

Android - HTTP, XML, JSON

HTTP

The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers.

HTTP works as a request-response protocol between a client and server.

A web browser or our application may be the client, and an application on a computer that hosts a web site may be the server.

Example: A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

Get Vs Post requests

Two commonly used methods for a request-response between a client and server are: GET and POST.

The main difference between GET and POST requests is where the request data is passed.

In Get request the data and the parameters are passed in the URL. This is why they are limited in data length and considered to be less secure. More than that they remain in the browser history can be cached and bookmarked.

In oppose to Get, **POST request contain a special body to hold the data** passed in the request. They are used to submit data to be processed to a specified resource, **they have no restrictions on data length and are more suitable when dealing with large amount of data or sensitive data.** They are never cached, bookmarked and do not remain in the browser history.

Opening HTTP client communication in Android

The first thing to remember is that all **HTTP communication must be done from background thread** in order for your app to be responsive and not hold the UI thread. There is a special safety mechanism that will throw an exception if the HTTP communication will be made on the main thread of the app (this mechanism can be overridden in the run configuration but will not be explained here and should not be done!).

Using Volley

Volley is an HTTP library that makes networking for Android apps easier and most importantly, faster. Volley offers the following benefits:

- Automatic scheduling of network requests.
- Multiple concurrent network connections.
- Cancellation request API. You can cancel a single request, or you can set blocks or scopes of requests to cancel.
- Strong ordering that makes it easy to correctly populate your UI with data fetched asynchronously from the network.

The easiest way to add Volley to your project is to add the following dependency to your app's build.gradle file:

```
dependencies {  
    ...  
    implementation 'com.android.volley:volley:1.1.1'  
}
```

At a high level, you use Volley by creating a RequestQueue and passing it Request objects. The RequestQueue manages worker threads for running the network operations, reading from and writing to the cache, and parsing responses. Requests do the parsing of raw responses and Volley takes care of dispatching the parsed response back to the main thread for delivery.

A simple GET request should look like this:

```
RequestQueue queue = Volley.newRequestQueue(this);  
String url ="http://www.google.com";  
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,  
    new Response.Listener<String>() {  
        @Override  
        public void onResponse(String response) {}  
    }, new Response.ErrorListener() {  
        @Override  
        public void onErrorResponse(VolleyError error) {}  
    });  
queue.add(stringRequest);  
queue.start();
```

When you want the POST request you need also to override the `StringRequest` function **`getBody()`** where you return the `byte[]` data to be written on the request and the **`getHeaders()`** function which return a `Map` object of all the headers and it's values.

Server to client data interchange

When we communicate with the servers we have two ways of receiving data one is **XML** which is a markup language and the other is **JSON** that transmit data objects through key-value pairs. **Both are human readable machine to machine communication methods to send text data.** Here we will learn how to interpret the data in these formats on Android.

The XML Document Object Model (DOM)

The **Document Object Model (DOM)** defines an interface -independent convention for representing and interacting with objects in XML documents. Meaning that the DOM defines a general interface (not specific to any language but cross-platform) to parse XML documents. The **Document** main objects are **nodes** that are organized in a tree structure, called the **DOM tree**.

The Node object is the primary data type for the entire DOM. According to the DOM, everything in an XML document is a node.

The DOM says:

The entire document is a document node

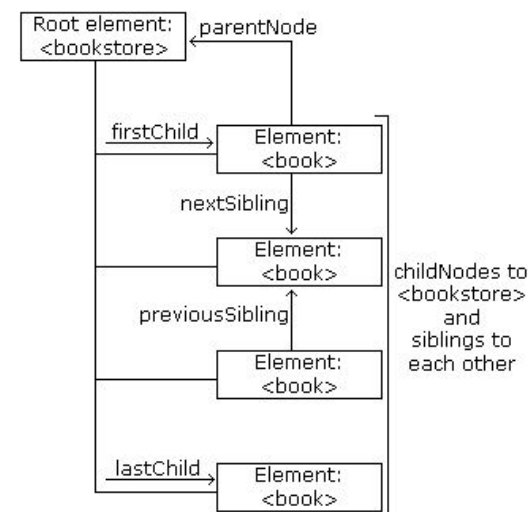
Every XML element is an element node

The text in the XML elements are text nodes

Every attribute is an attribute node.

Comments are comment nodes

(image from W3Schools.com)



IMPORTANT! Text is always stored in text nodes. A common error in DOM processing is to navigate to an element node and expect it to contain the text. However, even the simplest element node has a text node under it. For example, in `<year>2005</year>`, there is an element node (year), and a text node under it, which contains the text (2005).

Because the Element object is also a Node, it inherits the Node object's properties and methods.

So The implementation of the DOM in Android goes as follows:

1. Take the inputStream and create a Document object (inherited from node, as explained above). The Document object is the root of the document tree, and provides the primary access to the document's data. First create a DocumentBuilder using the DocumentBuilderFactory and call parse on the inputStream as follows:

```
Document doc = DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(inputStream);
```

2. Create a NodeList instance, which is an array of nodes, for a specific element tag name using the document's **getElementsByTagName("TAG_NAME")**. It returns a **NodeList** of all descendant Elements with a given tag name, in the document order.
3. For each Node in the array you can check the Node type with the **getNodeNodeType()** function that returns an int (all constants are stored in the Node class) and check whether it is an **ELEMENT_NODE**. If it is you can cast it to **Element** object and continue to go deeper into the xml document tree and use the **getElementsByTagName("TAG_NAME")** to retrieve the NodeList of the next siblings and so forth.
4. When you want to extract the text inside the elements you must remember that even that text is a node a Text Node so get the it by calling on the Element containing the text **getChildNodes()**, you will receive another NodeList and just use the Node **getNodeValue()** to retrieve the text

Parsing JSON response

JSON - JavaScript Object Notation is a format of transmitting data objects consisting of key-value pairs used for browser/server communication, largely replacing [XML](#).

Like the DOM, JSON is an interface. Code for parsing and generating JSON data is available in many languages through the same interface.

The main thing to understand when parsing JSON objects is that Json object is a map of key-value pairs, each Json object is defined by a { } block. When we see, under a certain key the [] brackets it means that this is Json array of json objects, it's always under a certain key in the json object. For example here is how Json looks like:

```
{
  "postalCodes": [
    {
      "adminCode1": "SG",
      "postalCode": "9011",
      "countryCode": "CH",
      "lng": 9.399858534040646,
      "placeName": "St. Gallen",
      "lat": 47.414775328611945,
      "adminName1": "Kanton St. Gallen"
    },
    {
      "adminCode1": "GS",
      "postalCode": "9011",
      "countryCode": "HU",
      "lng": 17.781944437499998,
      "placeName": "Gyor",
      "lat": 47.607638900000005,
      "adminName1": "Gyor-Moson-Sopron"
    },
    {
      "adminCode1": "19",
      "postalCode": "9011",
      "countryCode": "NO",
      "lng": 18.95508,
      "placeName": "Troms",
      "lat": 69.6489,
      "adminName1": "Troms"
    },
    ...
    ...
  ]
}
```

Here the main Json object ({})
has only one key which is
"postalCodes" it's value(:) is an
array([]) of Json objects
separated by commas(,) and
inside each object in the array
we can clearly see the
information passed as a
key-value mapping like
dictionary.

```
}
```

Parsing Json in android

1. First you need to combine an inputStreamReader on your httpURLConnection inputStream and get the result as a String after reading it line after line or char after char.
2. Create a JSONObject instance using it's constructor that takes a String.
3. From here and according to the JSON object for each string key the value can be either:
 - a. String, int, double or other type that it is the information that we want. We can get it through the JSON object getString(key),getInt(key) and more
 - b. Another JSON object and you can retrieve it using the getJSONObject(key)
 - c. JSON array, you can retrieve it by using the getJSONArray(key), go over the array with a loop and retrieve each JSON object using it's index with the JSONArray getJSONObject(index) function.

For example, in the JSON response above let's suppose we want to get the all the values for "placeName" (highlighted) key. we can use the following code:

```
JSONObject jsonObject = new JSONObject(inputStream_to_string);
JSONArray postalCodesItems = jsonObject.getJSONArray("postalCodes");

for (int i = 0; i < postalCodesItems.length(); i++) {
    JSONObject postalCodesItem = postalCodesItems.getJSONObject(i);
    Toast.makeText(context,postalCodesItem.getString("placeName"),Toast.LENGTH_SHORT);
}
```