This repository Search | Explore Gist Blog Help | happylishang

smanikandan14 / **Volley-demo**

Watch 62 | Unstar 529 | Fork 268

An demonstration of Volley - HTTP library announced by google in I/O 2013. Illustrates, JSONRequest,StringRequest, Image caching.

28 commits | **1** branch | **0** releases | **1** contributor

branch: **master** | **Volley-demo** / +

Removing old files.

mselvaraju authored on 10 Mar | `latest commit 6d1bf5f2df`

gradle/wrapper	Updated the project to work with Studio.	3 months ago
volleydemoapp	Updated the project to work with Studio.	3 months ago
.classpath	Cleaned up dimens.xml. Published to play store.	2 years ago
.project	Added GSON parsing.	2 years ago
README.md	Update README.md	2 years ago
build.gradle	Updated the project to work with Studio.	3 months ago
gradle.properties	Updated the project to work with Studio.	3 months ago
proguard-project.txt	Base project with sample jsonRequest to GoogleSearch api using volley…	2 years ago
project.properties	Removing old files.	3 months ago
settings.gradle	Updated the project to work with Studio.	3 months ago

Code | Issues 3 | Pull requests 1 | Wiki | Pulse | Graphs

HTTPS clone URL

`https://github.com`

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop | Download ZIP

README.md

Volley-demo

An demonstration of Volley - HTTP library announced by google in I/O 2013.

Play Store Link for demo download

https://play.google.com/store/apps/details?id=com.mani.volleydemo

Why Volley?

Android has provided two HTTP Clients *AndroidHttpClient* (Extended from apache HTTPClient) and *HttpUrlConnection* to make a HTTP Request. Both has its own pros and cons. When an application is developed, we write HTTP connection classes which handles all the HTTP requests, creating THREADS to run in background, managing THREAD pool, response parsing, response caching, handling error codes, SSL connections, running requests in parallel and others stuffs around that. Every developer has his own way of implementing these functionalities.Some might use AsycnTask for running network operations in background, or some might use passing handlers created from UI thread to HTTP connection classes which then executes network operation in worker thread and uses the handler to pass back the parsed HTTP response back to the main thread.

But we end up writing same boilerplate codes repeatedly and we try to reinvent the wheel in our application development.

For example, in the below snippet, a HTTP request is made in the AysncTask's *doBackground* method. When the response is obtained, data is copied from HttpUrlConnection's *InputStream* to *OutputStream* and then it tries to convert the string obtained from outputStream to *JSONObject* which is our final response. On the course, all the necessary try, catch block is handled. All these boilerplate codes are repeated throughout our code.

```
HttpURLConnection urlConnection = null;
try {
   URL url = new URL("http://www.android.com/");
   urlConnection = (HttpURLConnection) url.openConnection();
   InputStream in = new BufferedInputStream(urlConnection.getInputStream());
   ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
   byte[] buffer = new byte[1024]; // Adjust if you want
   int bytesRead;
   while ((bytesRead = in.read(buffer)) != -1) {
     outputStream.write(buffer, 0, bytesRead);
   }
   JSONObject resultJSON = new JSONObject(outputStream.toString());

}catch (Exception e) {
       e.printStackTrace();
} finally {
     urlConnection.disconnect();
}
```

Google has come up with Volley interface which helps developers to handle all the network related operations so that developers can concentrate implementing the business logic after the HTTP response is obtained.Also having less code for network calls helps developer reduce number of bugs.

NOTE Volley is not good for larger file download/upload operations as well video streaming operations.

Key features of the Volley are below

- Develop Super fast networked applications for android.
- Schedules all your HTTP requests running them parallely in background threads and manages those threads.
- Gives you flexible ways to run your networking requests concurrently with synchronization.
- Comes with inbuilt JSON parsing the response.
- Set prioirty for requests.
- Retry policy for timeout,certain ERROR codes as Internal Server error.
- Flexible Request cancellations.
- Memory & Disk Caching for images.Batch dispatch to Image Downloads.
- Flexible in giving your own cache implementations.
- You can include your own HTTPStack (to handle SSL connections, PATCH requests).
- Effective inbuilt cache - control to handle response caching.
- Request tracing for debugging.
- Excels in the way responses are given back to you.

Integrating Volley to your project.

You can include in two ways

- Create *Volley.jar* and include as *jar dependency* to your project.
- Include the volley project as *Library Dependency* in your project.

Clone the Volley project from below git repo.
https://android.googlesource.com/platform/frameworks/volley/)

1. Creating Volley.jar

   - Import the project into eclipse.
   - $ cd volley
   - $ android update project -p . (Generate local.properties file)
   - $ ant jar
   - Right click on build.xml file and ‘Run as Ant Task’ , volley.jar would be created in /bin folder.

2. Library Dependency
    - Edit the project.properties file and the add the below line
    - android.library=true
    - Now right click on your project--> Properties--> Android --> Under Library section, choose ‘Add’ and select ‘Volley’ project as library dependency to your project.

Using Volley involves two main classes **RequestQueue** and **Request**.

- RequestQueue - Dispatch Queue which takes a Request and executes in a worker thread or if cache found its takes from cache and responds back to the UI main thread.
- Request - All network(HTTP) requests are created from this class. It takes main parameters required for a HTTP request like
    - METHOD Type - GET, POST, PUT, DELETE
    - URL
    - Request data (HTTP Body)
    - Successful Response Listener
    - Error Listener

Volley Provides two specific implementations of **Request**.

- JsonObjectRequest
- StringRequest

Initialise RequestQueue

```
mVolleyQueue = Volley.newRequestQueue(this);
```

You can create a instance of RequestQueue by passing any *Object* to the static method *newRequestQueue* of Volley Class. In this case, activity instance *this* is passed to create the instance. Similarly while cancelling all the requests dispatched in this RequestQueue we should be using activity instance to cancel the requests.

JsonObjectRequest

Creates HTTP request which helps in connecting to JSON based response API's. Takes parameters as

- HTTP METHOD
- A JSON object as request body (mostly for POST & PUT api's)
- Success & Error Listener.

Volley parses the HTTP Response to a JSONObject for you. If your server api's are JSON based, you can straightway go ahead and use *JsonObjectRequest*

> The **Content-Type** for this request is always set to *application/json*

```
String url = "<SERVER_API>";

JsonObjectRequest jsonObjRequest = new JsonObjectRequest(Request.Method.GET,
                            url, null,
                            new Response.Listener<JSONObject>() {
    @Override
    public void onResponse(JSONObject response) {
    }
}, new Response.ErrorListener() {

    @Override
    public void onErrorResponse(VolleyError error) {
    }
});
```

```
mVolleyQueue.add(jsonObjRequest);
```

StringRequest

To obtain the HTTP Response as a String, create HTTP Request using *StringRequest* Takes parameters as

- HTTP METHOD
- Success & Error Listener.

```
String url = "<SERVER-API>";

StringRequest stringRequest = new StringRequest(Request.Method.GET, url, new Response.Listener<S
    @Override
    public void onResponse(String response) {
    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
    }
});

mVolleyQueue.add(stringRequest);
```

GsonRequest

You can customize the *Request* to make a new type of Request which can give the response in Java Class Object. GSON is a library which is used in converting JSON to Java Class Objects and vice-versa. You write custom request which takes Java Class name as a parameter and return the response in that Class Object.

You should include **gson.jar** as JAR dependency in your project.

```
public class GsonRequest<T> extends Request<T>{
    private Gson mGson;

    public GsonRequest(int method, String url, Class<T> cls, String requestBody, Listener<T> lis
        ErrorListener errorListener) {
        super(method, url, errorListener);
        mGson = new Gson();
    }

    @Override
    protected Response<T> parseNetworkResponse(NetworkResponse response) {
        try {
            String jsonString = new String(response.data, HttpHeaderParser.parseCharset(response
                T parsedGSON = mGson.fromJson(jsonString, mJavaClass);
            return Response.success(parsedGSON,
                    HttpHeaderParser.parseCacheHeaders(response));

        } catch (UnsupportedEncodingException e) {
            return Response.error(new ParseError(e));
        } catch (JsonSyntaxException je) {
            return Response.error(new ParseError(je));
        }
    }
}
```

Using GsonRequest

- Takes *FlickrResponsePhotos* class as parameter and returns the response as

FlickrResponsePhotos Object.
- Makes life easier for developers if you have the response in your desired Class objects.

```
gsonObjRequest = new GsonRequest<FlickrResponsePhotos>(Request.Method.GET, url,
        FlickrResponsePhotos.class, null, new Response.Listener<FlickrResponsePhotos>() {
    @Override
    public void onResponse(FlickrResponsePhotos response) {
    }
}, new Response.ErrorListener() {

    @Override
    public void onErrorResponse(VolleyError error) {
    }
});
mVolleyQueue.add(gsonObjRequest);
```

Image Download

Most common operation in an application is *Image* download operation. Volley provides different ways of downloading a image. Volley also provides transparent caching for images.flexible *Cache* implementations for your

- ImageRequest
  Just like other *Request* types, ImageRequest takes a *URL* as paramater and returns *Bitmap* as response on the main threa.

```
ImageRequest imgRequest = new ImageRequest(<URL>, new Response.Listener<Bitmap>() {
        @Override
        public void onResponse(Bitmap response) {
            mImageView.setImageBitmap(response);
        }
    }, 0, 0, Bitmap.Config.ARGB_8888, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            mImageView.setImageResource(R.drawable.error);
        }
    });
mVolleyQueue.add(imgRequest);
```

- ImageDownloader
  Handles loading and caching of images from a URL.Takes *URL*, *ImageView* and *ImageListener* as parameters.Need to initialies the ImageLoader with a cache. Cache can be of two types *MemoryCache* (or) *DiskCache*. Volley provides a DiskCache implementation *DiskBasedCache*.You can always use that. If you need to provide your own cache implementation for ImageDownLoader you need to implement the interface *ImageCache*.

```
//Initialising ImageDownloader
int max_cache_size = 1000000;
mImageLoader = new ImageLoader(mVolleyQueue, new DiskBitmapCache(getCacheDir(),max_cache_size))

( or )
//Memorycache is always faster than DiskCache. Check it our for yourself.
//mImageLoader = new ImageLoader(mVolleyQueue, new BitmapCache(max_cache_size));

mImageLoader.get(<URL>,
        ImageLoader.getImageListener(mImageView,
                R.drawable.flickr,
                android.R.drawable.ic_dialog_alert),
                //You can optional specify width & height of the bitmap to be scaled down wl
                50,50);
```

- NetworkImageView

Downloads the image as well as cancels the image request if the *ImageView* for which it was asked to download is recycled or no longer exists. So reduces the work of developer to manage the *ImageRequest* life cycle. Takes *URL* and *ImageDownloader*

```
mNetworkImageView.setImageUrl(testUrlToDownloadImage1, mImageLoader);
```

SSL connections

Volley doesnt support secured connection to HTTP.To connect to a secured HTTP connection, you need to provide to your own *HttpStack*.
Download httpclient components from below link

- http://hc.apache.org/downloads.cgi
- Copy these two jar files into your projects *libs* folder. **httpclient-4.2.5,httpmime-4.2.5**

httpmime is used for MultipartRequest.

You can set your SSL socket factory into the scheme and set it in *httpClient* to execute your request in Secured Layer. This is a detailed topic of dealing with SSL for self signed certificates. I am not getting detailed into that. But the demo has every code explaining how it is done.

```
    SslHttpStack implements HtpStack

    @Override
    public HttpResponse performRequest(Request<?> request, Map<String, String> additionalHeader
            throws IOException, AuthFailureError {
        HttpUriRequest httpRequest = createHttpRequest(request, additionalHeaders);
    -----------------------
    -----------------------

    /* Register schemes, HTTP and HTTPS */
        SchemeRegistry registry = new SchemeRegistry();
        registry.register(new Scheme("http", new PlainSocketFactory(), 80));
        registry.register(new Scheme("https", new EasySSLSocketFactory(mIsConnectingToYourServe

        /* Make a thread safe connection manager for the client */
        ThreadSafeClientConnManager manager = new ThreadSafeClientConnManager(httpParams, regis
        HttpClient httpClient = new DefaultHttpClient(manager, httpParams);
    }
```

For more detail check the link i have provided in credits section.

Multipart Request

You application might require file/picture uploading to your server. In that case most preferred way is using *Multipart* Request. Volley doesnt provide support for Multipart. But you can still use volley's framework and provide you own implementation of *HttpStack* just like we used for SSL connections.

```
public class MultiPartRequest extends JsonRequest<JSONObject> {

    /* To hold the parameter name and the File to upload */
    private Map<String,File> fileUploads = new HashMap<String,File>();

    /* To hold the parameter name and the string content to upload */
    private Map<String,String> stringUploads = new HashMap<String,String>();

    public void addFileUpload(String param,File file) {
        fileUploads.put(param,file);
    }

    public void addStringUpload(String param,String content) {
        stringUploads.put(param,content);
    }

    public Map<String,File> getFileUploads() {
        return fileUploads;
    }

    public Map<String,String> getStringUploads() {
        return stringUploads;
    }

}
```

And in your HttpStack implementation, create *MultiPartEntity* and set it to HttpRequest. Refer *SslHttpStack createHttpRequest* method for more details.

```
    private static void setMultiPartBody(HttpEntityEnclosingRequestBase httpRequest,
            Request<?> request) throws AuthFailureError {

        // Return if Request is not MultiPartRequest
        if(request instanceof MultiPartRequest == false) {
            return;
        }

        MultipartEntity multipartEntity = new MultipartEntity(HttpMultipartMode.BROWSER_COMPATI
        //Iterate the fileUploads
        Map<String,File> fileUpload = ((MultiPartRequest)request).getFileUploads();
        for (Map.Entry<String, File> entry : fileUpload.entrySet()) {
            multipartEntity.addPart(((String)entry.getKey()), new FileBody((File)entry.getValue
        }

        //Iterate the stringUploads
        Map<String,String> stringUpload = ((MultiPartRequest)request).getStringUploads();
        for (Map.Entry<String, String> entry : stringUpload.entrySet()) {
            try {
                multipartEntity.addPart(((String)entry.getKey()), new StringBody((String)entry.
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        httpRequest.setEntity(multipartEntity);
    }
```

Adding Headers

By default Volley adds *Content-Type* parameter in all Request Header.

- JsonRequest --> *application/json*
- StringRequest --> *application/x-www-form-urlencoded*
- Request --> *application/x-www-form-urlencoded*
- If you wanted to change this behavior, you need to override

```
public String getBodyContentType() {
    return "application/x-www-form-urlencoded; charset=UTF-8";
}
```

To add additional headers to the requests, you need to extend from the existing Requests type and implement the *getHeaders()* method

```
public class MyStringRequest extends StringRequest{
    private Map<String, String> headers = new HashMap<String, String>();
    @Override
    public Map<String, String> getHeaders() throws AuthFailureError {
        return headers;
    }

    public void setHeader(String title, String content) {
        headers.put(title, content);
    }
}
```

Handling Error Codes

Volley checks the HTTP error codes and throws different Error types accordingly. All Error Types are extended from *VolleyError* Other than 200 (OK) & 204 (NO_MODIFIED) Volley treats other HTTP status codes as ERROR. The available Errors are

- TimeoutError -- ConnectionTimeout or SocketTimeout
- AuthFailureError -- 401 (UNAUTHORIZED) && 403 (FORBIDDEN)
- ServerError -- 5xx
- ClientError -- 4xx(Created in this demo for handling all 4xx error which are treated as Client side errors)
- NetworkError -- No network found
- ParseError -- Error while converting HTTP Response to JSONObject.

```
@Override
public void onErrorResponse(VolleyError error) {
    // Handle your error types accordingly.For Timeout & No connection error, you can show 'ret
    // For AuthFailure, you can re login with user credentials.
    // For ClientError, 400 & 401, Errors happening on client side when sending api request.
    // In this case you can check how client is forming the api and debug accordingly.
    // For ServerError 5xx, you can do retry or handle accordingly.
    if( error instanceof NetworkError) {
    } else if( error instanceof ClientError) {
    } else if( error instanceof ServerError) {
    } else if( error instanceof AuthFailureError) {
    } else if( error instanceof ParseError) {
    } else if( error instanceof NoConnectionError) {
    } else if( error instanceof TimeoutError) {
    }

}
```

If your server implementation sends custom error codes. You may need to parse the *networkResponse* can be obtained from Error object to handle the agreed internal error codes. For example to handle the below mentioned error code, you can do like below.

> **422; 3001; Invalid parameter for: user_id (not numeric)**

```
@Override
public void onErrorResponse(VolleyError error) {
    hideProgressDialog();
    if(error instanceof AuthFailureError) {
        showAlertError(ResourcesUtil.getString(R.string.login_error_invalid_credentials));
    }else if(error instanceof ClientError) {
        // Convert byte to String.
        String str = null;
        try {
            str = new String(error.networkResponse.data, "UTF8");
            JSONObject errorJson = new JSONObject(str);
            if( errorJson.has(Constants.ERROR)) {
                JSONObject err = errorJson.getJSONObject(Constants.ERROR);
                if(err.has(Constants.ERROR_MESSAGE)) {
                        String errorMsg = errorJson.has(Constants.ERROR_MESSAGE);
                        showAlertError(errorMsg);
                }
                return;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

    } else {
        MessageUtil.showMessage(ResourcesUtil.getString(R.string.error_service), true);
    }
}
```

Request Cancellation

You can cancel a single request or multiple requests or cancel requests under a TAG name. Using any of one approach.

```
request.cancel();

( or )

for( Request<T> req : mRequestList) {
    req.cancel();
}

( or )

volleyQueue.cancelAll(Object);
```

You can probably store all the requests made in a screen into a List and cancel the requests one by one iterating the list. Or Cancel all the requests made by using *VolleyQueue* instance.

You should (must) do cancelling all the requests made in activitie's **onStop()** method. (Except where you want a POST/PUT request to continue though user leaves the screen).

Set PRIORITY to Requests

You can set Priority to your requests. Normally *ImageRequests* are assigned *LOW* priority and other requests like *JsonObjectRequest* and *StringObjectRequest* are set to *NORMAL* priority. To change the priority for different server requests for your needs you should customize the *Request* class and override *setPriority* and *getPriority* methods.

```
Priority priority;
public void setPriority(Priority priority) {
    this.priority = priority;
}

/*
 * If prioirty set use it,else returned NORMAL
 * @see com.android.volley.Request#getPriority()
 */
public Priority getPriority() {
    if( this.priority != null) {
        return priority;
    } else {
        return Priority.NORMAL;
    }
}
```

> jsonRequest.setPriority(Priority.HIGH);

Retry Policy

Volley provides an easy way to implement your RetryPolicy for your requests.
Volley sets default Socket & ConnectionTImeout to **5 secs** for all requests.

RetryPolicy is an interface where you need to implement your logic of how you want to retry a particular request when a timeout happens.

It deals with these three parameters

- Timeout - Specifies Socket Timeout in millis per every retry attempt.
- Number Of Retries - Number of times retry is attempted.
- BackOff Multiplier - A multiplier which is used to determine exponential time set to socket for every retry attempt.

For ex. If RetryPolicy is created with these values

> Timeout - 3000 secs, Num of Attempt - 3, Back Off Multiplier - 2

Attempt 1: time = time + (time * Back Off Multiplier);
time = 3000 + 6000 = 9000
Socket Timeout = time;
Request dispatched with *Socket Timeout* of *9 Secs*

Attempt 2: time = time + (time * Back Off Multiplier);
time = 9000 + 18000 = 27000
Socket Timeout = time;
Request dispatched with *Socket Timeout* of *27 Secs*

Attempt 3:
time = time + (time * Back Off Multiplier);
time = 27000 + 54000 = 91000
Socket Timeout = time;
Request dispatched with *Socket Timeout* of *1min 31 Secs*

So at the end of *Attempt 3* if still Socket Timeout happenes Volley would throw a **TimeoutError** in your UI Error response handler.

```
//Set a retry policy in case of SocketTimeout & ConnectionTimeout Exceptions.
//Volley does retry for you if you have specified the policy.
jsonObjRequest.setRetryPolicy(new DefaultRetryPolicy(5000,
                DefaultRetryPolicy.DEFAULT_MAX_RETRIES,
                DefaultRetryPolicy.DEFAULT_BACKOFF_MULT));
```

Response Caching

Enable response caching to quickly fetch the response from cache, if below api is set to true.

```
request.setShouldCache(true);
```

- Handling response headers **Cache-Control**
  Volley decides whether to cache the response or not, based on response headers obtained. Some of the parameters it looks for are *Cache-control, maxAge, Expires*.

- In demo of stringObjectRequest it uses weather api, the response headers has **Cache-Control: no-cache, must-revalidate**. In this case, even if *setShouldCache()* api is set true for the request, Volley decides not to store the response, because server has sent response headers as *must-revalidate* So storing response doesn't make sense for this api. Some of these intelligences are implemented already in Volley, you need not take the burden of parsing response headers for especially for caching.

```
Sample Response headers for different requests.

curl -i http://api.openweathermap.org/data/2.5/weather?q=London,uk
HTTP/1.1 200 OK
Cache-Control: no-cache, must-revalidate
Content-Type: application/json; charset=utf-8
Date: Mon, 15 Jul 2013 08:10:31 GMT
Pragma: no-cache
Server: nginx
X-Powered-By: OWM
Content-Length: 402
Connection: keep-alive

curl -i http://farm4.static.flickr.com/3792/9109500182_a2721e9a32_t.jpg
HTTP/1.1 200 OK
Date: Sun, 23 Jun 2013 14:32:24 GMT
Content-Type: image/jpeg
Content-Length: 3253
Accept-Ranges: bytes
Cache-Control: max-age=315360000,public
Expires: Fri, 23 Jun 2023 02:09:58 UTC
Last-Modified: Sat, 22 Jun 2013 15:54:32 GMT
```

Enabling DEBUG Logs on adb logcat for Volley

- $adb shell
- $setprop log.tag.Volley VERBOSE
- logcat

Now you can see Volley debug logs shown in terminal. You can test it by launching **Play Store** app which uses Volley.

```
D/Volley  (24482): [1] MarkerLog.finish: (7067 ms) [X] http://api.flickr.com/services/rest?api_
D/Volley  (24482): [1] MarkerLog.finish: (+0   ) [ 1] add-to-queue
D/Volley  (24482): [1] MarkerLog.finish: (+6   ) [9624] cache-queue-take
D/Volley  (24482): [1] MarkerLog.finish: (+7   ) [9624] cache-hit-expired
D/Volley  (24482): [1] MarkerLog.finish: (+0   ) [9625] network-queue-take
D/Volley  (24482): [1] MarkerLog.finish: (+5726) [9625] network-http-complete
D/Volley  (24482): [1] MarkerLog.finish: (+1158) [9625] network-parse-complete
D/Volley  (24482): [1] MarkerLog.finish: (+169 ) [9625] network-cache-written
D/Volley  (24482): [1] MarkerLog.finish: (+1   ) [9625] post-response
**D/Volley  (24482): [1] MarkerLog.finish: (+0   ) [ 1] canceled-at-delivery**
```

The above debug logs shows that response is ignored since user has called cancelled for the request. Volley has decided to ignore the response before parsing the response avoiding wasteful CPU cycles.

Credits

- http://howrobotswork.wordpress.com/2013/06/02/downloading-a-bitmap-asynchronously-with-volley-example/
- http://bon-app-etit.blogspot.in/2013/04/the-dark-side-of-asynctask.html
- http://www.checkupdown.com/status/E304.html
- http://developer.android.com/training/articles/security-ssl.html

License