A Case Study of an Android* Client App Using Cloud-Based Alert Service

Abstract

This article discusses a case study of an Android client app using a cloud-based web service. The project was built on the Google* Cloud Messaging (GCM) a free service by Google. It uses a third-party library, Android Asynchronous HttplClient, to post callback-based HTTP requests. It exchanges messages with the app server using JSON strings. The google-gson library is used to parse the JSON strings. The client also consumes web services exposed by the app server through a RESTful API.

Overview

This case study focused on developing an Android client app for a cloud-based healthcare alert service. A doctor or a nurse would have this client app installed on an Android device. The hospital’s central paging system sends short notification messages to the device. The notification plays an alert sound and shows up on the device’s status bar. The user acknowledges or responds to the alert. The user could also browse all the outstanding alerts and the history. The use case was very generic; it could be easily applied to other industries.

Google Cloud Messaging for Android

Google Cloud Messaging for Android (GCM) is a free service by Google that allows you to send messages from your server to an Android device where your app is running (Figure 1). It also allows your server to receive upstream messages from the Android app. For more details on GCM, please go to http://developer.android.com/google/gcm/gcm.html.
Figure 1 An Android® app exchanges data with the app server via GCM

Android Asynchronous Http Client

GCM provides connection servers for HTTP and XMPP. In this project, we used a third-party library, Android Asynchronous Http Client to send requests to the application server. The library is under Apache Version 2 license. It is a callback-based HTTP client, in which HTTP requests happen outside the UI thread. For detailed usages of this library, please go to [http://loopj.com/android-async-http/](http://loopj.com/android-async-http/).

Data Exchange Formats

JSON (JavaScript® Object Notation, [http://www.json.org/](http://www.json.org/)) is a popular data-interchange format used between web services and clients. In our Android client app, we used google-gson to parse the JSON string received from the server. For more information on the google-gson library, please visit [https://code.google.com/p/google-gson/](https://code.google.com/p/google-gson/).
Register with GCM and Application Server

The first time your Android app uses the GCM service, it needs to register with GCM by calling the com.google.android.gcm.GCMRegistrar method register(). In our Android client app, this was done in the onCreate() method of the start activity (Code Example 1).

```java
package com.intcmobile.medipaging;

@override
donCreate(Bundle savedInstanceState) {

registerDeviceWithGCM();

/*
 * Register device with GCM in background
 */
AsyncTask<Void, Void, Void> mRegisterTask;
void registerDeviceWithGCM() {
    if ((SENDER_ID == null) && (SERVER_URL == null)) {
        return;
    }
gcmRegistrationId = GCMRegistrar.getRegistrationId(this);
if (gcmRegistrationId.equals("")) {
    GCMRegistrar.register(this, SENDER_ID);
} else {
    if (!GCMRegistrar.isRegisteredOnServer(this)) {
        final Context context = this;
        mRegisterTask = new AsyncTask<Void, Void, Void>() {
            @Override
            protected Void doInBackground(Void param) {
                boolean registered =
Utilities.serverRegister(context, gcmRegistrationId);
if (!registered) {
    // Asynchronous registration task
    failed. Unregister this device with GCM
        GCMRegistrar.unregister(context);
```
In Code Example 1, we first check if the device is registered with GCM. If not, we register it with the GCM server. Then we check if the device is registered on the app server, if not, we kick off an AsyncTask to register it in the background. Code Example 2 shows the definition of the Utilities class.
package com.intcmobile.medipaging;

import static com.intcmobile.medipaging.Utilities.DEVICE_NAME;
import static com.intcmobile.medipaging.Utilities.SERVER_URL;
import static com.intcmobile.medipaging.Utilities.TAG;
...
import com.google.android.gcm.GCMRegistrar;
...

public class Utilities {
    ...
    /**
     * Handle register/unregister call back from GCMIntentService
     */
    /**
     * Register this account/device pair with the gcm server.
     * This function is normally called from the onRegistered call back
     * function in GCMIntentService
     *
     * @return whether the registration succeeded or not.
     */
    static boolean serverRegister(final Context context, final String regId) {
        String serverUrl = SERVER_URL + "/login";
        Map<String, String> params = new HashMap<String, String>();
        params.put("username", Utilities.userName);
        params.put("password", Utilities.password);
        params.put("regId", regId);
        try {
            post(serverUrl, params);
            GCMRegistrar.setRegisteredOnServer(context, true);
            return true;
        } catch (IOException e) {
            //handle exception
        }
        return false;
    }
    /**
     * Unregister this account/device pair on the server.
     */
    static void serverUnregister(final Context context, final String regId) {
        String serverUrl = SERVER_URL + "/logout";
        Map<String, String> params = new HashMap<String, String>();
        try {
            post(serverUrl, params, regId);
            GCMRegistrar.setRegisteredOnServer(context, false);
        } catch (IOException e) {
            //handle exception
        }
    }
}
/**
 * Issue a GET request to the server
 *
 */

private static void get(String endpoint)
    throws IOException {
    URL url;
    try {
        url = new URL(endpoint);
    } catch (MalformedURLException e) {
        throw new IllegalArgumentException("invalid url: " + endpoint);
    }
    HttpURLConnection conn = null;
    try {
        conn = (HttpURLConnection) url.openConnection();
        conn.setDoOutput(true);
        conn.setUseCaches(false);
        conn.setRequestMethod("GET");
        conn.setRequestProperty("Content-Type","text/html");
        // post the request
        OutputStream out = conn.getOutputStream();
        out.close();
        // handle the response
        int status = conn.getResponseCode();
        if (status != 200) {
            throw new IOException("HTTP GET failed with error code " + status);
        }
    } finally {
        if (conn != null) {
            conn.disconnect();
        }
    }
}

/**
 * Issue a POST request to the server.
 *
 * @param endpoint POST address.
 * @param params request parameters.
 *
 * @throws IOException propagated from POST.
 */
private static void post(String endpoint, Map<String, String> params)
    throws IOException {
    URL url;
    try {
        url = new URL(endpoint);
        // post the request
        OutputStream out = conn.getOutputStream();
        out.close();
        // handle the response
        int status = conn.getResponseCode();
        if (status != 200) {
            throw new IOException("HTTP GET failed with error code " + status);
        }
    } finally {
        if (conn != null) {
            conn.disconnect();
        }
    }
}
} catch (MalformedURLException e) {
    throw new IllegalArgumentException("invalid url: " + endpoint);
}

StringBuilder bodyBuilder = new StringBuilder();
Iterator<Entry<String, String>> iterator =
    params.entrySet().iterator();
    // constructs the POST body using the parameters
    while (iterator.hasNext()) {
        Entry<String, String> param = iterator.next();
        bodyBuilder.append(param.getKey())
            .append('=')
            .append(param.getValue());
        if (iterator.hasNext()) {
            bodyBuilder.append('&');
        }
    }
String body = bodyBuilder.toString();

byte[] bytes = body.getBytes();
HttpURLConnection conn = null;
try {
    conn = (HttpURLConnection) url.openConnection();
    conn.setDoOutput(true);
    conn.setUseCaches(false);
    conn.setFixedLengthStreamingMode(bytes.length);
    conn.setRequestMethod("POST");
    conn.setRequestProperty("Content-Type",
        "application/x-www-form-urlencoded;charset=UTF-8");
    // post the request
    OutputStream out = conn.getOutputStream();
    out.write(bytes);
    out.close();
    // handle the response
    int status = conn.getResponseCode();
    if (status != 200) {
        throw new IOException("HTTP POST operation failed with error code " + status);
    }
} finally {
    if (conn != null) {
        conn.disconnect();
    }
}

Code Example 2 Utility methods used to register and unregister the device to the service running on the third-party app server
Receiving Push Notifications and Sending Confirmations

In this project, we used a service called GCMIntentService to handle GCM push notifications. The service was derived from com.google.android.gcm.GCMBaseIntentService. When the Android client application received a message pushed by the app server, the GCMIntentService class's onMessage(Context context, Intent intent) method was called, no matter if the app is currently active or not.
package com.intcmobile.medipaging;

... import com.google.android.gcm.GCMBaseIntentService; import com.google.android.gcm.GCMRegistrar; import com.google.gson.Gson; import com.google.gson.GsonBuilder;

public class GCMIntentService extends GCMBaseIntentService {

    public GCMIntentService () {
        super(SENDER_ID);
    }

    @Override
    protected void onError(Context context, String errorMessage) {
        displayMessage(context, "There were error receiving notification. Error message: " + errorMessage);
    }

    @Override
    protected void onMessage(Context context, Intent intent) {
        String alertType = intent.getStringExtra("alertType");
        String alertId = intent.getStringExtra("alertId");
        String alertSubject = intent.getStringExtra("subject");
        SimpleAlertInfo alert = new SimpleAlertInfo(alertType, alertId, alertSubject);
        alerts.add(alert);
        displayMessage(context, alert.toString());
    }

    @Override
    protected void onRegistered(Context context, String registrationId) {
        // This method is called when the application successfully register with GCM service
        // register with 3rd party app server
        Utilities.serverRegister(context, registrationId);
    }

    @Override
    protected void onUnregistered(Context context, String registrationId) {
        if (GCMRegistrar.isRegisteredOnServer(context)) {
            Utilities.serverUnregister(context, registrationId);
        }
    }
}

Code Example 3 The GCMIntentService class handles the server push notifications **

We added the GCMIntentService in the app's AndroidManifest.xml file as shown in Code Example 4.
Consuming RESTful Services

Besides receiving pushed alert notifications from the app server, the Android client app also consumes cloud-based web services exposed by the app server API via a REST interface. The basic mechanism for doing this is the Android client app posting an asynchronous HTTP request to the server. If successful, the app server returns a JSON string back to the client app (Code Example 5).
package com.intcmobile.medipaging;
...
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.loopj.android.http.AsyncHttpClient;
import com.loopj.android.http.RequestParams;
...

public class DisplayAlertsActivity extends Activity {
...

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_display_alerts);
        fetchDataForHomeScreen();
    }

    private void fetchDataForHomeScreen () {
        AsyncHttpClient client = new AsyncHttpClient();
        StdRequestParams parameters = new StdRequestParams();
        String url = SERVER_URL + "/getDataForHomeScreen";
        showProgress(true);
        client.post(url, parameters, new AsyncHttpResponseHandler() {
            @Override
            public void onSuccess(String response) {
                jsonDataForHomeScreen = response;
                Gson gson_obj = new GsonBuilder()
                        .setPrettyPrinting()
                        .create();
                medipagingDataForHomeScreen =
                        gson_obj.fromJson(jsonDataForHomeScreen, GetDataResp.class);
                isHomeScreenDataAvailable = true;
                showProgress(false);
                displayAlerts();
            }
            @Override
            public void onFailure(Throwable e, String response) {
                // Response failed :(
                showProgress(false);
            }
            @Override
            public void onFinish() {
                showProgress(false);
            }
        });
    }
...
}
In Code Example 5, we can see the Android client app uses Android Asynchronous Http client to post an HTTP request to the app server. This happens outside of the UI thread so that the Android app could still be responsive to user input. We also see, if successful, a JSON string is returned from the app server. The Android client app uses google-gson to parse the JSON string and create a GetDataResp object. Code Example 6 shows the definition of the GetDataResp class.

```java
package com.intcmobile.common;

import java.util.*;

/**
 * The return from Get Data as well as a bunch of the other calls.
 */
public class GetDataResp extends BaseReturn {
    /**
     * Outstanding alerts.
     */
    public ArrayList<AlertInfo> alertTable;

    /**
     * Constructor.
     */
    public GetDataResp() {
        }

    /**
     * Constructor.
     */
    public GetDataResp(int _returnCode, String _reason) {
        super(_returnCode, _reason);
    }
}
```

**Summary**

In this article, we have discussed a case study of how to use Google Cloud Messaging to create an Android cloud-based alert service client app. In this project, we used a third-party asynchronous HTTP client to achieve the non-blocking UI. We also used JSON as the data exchange format between the app server and the Android client. From this case study, we can see a cloud service Android client app is very easy to implement. The approach discussed in this article can be applied in other cloud-based service client applications.
About the Author

Miao Wei is a software engineer in the Intel Software and Services Group. He currently works on the Intel® Atom™ processor scale enabling projects.

Notices

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL’S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or ”undefined.” Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific
computer systems, components, software, operations, and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license.

Intel, the Intel logo, and Atom are trademarks of Intel Corporation in the U.S. and/or other countries.

Copyright © 2013 Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.

**This sample source code is released under the Intel Sample Source Code License Agreement (http://software.intel.com/en-us/articles/intel-sample-source-code-license-agreement/)