**Retrofit2 multpart多文件上传详解**

**Apr 2, 2016 |**[**android**](https://chenkaihua.com/categories/android/)**| 159 阅读**

Retrofit2是目前很流行的android网络框架，运用注解和动态代理，极大的简化了网络请求的繁琐步骤，非常适合处理restfull网络请求。在项目中，经常需要上传文件到服务器，有时候是需要上传多个文件。网上文章基本都是单文件上传教程，这篇文章主要讲retrofit的多文件上传实现。

个人觉得有必要深入理解http协议，这样无论使用哪个网络框架，碰到类似这样上传的问题，一眼就能知道问题出在哪里。因此就有必要了解http协议的上传机制。

了解 multipart/form-data

在最初的http协议中，没有定义上传文件的Method，为了实现这个功能，http协议组改造了post请求，添加了一种post规范，设定这种规范的Content-Type为multipart/form-data;boundary=${bound} ,其中${bound}是定义的分隔符，用于分割各项内容(文件,key-value对)，不然服务器无法正确识别各项内容。post body里需要用到,尽量保证随机唯一。

post格式如下:

|  |
| --- |
| --${bound}  Content-Disposition: form-data; name="Filename"    HTTP.pdf  --${bound}  Content-Disposition: form-data; name="file000"; filename="HTTP协议详解.pdf"  Content-Type: application/octet-stream    %PDF-1.5  file content  %%EOF    --${bound}  Content-Disposition: form-data; name="Upload"    Submit Query  --${bound}-- |

${bound}是Content-Type里boundary的值

Retrofit2 对multipart/form-data的封装

Retrofit其实是个网络代理框架，负责封装请求，然后把请求分发给http协议具体实现者-httpclient。retrofit默认的httpclient是okhttp。

既然Retrofit不实现http，为啥还用它呢。因为他方便！！  
Retrofit会根据注解封装网络请求，待httpclient请求完成后，把原始response内容通过转化器(converter)转化成我们需要的对象(object)。

具体怎么使用 retrofit2,请参考: [Retrofit2官网](http://square.github.io/retrofit/)

那么Retrofit和okhttp怎么封装这些multipart/form-data上传数据呢

* 在**retrofit**中:
  + @retrofit2.http.Multipart: 标记一个请求是multipart/form-data类型,需要和@retrofit2.http.POST一同使用，并且方法参数必须是@retrofit2.http.Part注解。
  + @retrofit2.http.Part: 代表Multipart里的一项数据,即用${bound}分隔的内容块。
* 在**okhttp3**中:
  + okhttp3.MultipartBody: multipart/form-data的抽象封装,继承okhttp3.RequestBody
  + okhttp3.MultipartBody.Part: multipart/form-data里的一项数据。

Service接口定义

假设服务器上传接口返回数据类型为application/json,字段如下

|  |
| --- |
| {  data: "上传了3个文件",  msg: "访问成功",  code: 200  } |

因此需要对返回数据封装成一个对象，考虑到复用性,封装成泛型最佳:

|  |
| --- |
| public class BaseResponse<T> {  public int code;  public String msg;  public T data;  } |

接着**定义一个上传的网络请求Service**:

|  |
| --- |
| public interface FileuploadService {  /\*\*  \* 通过 List<MultipartBody.Part> 传入多个part实现多文件上传  \* @param parts 每个part代表一个  \* @return 状态信息  \*/  @Multipart  @POST("users/image")  Call<BaseResponse<String>> uploadFilesWithParts(@Part() List<MultipartBody.Part> parts);  /\*\*  \* 通过 MultipartBody和@body作为参数来上传  \* @param multipartBody MultipartBody包含多个Part  \* @return 状态信息  \*/  @POST("users/image")  Call<BaseResponse<String>> uploadFileWithRequestBody(@Body MultipartBody multipartBody);  } |

由上可知，有两种方式实现上传

* 使用@Multipart注解方法，并用@Part注解方法参数，类型是List<okhttp3.MultipartBody.Part>
* 不使用@Multipart注解方法，直接使用@Body注解方法参数，类型是okhttp3.MultipartBody

可以看到,无论方法参数类型是MultipartBody.Part还是MultipartBody，这些类都不是Retrofit的类，而是okhttp实现上传的源生类。

**为什么可以这样写**：

* Retrofit会判断@Body的参数类型，如果参数类型为okhttp3.RequestBody,则Retrofit不做包装处理，直接丢给okhttp3处理。而MultipartBody是继承RequestBody，因此Retrofit不会自动包装这个对象。
* 同理，Retrofit会判断@Part的参数类型，如果参数类型为okhttp3.MultipartBody.Part,则Retrofit会把RequestBody封装成MultipartBody，再把Part添加到MultipartBody。

上传多个文件

写好service接口后，来看看怎么构造MultipartBody  
可以写一个方法，**用于把File对象转化成MultipartBody:**

|  |
| --- |
| public static MultipartBody filesToMultipartBody(List<File> files) {  MultipartBody.Builder builder = new MultipartBody.Builder();    for (File file : files) {  // TODO: 16-4-2 这里为了简单起见，没有判断file的类型  RequestBody requestBody = RequestBody.create(MediaType.parse("image/png"), file);  builder.addFormDataPart("file", file.getName(), requestBody);  }  builder.setType(MultipartBody.FORM);  MultipartBody multipartBody = builder.build();  return multipartBody;  } |

**或者把File转化成MultipartBody.Part**:

|  |
| --- |
| public static List<MultipartBody.Part> filesToMultipartBodyParts(List<File> files) {  List<MultipartBody.Part> parts = new ArrayList<>(files.size());  for (File file : files) {  // TODO: 16-4-2 这里为了简单起见，没有判断file的类型  RequestBody requestBody = RequestBody.create(MediaType.parse("image/png"), file);  MultipartBody.Part part = MultipartBody.Part.createFormData("file", file.getName(), requestBody);  parts.add(part);  }  return parts;  } |

最后就剩下调用了

**为了复用，因此把构建Retrofit简单封装成一个builder类:**

|  |
| --- |
| public class RetrofitBuilder {  private static Retrofit retrofit;  public synchronized static Retrofit buildRetrofit() {  if (retrofit == null) {  HttpLoggingInterceptor logging = new HttpLoggingInterceptor();  Gson gson = new GsonBuilder().setDateFormat("yyyy-MM-dd HH:mm:ss").create();  GsonConverterFactory gsonConverterFactory = GsonConverterFactory.create(gson);  logging.setLevel(HttpLoggingInterceptor.Level.BODY);  OkHttpClient client = new OkHttpClient.Builder()  .addInterceptor(logging).retryOnConnectionFailure(true)  .build();  retrofit = new Retrofit.Builder().client(client)  .baseUrl(Config.NetURL.BASE\_URL)  .addConverterFactory(gsonConverterFactory)  .addCallAdapterFactory(RxJavaCallAdapterFactory.create())  .build();  }  return retrofit;  }  } |

**接着可以在activity里调用FileUploadService的接口了:**

|  |
| --- |
| private void uploadFile() {  MultipartBody body = MultipartBuilder.filesToMultipartBody(mFileList);  RetrofitBuilder.buildRetrofit().create(FileUploadService.class).uploadFileWithRequestBody(body)  .enqueue(new Callback<BaseResponse<String>>() {  @Override  public void onResponse(Call<BaseResponse<String>> call, Response<BaseResponse<String>> response) {  if (response.isSuccessful()) {  BaseResponse<String> body = response.body();  Toast.makeText(LoginActivity.this, "上传成功:"+response.body().getMsg(), Toast.LENGTH\_SHORT).show();  } else {  Log.d(TAG,"上传失败");  Toast.makeText(RegisterActivity.this, "上传失败", Toast.LENGTH\_SHORT).show();  }  }  @Override  public void onFailure(Call<BaseResponse<String>> call, Throwable t) {  Toast.makeText(RegisterActivity.this, "网络连接失败", Toast.LENGTH\_SHORT).show();  }  });  } |