James Smith

Android Asynchronous Http Client

A Callback-Based Http Client Library for Android

Tweet

Overview

Download
version 1.4.9 (latest)

or fork me on github

An asynchronous callback-based Http client for Android built on top of Apache's HttpClient libraries. All requests are made outside of your app's main UI thread, but any callback logic will be executed on the same thread as the callback was created using Android's Handler message passing. You can also use it in Service or background thread, library will automatically recognize in which context is ran.

If you are also looking for a great android crash reporting service, I'd also recommend checking out my company, Bugsnag.

Features

- Using upstream HttpClient of version 4.3.6 instead of Android provided DefaultHttpClient
- Compatible with Android **API 23** and higher
- Make **asynchronous** HTTP requests, handle responses in **anonymous callbacks**
- HTTP requests happen **outside the UI thread**
- Requests use a **threadpool** to cap concurrent resource usage
- GET/POST **params builder** (RequestParams)
- **Multipart file uploads** with no additional third party libraries
- **Streamed JSON uploads** with no additional libraries
- Handling circular and relative redirects
- Tiny size overhead to your application, only **90kb** for everything
- Automatic smart **request retries** optimized for spotty mobile connections
- Automatic **gzip** response decoding support for super-fast requests
- Binary protocol communication with `BinaryHttpResponseHandler`
- Built-in response parsing into **JSON** with `JsonHttpResponseHandler`
- Saving response directly into file with `FileAsyncHttpResponseHandler`
- **Persistent cookie store**, saves cookies into your app's SharedPreferences
- Integration with Jackson JSON, Gson or other JSON (de)serializing libraries with `BaseJsonHttpResponseHandler`

- Support for SAX parser with `SaxAsyncHttpResponseHandler`
- Support for languages and content encodings, not just UTF-8

Used in Production By Top Apps and Developers

Instagram

Instagram is the #1 photo app on android, with over 10million users

Pinterest

Popular online pinboard. Organize and share things you love.

Frontline Commando (Glu Games)

#1 first person shooting game on Android, by Glu Games.

Heyzap

Social game discovery app with millions of users

Pose

Pose is the #1 fashion app for sharing and discovering new styles

Thousands more apps…

Async HTTP is used in production by thousands of top apps.

Installation & Basic Usage

Add maven dependency using Gradle buildscript in format

```
dependencies {
  compile 'com.loopj.android:android-async-http:1.4.9'
}
```

Import the http package.

```
import com.loopj.android.http.*;
```

Create a new `AsyncHttpClient` instance and make a request:

```
AsyncHttpClient client = new AsyncHttpClient();
client.get("https://www.google.com", new AsyncHttpResponseHandler() {

    @Override
    public void onStart() {
        // called before request is started
    }

    @Override
    public void onSuccess(int statusCode, Header[] headers, byte[] response) {
        // called when response HTTP status is "200 OK"
```

```
    }

    @Override
    public void onFailure(int statusCode, Header[] headers, byte[] errorResponse, Throwable e) {
        // called when response HTTP status is "4XX" (eg. 401, 403, 404)
    }

    @Override
    public void onRetry(int retryNo) {
        // called when request is retried
        }
});
```

Recommended Usage: Make a Static Http Client

In this example, we'll make a http client class with static accessors to make it easy to communicate with Twitter's API.

```
import com.loopj.android.http.*;

public class TwitterRestClient {
  private static final String BASE_URL = "https://api.twitter.com/1/";

  private static AsyncHttpClient client = new AsyncHttpClient();

  public static void get(String url, RequestParams params, AsyncHttpResponseHandler responseHand
      client.get(getAbsoluteUrl(url), params, responseHandler);
  }

  public static void post(String url, RequestParams params, AsyncHttpResponseHandler responseHan
      client.post(getAbsoluteUrl(url), params, responseHandler);
  }

  private static String getAbsoluteUrl(String relativeUrl) {
      return BASE_URL + relativeUrl;
  }
}
```

This then makes it very easy to work with the Twitter API in your code:

```
import org.json.*;
import com.loopj.android.http.*;

class TwitterRestClientUsage {
    public void getPublicTimeline() throws JSONException {
        TwitterRestClient.get("statuses/public_timeline.json", null, new JsonHttpResponseHandler
            @Override
            public void onSuccess(int statusCode, Header[] headers, JSONObject response) {
                // If the response is JSONObject instead of expected JSONArray
            }

            @Override
            public void onSuccess(int statusCode, Header[] headers, JSONArray timeline) {
                // Pull out the first event on the public timeline
                JSONObject firstEvent = timeline.get(0);
                String tweetText = firstEvent.getString("text");

                // Do something with the response
                System.out.println(tweetText);
            }
        });
    }
}
```

Check out the AsyncHttpClient, RequestParams and AsyncHttpResponseHandler Javadocs for more details.

Persistent Cookie Storage with `PersistentCookieStore`

This library also includes a `PersistentCookieStore` which is an implementation of the Apache HttpClient `CookieStore` interface that automatically saves cookies to `SharedPreferences` storage on the Android device.

This is extremely useful if you want to use cookies to manage authentication sessions, since the user will remain logged in even after closing and re-opening your app.

First, create an instance of `AsyncHttpClient`:

```
AsyncHttpClient myClient = new AsyncHttpClient();
```

Now set this client's cookie store to be a new instance of `PersistentCookieStore`, constructed with an activity or application context (usually `this` will suffice):

```
PersistentCookieStore myCookieStore = new PersistentCookieStore(this);
myClient.setCookieStore(myCookieStore);
```

Any cookies received from servers will now be stored in the persistent cookie store.

To add your own cookies to the store, simply construct a new cookie and call `addCookie`:

```
BasicClientCookie newCookie = new BasicClientCookie("cookiesare", "awesome");
newCookie.setVersion(1);
newCookie.setDomain("mydomain.com");
newCookie.setPath("/");
myCookieStore.addCookie(newCookie);
```

See the PersistentCookieStore Javadoc for more information.

Adding GET/POST Parameters with `RequestParams`

The `RequestParams` class is used to add optional GET or POST parameters to your requests. `RequestParams` can be built and constructed in various ways:

Create empty `RequestParams` and immediately add some parameters:

```
RequestParams params = new RequestParams();
params.put("key", "value");
params.put("more", "data");
```

Create `RequestParams` for a single parameter:

```
RequestParams params = new RequestParams("single", "value");
```

Create `RequestParams` from an existing `Map` of key/value strings:

```
HashMap<String, String> paramMap = new HashMap<String, String>();
paramMap.put("key", "value");
RequestParams params = new RequestParams(paramMap);
```

See the RequestParams Javadoc for more information.

Uploading Files with `RequestParams`

The `RequestParams` class additionally supports multipart file uploads as follows:

Add an `InputStream` to the `RequestParams` to upload:

```
InputStream myInputStream = blah;
RequestParams params = new RequestParams();
params.put("secret_passwords", myInputStream, "passwords.txt");
```

Add a `File` object to the `RequestParams` to upload:

```
File myFile = new File("/path/to/file.png");
RequestParams params = new RequestParams();
try {
    params.put("profile_picture", myFile);
} catch(FileNotFoundException e) {}
```

Add a byte array to the `RequestParams` to upload:

```
byte[] myByteArray = blah;
RequestParams params = new RequestParams();
params.put("soundtrack", new ByteArrayInputStream(myByteArray), "she-wolf.mp3");
```

See the RequestParams Javadoc for more information.

Downloading Binary Data with `FileAsyncHttpResponseHandler`

The `FileAsyncHttpResponseHandler` class can be used to fetch binary data such as images and other files. For example:

```
AsyncHttpClient client = new AsyncHttpClient();
client.get("https://example.com/file.png", new FileAsyncHttpResponseHandler(/* Context */ this)
    @Override
    public void onSuccess(int statusCode, Header[] headers, File response) {
        // Do something with the file `response`
    }
});
```

See the FileAsyncHttpResponseHandler Javadoc for more information.

Adding HTTP Basic Auth credentials

Some requests may need username/password credentials when dealing with API services that use HTTP Basic Access Authentication requests. You can use the method `setBasicAuth()` to provide your credentials.

Set username/password for any host and realm for a particular request. By default the Authentication Scope is for any host, port and realm.

```
AsyncHttpClient client = new AsyncHttpClient();
client.setBasicAuth("username","password/token");
client.get("https://example.com");
```

You can also provide a more specific Authentication Scope (recommended)

```
AsyncHttpClient client = new AsyncHttpClient();
client.setBasicAuth("username","password", new AuthScope("example.com", 80, AuthScope.ANY_REALM));
client.get("https://example.com");
```

See the RequestParams Javadoc for more information.

Testing on device

You can test the library on real device or emulator using provided Sample Application. Sample application implements all important functions of library, you can use it as source of inspiration.

Source code of sample application: https://github.com/loopj/android-async-http/tree/master/sample

To run sample application, clone the android-async-http github repository and run command in it's root:

```
gradle :sample:installDebug
```

Which will install Sample application on connected device, all examples do work immediately, if not please file bug report on https://github.com/loopj/android-async-http/issues

Building from Source

To build a `.jar` file from source, first make a clone of the android-async-http github repository. Then you have to have installed Android SDK and Gradle buildscript, then just run:

```
gradle :library:jarRelease
```

This will generate a file in path `{repository_root}/library/build/libs/library-1.4.9.jar`.

Reporting Bugs or Feature Requests

Please report any bugs or feature requests on the github issues page for this project here:

https://github.com/loopj/android-async-http/issues

Credits & Contributors

James Smith (https://github.com/loopj)

Creator and Maintainer

Marek Sebera (https://github.com/smarek)

Maintainer since 1.4.4 release

Noor Dawod (https://github.com/fineswap)

Maintainer since 1.4.5 release

Luciano Vitti (https://github.com/xAnubiSx)

Collaborated on Sample Application

Jason Choy (https://github.com/jjwchoy)

Added support for RequestHandle feature

Micah Fivecoate (https://github.com/m5)

Major Contributor, including the original `RequestParams`

The Droid Fu Project (https://github.com/kaeppler/droid-fu)

Inspiration and code for better http retries

Rafael Sanches (https://blog.rafaelsanches.com)

Original `SimpleMultipartEntity` code

Anthony Persaud (https://github.com/apersaud)

Added support for HTTP Basic Authentication requests.

Linden Darling (https://github.com/coreform)

Added support for binary/image responses

And many others, contributions are listed in each file in license header. You can also find contributors by looking on project commits, issues and pull-requests on Github

License

The Android Asynchronous Http Client is released under the Android-friendly Apache License, Version 2.0. Read the full license here:

https://www.apache.org/licenses/LICENSE-2.0

About the Author

James Smith, British entrepreneur and developer based in San Francisco.

I'm the co-founder of Bugsnag with Simon Maynard, and from 2009 to 2012 I led up the product team as CTO of Heyzap.

Follow @loopj