Recording in Progress

This class is being recorded

Please turn off your video and/or video if you do not wish to be recorded

CMSC436: Programming Handheld Systems

Networking

Today’s Topics

Networking

Android networking classes

Processing HTTP responses

Networking

Early handheld devices gave us mobility, but had limited connectivity

Today’s devices have greater mobility and connectivity

Today, many applications use data and services via the Internet

Networking

Android includes multiple networking support classes, e.g.,

java.net – (Socket, URL, URLConnection)

Example Application

Sends a request to a networked server for earthquake data

Receives the earthquake data

Displays the requested data

Sending HTTP Requests

Socket

HttpURLConnection

OkHttpClient

Networking Permissions

Applications need permission to open network sockets

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

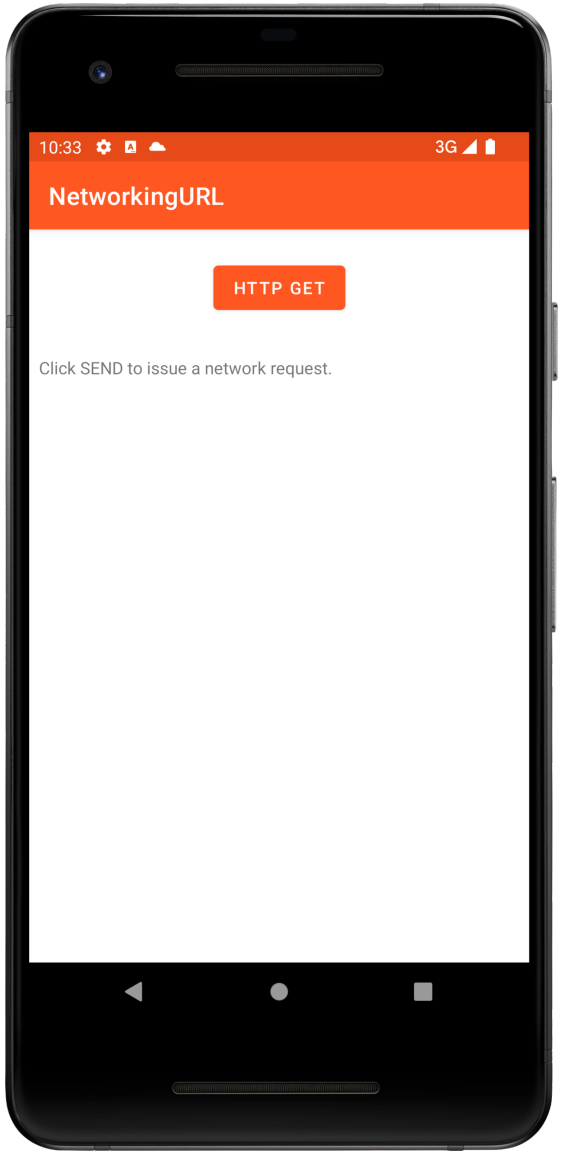
OkHttpClient

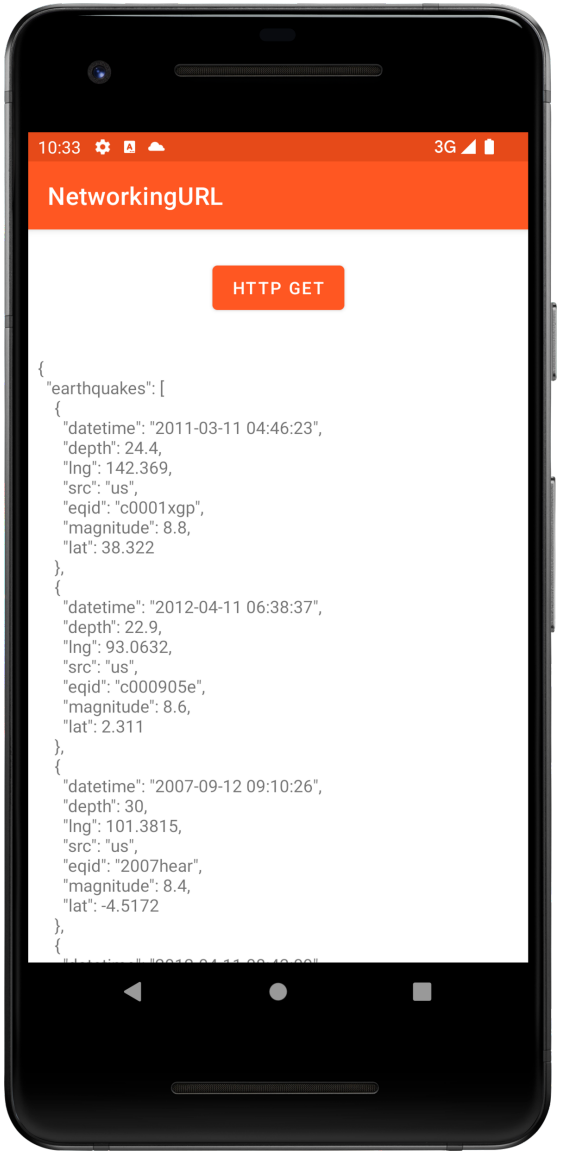
Higher-level than using raw sockets

Usage Pattern for Http Get

1. Get an OkHttpClient instance
2. Prepare your request
3. Issue get() call (or related method)
4. Read the response
5. Process response

Networking URL





MainViewModel.kt

```
fun onSendButtonClicked() {
    …
    // Launch a new coroutine to run network request in the background.
    job = viewModelScope.launch {
        try {
            // 1. Run the suspending network request.
            val rawJson = makeNetworkCall(URL)

            // 2. Post the returned JSON string to the LiveData feed.
            _liveData.postValue(rawJson.prettyPrint())
        } catch (e: Exception) {
            …
        }
    }
}
```

MainViewModel.kt

```
private suspend fun makeNetworkCall(url: String): String =
    withContext(Dispatchers.IO) {
        // Construct a new Ktor HttpClient to perform the get
        // request and then return the JSON result.
        HttpClient().get(url)
    }
    …
companion object {
        …
    // Get your own user name at http://www.geonames.org/login
    private const val USER_NAME = "aporter"

    private const val HOST = "api.geonames.org"
    private const val URL = "http://$HOST/earthquakesJSON?" +
     "north=44.1&" + "south=-9.9&"+"east=-22.4&" + "west=55.2&"+
     "username=$USER_NAME"
}
```

Processing Http Responses

Will focus on two popular formats:

- JSON
- XML

Javascript Object Notation (JSON)

A lightweight data interchange format

Data packaged in two types of structures:

- Maps of key/value pairs
- Ordered lists of values

See: http://www.json.org/

Earthquake Data Request (JSON)

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

JSON Response

{"earthquakes”: [

{"eqid":"c0001xgp","magnitude":8.8,"lng":142.369, "src":"us”, "datetime":"2011-03-11 04:46:23","depth":24.4,"lat":38.322}

{"eqid":"2007hear","magnitude":8.4,"lng":101.3815,src":"us”,"datetime":"2007-09-12 09:10:26","depth": 30,"lat":-4.5172},

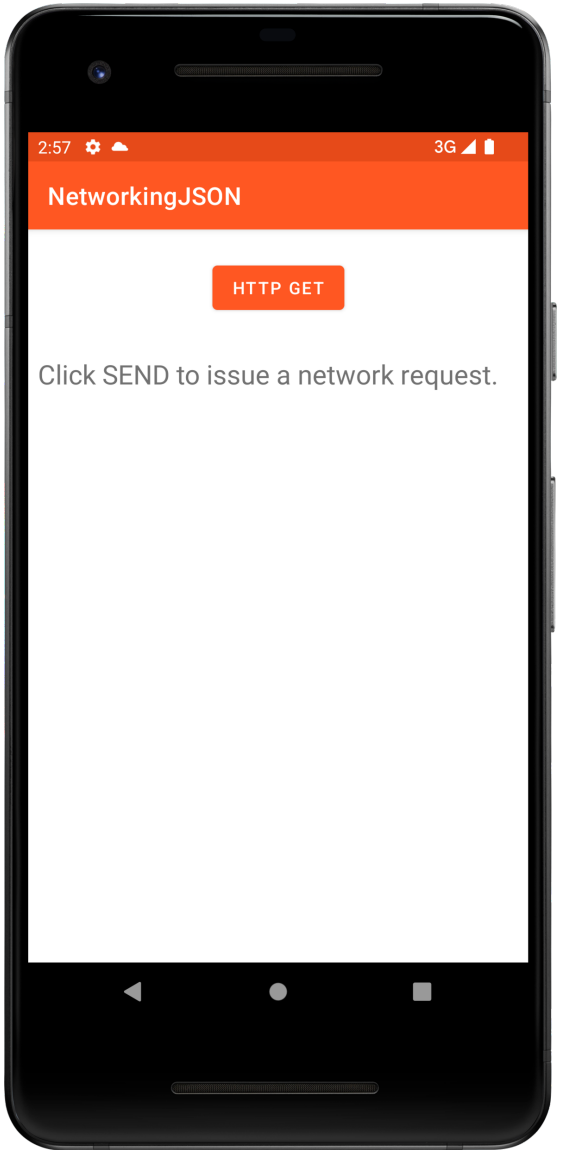
…

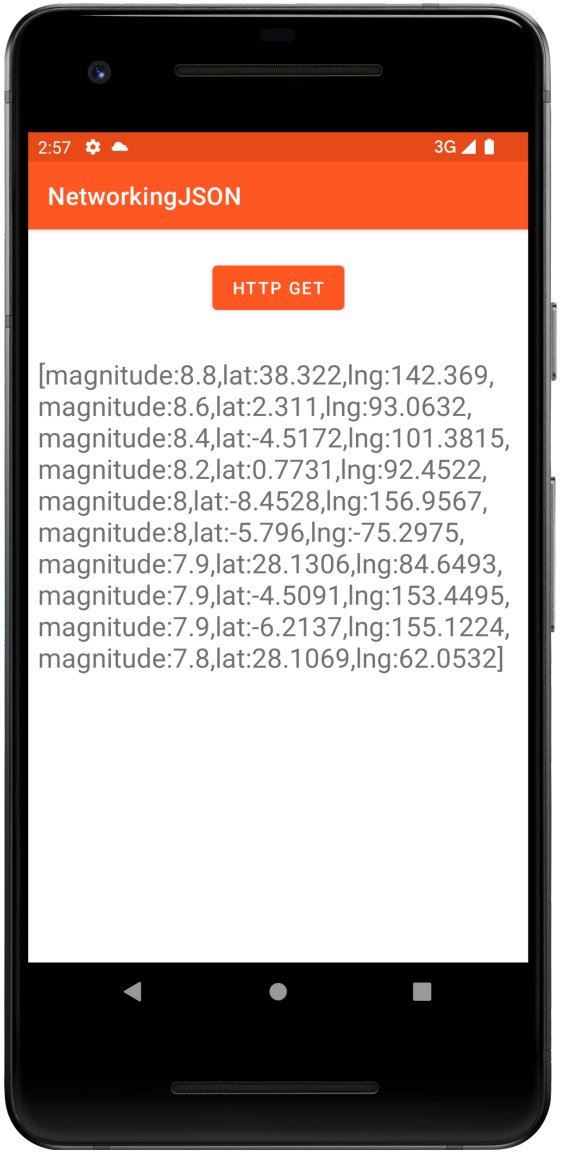
{"eqid":"2010xkbv","magnitude":7.5,"lng":91.9379,"src":"us”,"datetime":”2010-06-12 17:26:50","depth":35,"lat":7.7477}

]

}

Networking JSON





MainViewModel.kt

```
fun onSendButtonClicked() {
    …
    // Launch a new coroutine to run network request in the background.
    job = viewModelScope.launch {
        try {
            // 1. Run the suspending network request.
            val rawJson = makeNetworkCall(URL)

            // 2. Post the returned JSON string to the LiveData feed.
            _liveData.postValue(parseJsonString(rawJson))
        } catch (e: Exception) {
            …
        }
    }
}
```

MainViewModel.kt

```
private fun parseJsonString(data: String?): List<String> {
    val result = ArrayList<String>()
    try {
        // Get top-level JSON Object - a Map
        val responseObject = JSONTokener(data).nextValue() as JSONObject

        // Extract value of "earthquakes" key -- a List
        val earthquakes = responseObject.getJSONArray(EARTHQUAKE_TAG)

        // Iterate over earthquakes list
        for (idx in 0 until earthquakes.length()) {

            // Get single earthquake mData - a Map
            val earthquake = earthquakes.get(idx) as JSONObject
```

MainViewModel.kt

```
            // Summarize earthquake mData as a string and add it to
            // result
            result.add(MAGNITUDE_TAG + ":"
                 + earthquake.get(MAGNITUDE_TAG) + ","
                 + LATITUDE_TAG + ":"
                 + earthquake.getString(LATITUDE_TAG) + ","
                 + LONGITUDE_TAG + ":"
         + earthquake.get(LONGITUDE_TAG))
        }
    } catch (e: JSONException) {
        e.printStackTrace()
    }
    return result
}
```

eXtensible Markup Language (XML)

XML documents can contain markup & content

Markup encodes a description of the document's storage layout and logical structure

Content is everything else

See http://www.w3.org/TR/xml

Earthquake Data (XML)

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

XML Response

```
<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>
…
</geonames>
```

Parsing XML

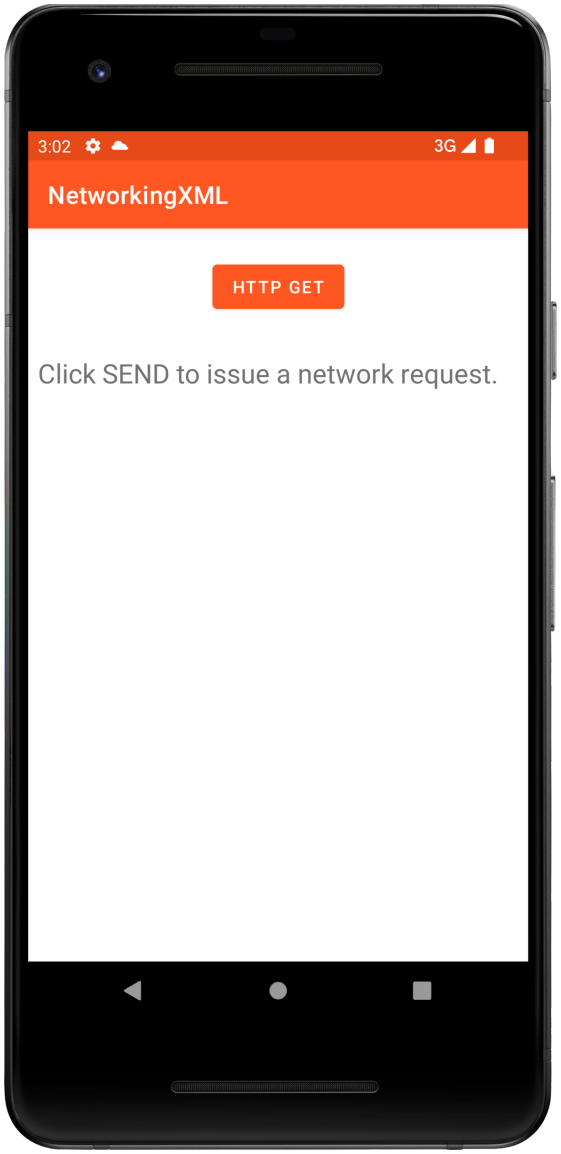
Several types of parsers available

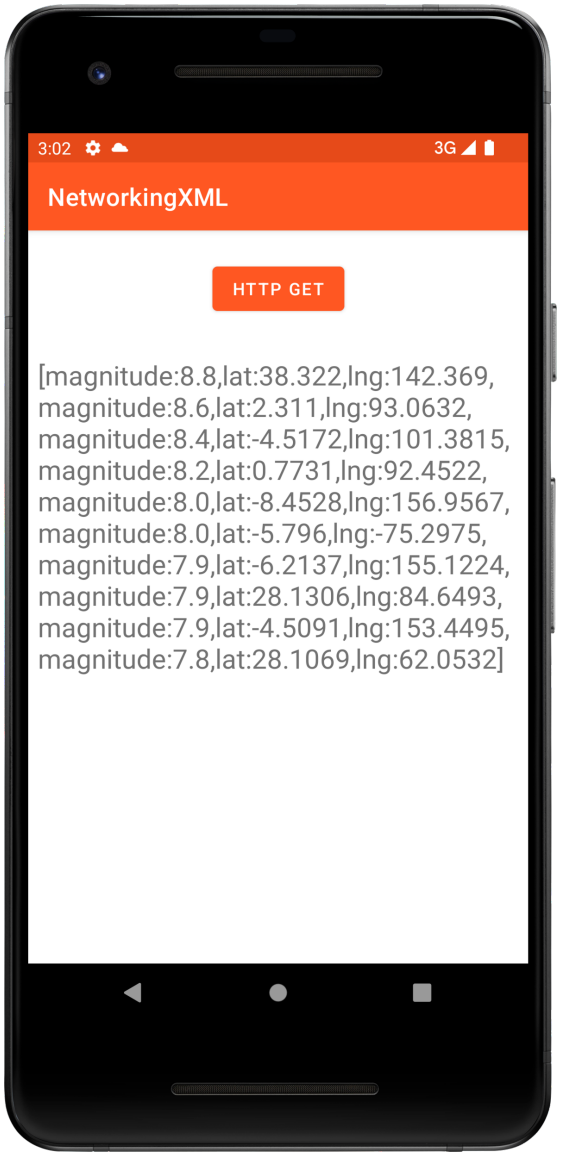
DOM – Converts document into a tree of nodes

SAX – streaming with application callbacks

Pull – Application iterates over XML entries

Networking XML





MainViewModel.kt

```
fun onSendButtonClicked() {
    …
    // Launch a new coroutine to run network request in the background.
    job = viewModelScope.launch {
        try {
            // 1. Run the suspending network request.
            val rawXML = makeNetworkCall(URL)

            // 2. Post the returned JSON string to the LiveData feed.
            _liveData.postValue(parseXmlString(rawXML))
        } catch (e: Exception) {
            …
        }
    }
}
```

MainViewModel.kt

```
private fun parseXmlString(data: String?): List<String>? {
    try {
      // Create the Pull Parser
      val factory = XmlPullParserFactory.newInstance()
      val xpp = factory.newPullParser()
      xpp.setInput(StringReader(data!!))

      // Get the first Parser event and start iterating over the XML document
      var eventType = xpp.eventType
      while (eventType != XmlPullParser.END_DOCUMENT) {
          when (eventType) {
            XmlPullParser.START_TAG -> startTag(xpp.name)
            XmlPullParser.END_TAG -> endTag(xpp.name)
            XmlPullParser.TEXT -> text(xpp.text)
          }
          eventType = xpp.next()
      }
```

MainViewModel.kt

```
 return mResults
    } catch (e: XmlPullParserException) {
        e.printStackTrace()
    } catch (e: IOException) {
        e.printStackTrace()
    }
    return null
}
```

MainViewModel.kt

```
private fun startTag(localName: String) {
    when (localName) {
        LATITUDE_TAG -> mIsParsingLat = true
        LONGITUDE_TAG -> mIsParsingLng = true
        MAGNITUDE_TAG -> mIsParsingMag = true
    }
}

private fun text(text: String) {
    when {
        mIsParsingLat -> mLat = text.trim { it <= ' ' }
        mIsParsingLng -> mLng = text.trim { it <= ' ' }
        mIsParsingMag -> mMag = text.trim { it <= ' ' }
    }
}
```

MainViewModel.kt

```
private fun endTag(localName: String) {
    when (localName) {
        LATITUDE_TAG -> mIsParsingLat = false
        LONGITUDE_TAG -> mIsParsingLng = false
        MAGNITUDE_TAG -> mIsParsingMag = false
        EARTHQUAKE_TAG -> {
            mResults.add(MAGNITUDE_TAG + ":" + mMag + "," + LATITUDE_TAG + ":"
                    + mLat + "," + LONGITUDE_TAG + ":" + mLng)
            mLat = null
            mLng = null
            mMag = null
        }
    }
}
```

Next Time

Graphics and Animation

Example Applications

NetworkingURL

NetworkingJSON

NetworkingXML