy to understand, manipulate and generate

Parsing JSON

JSON syntax

- Data is in name/value pairs

```
"name":"John"
```

- Data is separated by commas

```
"name":"John", "age":20, "gender":"male"
```

- Curly braces hold objects

```
{"name":"John", "age":20, "gender":"male"}
```

- Square braces hold arrays

```
[{"name":"John", "age":20,
"gender":"male"}, {"name":"Peter", "age":21,
"gender":"male"}, {"name":"July", "age":19,
"gender":"female"}]
```

Parsing JSON

JSON values

- In JSON, values must be one of the following data types:
  - A string

```
{ "name":"John" }
```

  - A number

```
{ "age":30 }
```

  - An object

```
{"employee":{ "name":"John", "age":30, "city":"New York" }}
```

  - An array

```
{"employees":[ "John", "Anna", "Peter" ]}
```

  - A Boolean

```
{ "sale":true }
```

  - Null

```
{ "middlename":null }
```

Parsing JSON

Parsing JSON string

- Basic types such as *strings*, *numbers*, and *Boolean values*, are represented in Java as their corresponding types (String, int/double, boolean, respectively).
- Compound types are represented using classes in the org.json package.
  - JSON arrays are represented by the class org.json.JSONArray;
  - JSON objects are represented by the class org.json.JSONObject.
- *Null values* are represented by the instance JSONObject.NULL.

Parsing JSON

Parsing JSON string

- To parse compound JSON data from a String, create a new Java object of the appropriate type, passing the String as the only argument to the constructor.

```
// Parsing JSON object
val jsonData = JSONObject(jsonString)

// Parsing JSON array
val jsonData = JSONArray(jsonString)
```

Parsing JSON

Parsing JSON string – Retrieving data

- Values for the keys in a JSONObject may be obtained using ***get*(String key)*** methods
  - ***getBoolean(key)*** will get a boolean value
  - ***getInt(key)*** will get an int value
  - ***getString(key)*** will get a String value
  - ***getJSONObject(key)*** will get a JSONObject value
  - ***isNull(String key)*** may be used to test if the value of a key is null. It will also return true if key does not exist in the JSONObject
- ***keys()*** will return an iterator Java object (java.util.Iterator) which you can use to iterate through the keys in a JSONObject.
- Values in a JSONArray may be obtained using ***get*(int index)***
- Both JSONObject and JSONArray provide the ***count()*** method to return the number of items in the *object*/*array*

Parsing JSON

Example: Parsing JSON data from URL

- Sample URL contains JSON data
  https://jsonplaceholder.typicode.com/users

- Steps:
  - Use coroutine to get data in background
  - Implement GET request by using URLConnection
  - Parse JSON by using JSONArray and JSONObject classes
  - Show data using ListView

Serialize object to JSON

Convert JAVA object to JSON string

- Use GSON library https://github.com/google/gson
- Main functions:
  - toJson() serialize object to Json
  - fromJson() deserialize json to object
- Example:

```
val gson = Gson() // Or use new GsonBuilder().create();
val obj1 = MyType()
val json = gson.toJson(obj1) // serializes obj1 to Json
val obj2 = gson.fromJson(json, MyType.class) // deserializes json into obj2
```

Working with Socket class

Working with Socket class

Setup a client socket

```
lifecycleScope.launch(Dispatchers.IO) {
    val serverAddress = InetAddress.getByName("192.168.1.26")
    client = Socket()
    client?.connect(InetSocketAddress(serverAddress, 8000))

    withContext(Dispatchers.Main) {
        if (client?.isConnected == true) {
            Toast.makeText(applicationContext, "Connected", Toast.LENGTH_LONG).show()

            // TODO: Read message from server

        }
        else
            Toast.makeText(applicationContext, "Connect Failed", Toast.LENGTH_LONG).show()
    }
}
```

Working with Socket class

Receive data from server

```
withContext(Dispatchers.IO) {
    val reader = client?.getInputStream()?.reader()
    val bufferedReader = BufferedReader(reader)
    while (client?.isConnected == true && client?.isClosed == false) {
        val line = bufferedReader.readLine()
        if (line.isNullOrEmpty())
            break
        Log.v("TAG", "Received: $line")
        withContext(Dispatchers.Main) {
            binding.textLog.append(line)
        }
    }
    Log.v("TAG", "Disconnected")
}
```

Working with Socket class

Send data to server

```
lifecycleScope.launch(Dispatchers.IO) {
    if (client?.isConnected == true && client?.isClosed == false) {
        val writer = client?.getOutputStream()?.writer()
        writer?.write(message)
        writer?.flush()

        withContext(Dispatchers.Main) {
            binding.editMessage.setText("")
        }
    }
}
```