Android Web Service

Sang Shin
Michèle Garoche
www.javapassion.com
"Learn with Passion!"

Disclaimer

- Portions of this presentation are modifications based on work created and shared by the Android Open Source Project
  - http://code.google.com/policies.html
- They are used according to terms described in the Creative Commons 2.5 Attribution License
  - http://creativecommons.org/licenses/by/2.5/

Topics

- HttpClient API
- Response Formats
- Asynch. Invocation
  - Via a new thread

HttpClient API

HTTP Operations

- Based on Apache HTTP package
- *HttpClient* interface
  - Interface for an HTTP client
  - HTTP clients encapsulate a smorgasbord of objects required to execute HTTP requests while handling cookies, authentication, connection management, and other features
  - Thread safety of HTTP clients depends on the implementation and configuration of the specific client
- *DefaultHttpClient* class
  - Default implementation of HttpClient interface

Invocation Styles in HttpClient

- *public abstract HttpResponse execute (HttpUriRequest request)*
- *public abstract T execute (HttpUriRequest request, ResponseHandler<? extends T> responseHandler)*

Sending HTTP Request

- HttpRequest
- HttpGet
  - Provides HTTP Get
- HttpPost
  - Provides HTTP Post

Receiving HTTP Response

- *HttpResponse*
  - Deals with responses in HTTP MIME type
- *RespondHandler* interface
  - Handler that encapsulates the process of generating a response object from a HttpResponse
- *BasicResponseHandler*
  - A ResponseHandler that returns the response body as a String for successful (2xx) responses. If the response code was >= 300, the response body is consumed and an HttpResponseException is thrown.

Example Code #1

```
HttpClient   httpclient = new DefaultHttpClient();

// Prepare a request object
HttpGet   httpget = new HttpGet(url);

// Execute the request
HttpResponse response;
try {
    response = httpclient.execute(httpget);

    // Get hold of the response entity
    HttpEntity entity = response.getEntity();

    if (entity != null) {

       // A Simple JSON Response Read
       InputStream instream = entity.getContent();
       result = convertStreamToString(instream);
```

Example Code #2

```
HttpClient httpclient = new DefaultHttpClient();

// Prepare a request object
HttpGet httpget = new HttpGet(url);

try {

    ResponseHandler<String> mResponseHandler =
                                        new  BasicResponseHandler();
    result = httpclient.execute(httpget, mResponseHandler);
    ...
```

Response Formats

Response Formats

- XML
- JSON
- RSS, Atom
- ...

XML Parsing

- SAX
- DOM
- Pull-parser

JSON Parsing

- Use *org.json.JSONObject* class
- Example

```
JSONObject json = new JSONObject(result);

// A Simple JSONObject Parsing
JSONArray nameArray = json.names();
JSONArray valArray = json.toJSONArray(nameArray);

// A Simple JSONObject Value Pushing
json.put("sample key", "sample value");

// A simple access to the first name
jsonResult = nameArray.getString(0);
```

Asynch. Invocation

Asynch. Invocation

- Why?
  - Using UI thread to perform Web service invocation, which could take a long time, is not a good idea
- 2 Schemes to use
  - Scheme #1: Create a new thread that performs Web service invocation freeing the UI thread
    - Pass the response to the UI thread via Handler object
  - Scheme #2: Use AsyncTask class

Scheme #1: Creating a New Thread

```
public void onClick(View v) {
    // Create Handler object for displaying the result
    // later on.  This handler is attached to the main
    // thread and its message queue.
    mHandler = new Handler();
    // Start a new thread for performing web service invocation.
    checkUpdate.start();

    // Control is returned here. Do whatever you have to do.
     Toast.makeText(Main.this, "Time: " + new Date().toString(), Toast.LENGTH_LONG).show();
}


// Continued in the following slide

    ...
```

Scheme #1: Creating a New Thread

```
// A separate thread that performs the Web service invocation
private Thread checkUpdate = new Thread() {
    public void run() {
        try {
            response = RESTClient.callRESTService(mEditText.getText().toString());
            // Add the Runnable the message queue of the thread
            // to which mHandler handler is attached.
            mHandler.post(showUpdate);
        } catch (Exception e) {
        }
    }
};

private Runnable showUpdate = new Runnable(){
    public void run(){
        Toast.makeText(Main.this, "Time: " + new Date().toString() + " HTML Code: " + response,
Toast.LENGTH_LONG).show();
    }
};

    ...
```

Scheme #2: Use of AsyncTask

```
public void onClick(View v) {
   new DownloadImageTask().execute("http://example.com/image.png");
}

private class DownloadImageTask extends AsyncTask {

   // The doInBackgroundThread method is called on a separate thread
   protected Bitmap doInBackground(String... urls) {
      return loadImageFromNetwork(urls[0]);
   }

   // The result is communicated to the onPostExecute method which is run on the UI thread.
   protected void onPostExecute(Bitmap result) {
      mImageView.setImageBitmap(result);
   }
}

    ...
```

Thank you!

Check JavaPassion.com Codecamps!
http://www.javapassion.com/codecamps
"Learn with Passion!"