Computer Science Large Practical: Accessing remote data and XML parsing

Stephen Gilmore
School of Informatics

October 13, 2017

Contents

1. Android system permissions
2. Getting a network connection
3. Accessing remote data
4. Parsing XML on Android

Android system permissions

Android system permissions

- By default, no Android application has permission to perform any operations that would adversely impact other applications, the operating system, or the user.
- This includes reading or writing the user's private data, reading or writing another application's files, performing network access, keeping the device awake, and so on.

See http://developer.android.com/guide/topics/security/permissions.html

Semantics of permissions

- When an Android application requests a permission to perform some action or access some service the question which is being asked is *May I . . . ?* not *Can I . . . ?*.
- That is, even if the application has permission to access a service, there is no guarantee that the service is available.

Permitted	Available	Result
No	No	Refused
No	Yes	Refused
Yes	No	Fail
Yes	Yes	Success

Granting permissions

The way in which Android asks the user to grant permissions depends on the system version, and the system version targeted by your app:

- If the device is running Android 6.0 “Marshmallow” (API level 23) or higher, and the app’s targetSdkVersion is 23 or higher, the app requests permissions from the user at run-time.
- If the device is running Android 5.1 “Lollipop” (API level 22) or lower, or the app’s targetSdkVersion is 22 or lower, the system asks the user to grant the permissions when the user installs the app.

See http://developer.android.com/guide/topics/security/permissions.html

Normal and dangerous permissions

System permissions are divided into several protection levels. The two most important protection levels to know about are *normal* and *dangerous* permissions:

- **Normal permissions** cover areas where your app needs to access data or resources outside the app's sandbox, but where there's very little risk to the user's privacy or the operation of other apps.
- **Dangerous permissions** cover areas where the app wants data or resources that involve the user's private information, or could potentially affect the user's stored data or the operation of other apps.

See http://developer.android.com/guide/topics/security/permissions.html

Normal permissions examples

- ACCESS_NETWORK_STATE
- ACCESS_WIFI_STATE
- BLUETOOTH
- CHANGE_WIFI_STATE
- INTERNET
- MODIFY_AUDIO_SETTINGS
- NFC
- SET_ALARM
- SET_TIME_ZONE
- SET_WALLPAPER
- ...

See developer.android.com/reference/android/Manifest.permission.html

Dangerous permissions examples

- READ_CALENDAR
- WRITE_CALENDAR
- READ_CONTACTS
- WRITE_CONTACTS
- ACCESS_FINE_LOCATION
- ACCESS_COARSE_LOCATION
- RECORD_AUDIO
- SEND_SMS
- READ_SMS
- READ_EXTERNAL_STORAGE
- WRITE_EXTERNAL_STORAGE
- ...

See developer.android.com/reference/android/Manifest.permission.html

Getting a network connection

Getting permission to access the internet

- Note that to perform any network operations, an application manifest must request the permissions: `android.permission.INTERNET` and `android.permission.ACCESS_NETWORK_STATE`.
- As before, permission is requested with the `uses-permission` element in your app manifest (`AndroidManifest.xml`).

See developer.android.com/training/basics/network-ops/connecting.html

Adding permissions to AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="...">

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

  <application
      ...
   </application>
</manifest>
```

If you fail to declare in your manifest a normal permission such as ACCESS_NETWORK_STATE your app will be allowed to execute but will fail at runtime with a java.lang.SecurityException.

Tracking connectivity changes [MainActivity ▸ onCreate]

```
// The BroadcastReceiver that tracks network connectivity changes.
private var receiver = NetworkReceiver()

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

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

Using ConnectivityManager

- An instance of android.net.ConnectivityManager answers queries about the state of network connectivity and identifies the type of connection available.
  - ConnectivityManager.TYPE_BLUETOOTH
  - ConnectivityManager.TYPE_ETHERNET
  - ConnectivityManager.TYPE_MOBILE
  - ConnectivityManager.TYPE_VPN
  - ConnectivityManager.TYPE_WIFI
  - ...

A typical BroadcastReceiver to conserve data use

```
private inner class NetworkReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        val connMgr =
            context.getSystemService(Context.CONNECTIVITY_SERVICE)
            as ConnectivityManager
        val networkInfo = connMgr.activeNetworkInfo
        if (networkPref == wifi &&
                networkInfo?.type == ConnectivityManager.TYPE_WIFI) {
            // Wi-Fi is connected, so use Wi-Fi
        } else if (networkPref == any && networkInfo != null) {
            // Have a network connection and permission, so use data
        } else {
            // No Wi-Fi and no permission, or no network connection
        }
    }
}
```

networkPref is a user setting; either WIFI, or ANY

The Safe Call operator

- The operator "?." which was used in the expression networkInfo?.type on the previous slide is Kotlin's safe call operator.
- This operator is used to prevent NullPointerExceptions when accessing a field or performing a method invocation on an object.
- The expression networkInfo?.type will evaluate to null if networkInfo is null and networkInfo.type if it is not null.
- Uses of the safe call operator will not throw null pointer exceptions.

Accessing remote data

Networking activities on Android

- In order to have the main user interface thread remain responsive in an Android application tasks which take some time to execute (say, a few seconds at least) are executed in a separate thread which runs in the background.
- Any computation which runs in the background and publishes its results on the UI thread is termed an *asynchronous task* and is formed by making a subclass of the class `android.os.AsyncTask<Params, Progress, Result>`.

Networking permissions and exceptions

- Accessing the network on the main Android UI thread is not just discouraged, it is actually forbidden, even when the app has requested android.permission.INTERNET in the app manifest.
- The Android runtime will throw a runtime exception of class android.os.NetworkOnMainThreadException if an Android app attempts to access the network on the main thread (for example by using a java.net.URLConnection).

Asynchronous tasks

- `android.os.AsyncTask<Params, Progress, Result>` performs background operations and publishes results on the UI thread.
- It is designed to be a helper class around `java.lang.Thread` and `android.os.Handler` and does not constitute a generic threading framework.
- An asynchronous task is defined by the methods,
  - `onPreExecute`,
  - `doInBackground`,
  - `onProgressUpdate`, and
  - `onPostExecute`.
- An asynchronous task myTask is invoked by myTask.execute()

Interface `DownloadCompleteListener`

We create a Kotlin interface to define a callback function to be called when the download is complete.

```kotlin
interface DownloadCompleteListener {
    fun downloadComplete(result: String)
}
```

An AsyncTask [DownloadXmlTask ▸ doInBackground]

```
class DownloadXmlTask(private val resources : Resources,
                      private val caller : DownloadCompleteListener,
                      private val summaryPref : Boolean) :
    AsyncTask<String, Void, String>() {

  override fun doInBackground(vararg urls: String): String {
    return try {
      loadXmlFromNetwork(urls[0])
    } catch (e: IOException) {
      "Unable to load content. Check your network connection"
    } catch (e: XmlPullParserException) {
      "Error parsing XML"
    }
  }
  ...
```

Load XML [DownloadXmlTask ▸ loadXmlFromNetwork]

```
private fun loadXmlFromNetwork(urlString: String): String {
    val result = StringBuilder()

    val stream = downloadUrl(urlString)

    // Do something with stream e.g. parse as XML, build result

    return result.toString()
}
```

Download from URL [DownloadXmlTask ▸ downloadUrl]

```
// Given a string representation of a URL, sets up a connection and gets
// an input stream.
@Throws(IOException::class)
private fun downloadUrl(urlString: String): InputStream {
    val url = URL(urlString)
    val conn = url.openConnection() as HttpURLConnection
    // Also available: HttpsURLConnection

    conn.readTimeout = 10000 // milliseconds
    conn.connectTimeout = 15000 // milliseconds
    conn.requestMethod = "GET"
    conn.doInput = true

    // Starts the query
    conn.connect()
    return conn.inputStream
}
```

Returning the result [DownloadXmlTask ▸ onPostExecute]

```
    ...
    override fun onPostExecute(result: String) {
        super.onPostExecute(result)

        caller.downloadComplete(result)
    }
}
```

Parsing XML on Android

Parsing XML on Android

- Once we have an object of class java.io.InputStream, we can start to read content from it, and process it.
- If our content is an XML file then we can use android.util.Xml to build an XmlPullParser to parse the input.

An RSS news feed with tags feed, entry, title, link, summary

```
<feed xmlns="http://www.w3.org/2005/Atom"
  xmlns:creativeCommons="http://backend...."
  xmlns:re="http://purl.org/atompub/rank/1.0">
    <title type="text">newest questions tagged android</title>
    <entry>
        <id>https://stackoverflow.com/q/46613480</id>
        <re:rank scheme="https://stackoverflow.com">0</re:rank>
        <title type="text">Image data send to the next activity</title>
        ...
        <link rel="alternate"
             href="https://stackoverflow.com/questions/46613480/..." />
        ...
        <summary type="html">
            <p>I select image from gallery and ....</p>
        </summary>
    </entry>
    ...
</feed>
```

A class for storing entries [StackOverflowXmlParser ▸ Entry]

We wish to parse the XML and build a list of objects of class Entry.

```
data class Entry(val title: String, val summary: String, val link: String)
```

Instantiate the parser [StackOverflowXmlParser ▸ parse]

```
// We don't use namespaces
private val ns: String? = null

@Throws(XmlPullParserException::class, IOException::class)
fun parse(input : InputStream): List<Entry> {

    input.use {
        val parser = Xml.newPullParser()
        parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES,
            false)
        parser.setInput(input, null)
        parser.nextTag()
        return readFeed(parser)
    }

}
```

Reading the XML feed [StackOverflowXmlParser ▸ readFeed]

```
@Throws(XmlPullParserException::class, IOException::class)
private fun readFeed(parser: XmlPullParser): List<Entry> {
    val entries = ArrayList<Entry>()
    parser.require(XmlPullParser.START_TAG, ns, "feed")
    while (parser.next() != XmlPullParser.END_TAG) {
        if (parser.eventType != XmlPullParser.START_TAG) {
            continue
        }
        // Starts by looking for the entry tag
        if (parser.name == "entry") {
            entries.add(readEntry(parser))
        } else {
            skip(parser)
        }
    }
    return entries
}
```

Reading an RSS entry [StackOverflowXmlParser ▸ readEntry]

```
@Throws(XmlPullParserException::class, IOException::class)
private fun readEntry(parser: XmlPullParser): Entry {
    parser.require(XmlPullParser.START_TAG, ns, "entry")
    var title = ""
    var summary = ""
    var link = ""
    while (parser.next() != XmlPullParser.END_TAG) {
        if (parser.eventType != XmlPullParser.START_TAG)
            continue
        when(parser.name){
            "title" -> title = readTitle(parser)
            "summary" -> summary = readSummary(parser)
            "link" -> link = readLink(parser)
            else -> skip(parser)
        }
    }
    return Entry(title, summary, link)
}
```

Reading a title [StackOverflowXmlParser ▸ readTitle]

<title type="text">Image data send to the next activity</title>

```
@Throws(IOException::class, XmlPullParserException::class)
private fun readTitle(parser: XmlPullParser): String {
    parser.require(XmlPullParser.START_TAG, ns, "title")
    val title = readText(parser)
    parser.require(XmlPullParser.END_TAG, ns, "title")
    return title
}
```

Documentation: [`parser.require`] tests if the current event is of the given type and if the namespace and name match. `null` will match any namespace and any name. If the test is not passed, an exception is thrown.

Reading a link [StackOverflowXmlParser ▸ readLink]

<link rel="alternate" href="https://stackoverflow.com/..." />

```
@Throws(IOException::class, XmlPullParserException::class)
private fun readLink(parser: XmlPullParser): String {
    var link = ""
    parser.require(XmlPullParser.START_TAG, ns, "link")
    val relType = parser.getAttributeValue(null, "rel")
    if (parser.name == "link") {
        if (relType == "alternate") {
            link = parser.getAttributeValue(null, "href")
            parser.nextTag()
        }
    }
    parser.require(XmlPullParser.END_TAG, ns, "link")
    return link
}
```

Reading summaries [StackOverflowXmlParser ▸ readSummary]

```
<summary type="html">
    <p>I select image from gallery and ....</p>
</summary>
```

```
@Throws(IOException::class, XmlPullParserException::class)
private fun readSummary(parser: XmlPullParser): String {
    parser.require(XmlPullParser.START_TAG, ns, "summary")
    val summary = readText(parser)
    parser.require(XmlPullParser.END_TAG, ns, "summary")
    return summary
}
```

Reading text [StackOverflowXmlParser ▸ readText]

```
@Throws(IOException::class, XmlPullParserException::class)
private fun readText(parser: XmlPullParser): String {
    var result = ""
    if (parser.next() == XmlPullParser.TEXT) {
        result = parser.text
        parser.nextTag()
    }
    return result
}
```

Documentation: [`parser.next`] Get next parsing event – element content will be coalesced and only one TEXT event must be returned for whole element content (comments and processing instructions will be ignored and entity references must be expanded or exception must be thrown if entity reference cannot be expanded).

Skipping uninteresting tags [StackOverflowXmlParser ▸ skip]

```
@Throws(XmlPullParserException::class, IOException::class)
private fun skip(parser: XmlPullParser) {
    if (parser.eventType != XmlPullParser.START_TAG) {
        throw IllegalStateException()
    }
    var depth = 1
    while (depth != 0) {
        when (parser.next()) {
            XmlPullParser.END_TAG -> depth--
            XmlPullParser.START_TAG -> depth++
        }
    }
}
```

Concluding remarks

- Android APIs sometimes throw exceptions on failure and sometimes return null on failure: check the documentation to find out if a method can return null.

Links

- http://developer.android.com/training/basics/network-ops/connecting.html
- http://developer.android.com/training/basics/network-ops/xml.html