Retrofit 自身功能的高度定制

##### **Converter**

###### **RequestBodyConverter**

首先看一下原生的文件上传的写法

@Multipart  
@POST("test/upload1")  
Observable<BaseResponse<String>> upload1(@Part MultipartBody.Part file);

private void update1() {  
 //找到项目日志的位置  
 File file = new File(PLog.*PATH*, PLog.*PLOG\_FILE\_NAME*);  
 RequestBody requestFile =  
 RequestBody.*create*(MediaType.*parse*("txt"), file);  
 MultipartBody.Part body =  
 MultipartBody.Part.*createFormData*("aFile", file.getName(), requestFile);  
 RetrofitSingleton.*apiService* .upload1(body)  
 .subscribeOn(Schedulers.*io*())  
 .observeOn(AndroidSchedulers.*mainThread*())  
 .subscribe(new Action1<BaseResponse<String>>() {  
 @Override  
 public void call(BaseResponse<String> s) {  
 PLog.*e*(s.model);  
 }  
 }, new Action1<Throwable>() {  
 @Override  
 public void call(Throwable throwable) {  
 PLog.*e*("发生错误");  
 }  
 });  
  
}

随便看一眼服务器端代码

@ResponseBody

@RequestMapping(value = "/app/test/upload1")

**public** ResponseJson<String> update1(HttpServletRequest request,@RequestParam(value ="aFile",required=**false**)MultipartFile file){

log.error("进入上传方法");

ResponseJson<String> rj = **new** ResponseJson<String>();

String fileName=file.getOriginalFilename();

String uploadContentType =file.getContentType();

**if** (file.getSize()> 1024 \* 1024 \* 50) {

**throw** **new** RuntimeException("文件大小不得大于50M");

}

DateFormat df = **new** SimpleDateFormat("yyyyMMddHHmmss");

fileName = "update\_"+df.format(**new** Date())+fileName;

String uploadPath = request.getSession().getServletContext().getRealPath("/app\_update")+File.*separatorChar*; //设置保存目录

File file1 =**new** File(uploadPath);

//如果文件夹不存在则创建

**if** (!file1.exists() && !file1 .isDirectory())

{

file1.mkdir();

}

**try** {

file.transferTo(**new** File(uploadPath +fileName));

} **catch** (IllegalStateException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

rj.setModel("上传成功" + fileName);

**return** rj;

}

这里app端是将log.txt上传至服务器端。

而我们在请求的时候希望上传文件可以这样写

@Multipart  
@POST("test/upload1")  
Observable<BaseResponse<String>> upload2(@Part("aFile") File file);

服务器端会收到”log.txt”

Retrofit 在发现自己收到的实际入参是个 File 时，不知道该怎么办，情急之下给 toString了，而且还是个 JsonString（后来查证原来是使用了 GsonRequestBodyConverter。。）。

接下来我们就自己实现一个 FileRequestBodyConverter，

static class FileRequestBodyConverterFactory extends Converter.Factory { @Override public Converter<File, RequestBody> requestBodyConverter(Type type, Annotation[] parameterAnnotations, Annotation[] methodAnnotations, Retrofit retrofit) { return new FileRequestBodyConverter(); }} static class FileRequestBodyConverter implements Converter<File, RequestBody> { @Override public RequestBody convert(File file) throws IOException { return RequestBody.create(MediaType.parse("application/otcet-stream"), file); }}

###### **ResponseBodyConverter**

如果请求得到的 Json 字符串与返回值类型不对应，比如：

接口返回的 Json 字符串：

{"err":0, "content":"This is a content.", "message":"OK"}

返回值类型

class Result{  
    int code;//等价于 err  
    String body;//等价于 content  
    String msg;//等价于 message  
}

Gson 就是再牛逼，也只能默默无语俩眼泪了，它哪儿知道字段的映射关系怎么这么任性啊。好，现在让我们自定义一个 Converter 来解决这个问题吧！

static class ArbitraryResponseBodyConverterFactory extends Converter.Factory{ @Override public Converter<ResponseBody, ?> responseBodyConverter(Type type, Annotation[] annotations, Retrofit retrofit) { return super.responseBodyConverter(type, annotations, retrofit); }} static class ArbitraryResponseBodyConverter implements Converter<ResponseBody, Result>{ @Override public Result convert(ResponseBody value) throws IOException { RawResult rawResult = new Gson().fromJson(value.string(), RawResult.class); Result result = new Result(); result.body = rawResult.content; result.code = rawResult.err; result.msg = rawResult.message; return result; }} static class RawResult{ int err; String content; String message;}

当然，别忘了在构造 Retrofit 的时候添加这个 Converter，这样我们就能够愉快的让接口返回 Result 对象了。

注意！！Retrofit 在选择合适的 Converter 时，主要依赖于需要转换的对象类型，在添加 Converter 时，注意 Converter 支持的类型的包含关系以及其顺序。

##### OkHttpClient配置

###### 拦截器

在做单元测试的时候，需要设置拦截器，控制返回值

实现拦截器

private class MockInterceptor implements Interceptor {  
 @Override  
 public Response intercept(Chain chain) throws IOException {  
 // 模拟网络数据  
 String content = path;  
  
 ResponseBody body = ResponseBody.*create*(MediaType.*parse*("application/x-www-form-urlencoded"), content);  
 Response response = new Response.Builder().request(chain.request())  
 .protocol(Protocol.*HTTP\_1\_1*)  
 .code(200)  
 .body(body)  
 .build();  
 return response;  
 }  
}

配置拦截器：

OkHttpClient client = new OkHttpClient.Builder()  
 .addInterceptor(new MockInterceptor())  
 .build();

接口源码

*/\*\*  
 \* Observes, modifies, and potentially short-circuits requests going out and the corresponding  
 \* requests coming back in. Typically interceptors will be used to add, remove, or transform headers  
 \* on the request or response.  
 \*/*public interface Interceptor {  
 Response intercept(Chain chain) throws IOException;  
  
 interface Chain {  
 Request request();  
  
 Response proceed(Request request) throws IOException;  
  
 Connection connection();  
 }  
}

使用拦截器统一处理Http请求头部与尾缀

private Retrofit getRetrofit(final String baseUrl, final boolean isWithToken) {  
 final OkHttpClient.Builder httpClient = new OkHttpClient.Builder();  
 httpClient.addInterceptor(new Interceptor() {  
 @Override  
 public Response intercept(Chain chain) throws IOException {  
 Request original = chain.request();  
  
 Request.Builder builder = original.newBuilder()  
 .method(original.method(), original.body())  
 //添加请求头部  
 .header("Content-Type", "application/json")  
 .header("Accept", "application/json")  
 .header("key", "value")  
 .header("key2", "value2");  
 if (isWithToken)  
 builder.header("token", App.get().getPreferencesHelper().getToken());  
  
 Request finalRequest = builder.build();  
 HttpUrl url = finalRequest.url().newBuilder()  
 // 在原链接上添加后缀，相当于在url上添加了 &platform=android&v=1.0  
 .addQueryParameter("platform", "android")  
 .addQueryParameter("v", "1.0")  
 .build();  
  
 Logger.i(url.toString());  
 return chain.proceed(builder.build());  
 }  
 });  
  
 OkHttpClient okHttpClient = httpClient.build();  
  
 Retrofit retrofit = new Retrofit.Builder()  
 .baseUrl(baseUrl)  
 .client(okHttpClient)  
 .addConverterFactory(GsonConverterFactory.*create*())  
 .addCallAdapterFactory(RxJavaCallAdapterFactory.*create*())  
 .build();  
 return retrofit;  
}

Log信息拦截器

OkHttpClient.Builder builder = new OkHttpClient.Builder();

if (BuildConfig.*DEBUG*){  
 // https://drakeet.me/retrofit-2-0-okhttp-3-0-config  
 HttpLoggingInterceptor loggingInterceptor = new HttpLoggingInterceptor();  
 loggingInterceptor.setLevel(HttpLoggingInterceptor.Level.*BASIC*);  
 builder.addInterceptor(loggingInterceptor);  
}

缓存机制

File cacheFile = new File(BaseApplication.*getInstance*().getExternalCacheDir(), "FrHereCache");  
Cache cache = new Cache(cacheFile, 1024 \* 1024 \* 50);  
Interceptor cacheInterceptor = new Interceptor() {  
 @Override  
 public Response intercept(Chain chain) throws IOException {  
 Request request = chain.request();  
 if (!NetUtils.*isNetworkConnected*(BaseApplication.*getInstance*())) {  
 request = request.newBuilder()  
 .cacheControl(CacheControl.*FORCE\_CACHE*)  
 .build();  
 }  
 Response response = chain.proceed(request);  
 if (NetUtils.*isNetworkConnected*(BaseApplication.*getInstance*())) {  
 int maxAge = 0;  
 // 有网络时 设置缓存超时时间0个小时  
 response.newBuilder()  
 .header("Cache-Control", "public, max-age=" + maxAge)  
 .build();  
 } else {  
 // 无网络时，设置超时为4周  
 int maxStale = 60 \* 60 \* 24 \* 28;  
 response.newBuilder()  
 .header("Cache-Control", "public, only-if-cached, max-stale=" + maxStale)  
 .build();  
 }  
 return response;  
 }  
};  
builder.cache(cache).addInterceptor(cacheInterceptor);

###### 设置cookie

服务端可能需要保持请求是同一个cookie，主要看各自需求  
1、app/build.gradle

compile 'com.squareup.okhttp3:okhttp-urlconnection:3.2.0'

2、设置cookie

CookieManager cookieManager = new CookieManager();

cookieManager.setCookiePolicy(CookiePolicy.ACCEPT\_ALL);

builder.cookieJar(new JavaNetCookieJar(cookieManager));

###### 设置超时和重连

希望超时时能重连

//设置超时

builder.connectTimeout(15, TimeUnit.SECONDS);

builder.readTimeout(20, TimeUnit.SECONDS);

builder.writeTimeout(20, TimeUnit.SECONDS);

//错误重连

builder.retryOnConnectionFailure(true);

##### 特殊的参数格式

###### 如果参数是json格式，如：

{

"apiInfo": {

"apiName": "WuXiaolong",

"apiKey": "666"

}

}

建立Bean

public class ApiInfo {

private ApiInfoBean apiInfo

public ApiInfoBean getApiInfo() {

return apiInfo;

}

public void setApiInfo(ApiInfoBean apiInfo) {

this.apiInfo = apiInfo;

}

public class ApiInfoBean {

private String apiName;

private String apiKey;

//省略get和set方法

}

}

ApiStores

@POST("client/shipper/getCarType")

Call<ResponseBody> getData(@Body ApiInfo apiInfo);

代码调用

ApiInfo apiInfo = new ApiInfo();

ApiInfo.ApiInfoBean apiInfoBean = apiInfo.new ApiInfoBean();

apiInfoBean.setApiKey("666");

apiInfoBean.setApiName("WuXiaolong");

apiInfo.setApiInfo(apiInfoBean);

//调接口

getData(apiInfo);

###### 传数组

@GET("v1/enterprise/find")

Call<ResponseBody> getData(@Query("id") String id, @Query("linked[]") String... linked);

代码调用

String id="shli"；

String[] s = new String[]{"shli"};

//调接口

getData(id, s);

###### 传文件-多个

@Multipart  
@POST("test/upload2")  
Observable<BaseResponse<String>> upload2(@Part("pictureName") RequestBody pictureName, @PartMap Map<String, RequestBody> params);

代码调用

//找到项目日志的位置  
 File file = new File(PLog.*PATH*, PLog.*PLOG\_FILE\_NAME*);  
 File file1 = new File(C.*APPIMAGE*, "11.png");  
 RequestBody pictureNameBody = RequestBody.*create*(MediaType.*parse*("application/x-www-form-urlencoded"), "This is a description");  
  
 RequestBody requestFile = RequestBody.*create*(MediaType.*parse*("application/x-www-form-urlencoded"), file);  
 RequestBody requestFile1 = RequestBody.*create*(MediaType.*parse*("application/x-www-form-urlencoded"), file1);  
  
 Map<String, RequestBody> params = new HashMap<>();  
 params.put("txt\"; filename=\"" + file.getName() + "", requestFile);  
 params.put("png\"; filename=\"" + file1.getName() + "", requestFile1);  
  
 RetrofitSingleton.*apiService* .upload2(pictureNameBody,params)  
 .subscribeOn(Schedulers.*io*())  
 .observeOn(AndroidSchedulers.*mainThread*())  
 .subscribe(new Action1<BaseResponse<String>>() {  
 @Override  
 public void call(BaseResponse<String> s) {  
 PLog.*e*(s.model);  
 }  
 }, new Action1<Throwable>() {  
 @Override  
 public void call(Throwable throwable) {  
 PLog.*e*(throwable.getMessage());  
 }  
 });  
  
}

看一眼服务器端

@ResponseBody

@RequestMapping(value = "/app/test/upload2")

**public** ResponseJson<String> update2(HttpServletRequest request){

log.error("进入上传方法");

String fileName = "";

//创建一个通用的多部分解析器

CommonsMultipartResolver multipartResolver = **new** CommonsMultipartResolver(request.getSession().getServletContext());

ResponseJson<String> rj = **new** ResponseJson<String>();

//判断 request 是否有文件上传,即多部分请求

**if**(multipartResolver.isMultipart(request)){

//转换成多部分request

MultipartHttpServletRequest multiRequest = (MultipartHttpServletRequest)request;

//取得request中的所有文件名

Iterator<String> iter = multiRequest.getFileNames();

String uploadPath = request.getSession().getServletContext().getRealPath("/app\_update")+File.*separatorChar*; //设置保存目录

**while**(iter.hasNext()){

//记录上传过程起始时的时间，用来计算上传时间

**int** pre = (**int**) System.*currentTimeMillis*();

//取得上传文件

MultipartFile file = multiRequest.getFile(iter.next());

**if**(file != **null**){

//取得当前上传文件的文件名称

String myFileName = file.getOriginalFilename();

//如果名称不为“”,说明该文件存在，否则说明该文件不存在

DateFormat df = **new** SimpleDateFormat("yyyyMMddHHmmss");

fileName = "update\_"+df.format(**new** Date())+myFileName;

File file1 = **new** File(uploadPath);

//如果文件夹不存在则创建

**if** (!file1.exists() && !file1 .isDirectory())

{

file1.mkdir();

}

**try** {

file.transferTo(**new** File(uploadPath +fileName));

} **catch** (IllegalStateException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

rj.setModel("上传成功"+ fileName);

**return** rj;

}

#### 总结

这次对Retrofit和RxJava联合使用，以及在项目中的封装都没有再次介绍。因为这方面在上周基本已经完成了。并且已经应用在项目中了。也有相关的文档。