Android Web Connectivity

Web Connectivity

- Web
  - A global collection of useful resources
  - Client-server architecture connecting users (through agents) with web-servers over the internet using standard protocols
  - Basic Protocol : HTTP
  - Web services support information exchange with other applications
  - Native mobile application acts as a **client** / agent and responsible for interpreting / parsing the resource
- Resource Formats
  - HTML, XML, JSON, Image, etc
- Java / Android API
  - URL
  - HttpURLConnection

HTTP Protocol

- A simple message-based protocol
- Message Types
  - Request
    - represents client request to access a resource (identified by URI)
    - Different request methods supported (e.g. GET, POST, etc)
  - Response
    - Represents server response capturing resource representation in one of the supported formats

Example request

```
GET http://www.google.com HTTP/1.1
Host: www.google.com
```

Example response

```
HTTP/1.1 200
Content-type: text/html
Content-length: xx

<html>
   <head> … </head>
   <body> Google </body>
</html>
```

1. Open connection with web resource

```
try{
   URL url = new URL("../resource");
   HttpURLConnection connection = (HttpURLConnection) url.openConnection();
   connection.setReadTimeout(10000);
   connection.setConnectTimeout(15000);
   connection.setRequestMethod("GET");
   connection.setDoInput(true);
   connection.connect();

   StringBuilder content = new StringBuilder();
   BufferedReader reader = new BufferedReader(
                              new InputStreamReader( connection.getInputStream() ) );


   while( (line = reader.readLine()) != null ){
      content.append(line);
   }
   line = content.toString();

   parse(line);

} catch(Exception ex) {
    ex.printStackTrace();
}
```

2. Download resource as a stream

3. Parse

Example: Pictures resource
(XML and JSON representations)

```
<pictures>

 <category name="Architecture" count="3">
  <im title=”...” url=”...” description=”...” />
  <im title=”...” url=”...” description=”...” />
  <im title=”...” url=”...” description=”...” />
 </category>

 <category name="Art" count="3">
  <im title=”...” url=”...” description=”...” />
  <im title=”...” url=”...” description=”...” />
  <im title=”...” url=”...” description=”...” />
 </category>

</pictures>
```

```
“pictures” : [

  {
     “type” : “category”,
     “name” : "Architecture",
     “count” : "3",
     “images” : [
        {“type” : “im”, “title” : ”...”,  “url”:”...”, “desc : ”...” },
        {“type” : “im”, “title” : ”...”,  “url”:”...”, “desc : ”...” },
        {“type” : “im”, “title” : ”...”,  “url”:”...”, “desc : ”...” }
     ]
  },

  {
     “type” : “category”,
     “name” : "Art",
     “count” : "3",
     “images” : [
        {“type” : “im”, “title” : ”...”,  “url”:”...”, “desc : ”...” },
        {“type” : “im”, “title” : ”...”,  “url”:”...”, “desc : ”...” },
        {“type” : “im”, “title” : ”...”,  “url”:”...”, “desc : ”...” }
     ]
  }
]
```

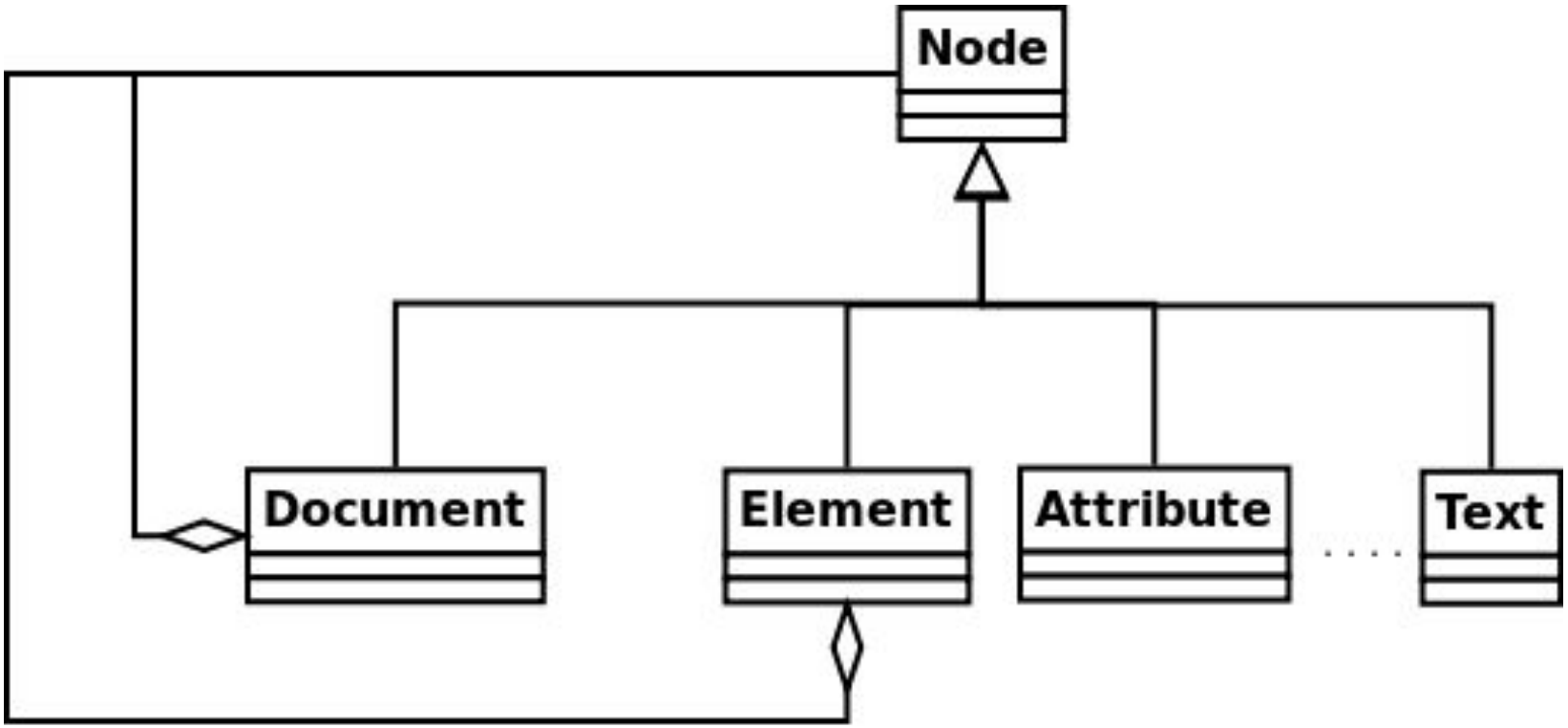
JSON Parsing

```
try {

  JSONArray array = new JSONArray(content);

   for (int i = 0; i < array.length(); i++) {

       JSONObject obj =
array.getJSONObject(i);

       if (obj.getString("type").equals("category")){

          // process category …

       }
   }
} catch (JSONException ex) {

}
```

XML Parsing

- DOM
  - Tree-based
  - Parses entire XML document and provides a tree-based representation of elements in form of a DOM
  - Easy-to-use
  - Memory intensive as well as slow
- Serial
  - Event-based
  - Parses entire XML document and generates events when a specific XML construct encountered
  - Requires customized code for handling events
  - Efficient with low memory footprint
- Pull
  - A good compromise between DOM and Serial parsing
  - Underlying implementation based upon Serial parsing

XML DOM



XML Pull Parsing

- Simple Event Model
  - START_DOCUMENT
  - START_TAG
  - TEXT
  - END_TAG
  - END_DOCUMENT
- Recursive descent parsing
  - Instead of parsing and managing entire document
  - Parse only the portion interested in and skip the rest

```
pictures = new ArrayList<Picture>();

try{
   XmlPullParser parser = Xml.newPullParser();
   parser.setInput(new StringReader(xml));

   int event = parser.getEventType();
   while(event != XmlPullParser.END_DOCUMENT){

      if(event == XmlPullParser.START_TAG &&
          parser.getName().equals("category") ) {

         category = parser.getAttributeValue(null,"name");
      }

      if(event == XmlPullParser.START_TAG &&
          parser.getName().equals("im") ){

          String url = parser.getAttributeValue(null,"url");
          String description = parser.getAttributeValue(null,"description");
          pictures.add(new Picture(url,description,category));
      }

      event = parser.next();
   }
} catch(Exception ex){ }
```

Upload Data

- HTTP POST
  - Request the web server to accept new (or enhanced) resource content
  - Resource is identified by URI
  - Content can be of several types identified and interpreted by content-type
  - No restrictions on data length
  - Used for transmission of data from client to webserver

```
POST http://10.0.2.2/notes/upload.php HTTP/1.1
Content-Type: application/xml
Content-Length: 68

<notes><note title='test note' content='this is test note'/></notes>
```

```
StringBuilder content = new StringBuilder();

try{
   URL url = new URL("http://10.0.2.2/notes/upload.php");
   HttpURLConnection connection = (HttpURLConnection) url.openConnection();
   connection.setReadTimeout(10000);
   connection.setConnectTimeout(15000);
   connection.setRequestMethod("POST");
   connection.setRequestProperty("Content-type","text/xml");
   connection.setDoOutput(true);
   connection.connect();

   BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(connection.getOutputStream()));
   writer.write("<notes><note title='test note' content='this is test note'/></notes>");
   writer.flush();

   BufferedReader reader = new BufferedReader( new
                                                InputStreamReader( connection.getInputStream() ) );

   String line = "";
   while( (line = reader.readLine()) != null ){
     content.append(line);
   }

   connection.disconnect();

} catch(Exception ex) {
    ex.printStackTrace();
}

return content.toString();
```

Best practices

- Download resources in a separate thread, AsyncTask
- Check for connectivity changes
  - Adapt according to network state
    - Wi-fi
    - Mobile radio
    - No connection
  - Use of broadcast receivers and connectivity manager
- Cache data
  - Cache size
  - Synchronization frequency
  - Strategies
- Preferences
  - Allow user to specify connectivity and cache preferences