链滴

Android OkHttp + Retrofit 下载文件与进度监听

作者：RustFisher

原文链接：https://ld246.com/article/1571462373336

来源网站：链滴

本文链接

下载文件是一个比较常见的需求。给定一个url，我们可以使用URLConnection下载文件。

使用OkHttp也可以通过流来下载文件。

给OkHttp中添加拦截器，即可实现下载进度的监听功能。

使用流来实现下载文件

代码可以参考：https://github.com/RustFisher/android-Basic4/tree/master/appdowloadsample

获取并使用字节流，需要注意两个要点，一个是服务接口方法的 @Streaming 注解，另一个是获取到esponseBody。

获取流（Stream）。先定义一个服务ApiService。给方法添加上@Streaming的注解。

```
private interface ApiService {
    @Streaming
    @GET
    Observable<ResponseBody> download(@Url String url);
}
```

初始化OkHttp。记得填入你的baseUrl。

```
OkHttpClient okHttpClient = new OkHttpClient.Builder()
        .connectTimeout(8, TimeUnit.SECONDS)
        .build();

retrofit = new Retrofit.Builder()
        .client(okHttpClient)
        .addCallAdapterFactory(RxJava2CallAdapterFactory.create())
        .baseUrl("https://yourbaseurl.com")
        .build();
```

发起网络请求。获取到ResponseBody。

```
String downUrl = "xxx.com/aaa.apk";
retrofit.create(ApiService.class)
        .download(downUrl)
        .subscribeOn(Schedulers.io())
        .observeOn(Schedulers.io())
        .doOnNext(new Consumer<ResponseBody>() {
            @Override
            public void accept(ResponseBody responseBody) throws Exception {
                // 处理 ResponseBody 中的流
            }
        })
        .doOnError(new Consumer<Throwable>() {
            @Override
            public void accept(Throwable throwable) throws Exception {
                Log.e(TAG, "accept on error: " + downUrl, throwable);
            }
        })
        .observeOn(AndroidSchedulers.mainThread())
```

```
        .subscribe(new Observer<ResponseBody>() {
            @Override
            public void onSubscribe(Disposable d) {

            }

            @Override
            public void onNext(ResponseBody responseBody) {

            }

            @Override
            public void onError(Throwable e) {
                Log.e(TAG, "Download center retrofit onError: ", e);
            }

            @Override
            public void onComplete() {

            }
        });
```

通过ResponseBody拿到字节流 body.byteStream()。这里会先创建一个临时文件tmpFile，把数据到临时文件里。

下载完成后再重命名成目标文件targetFile。

```
    public void saveFile(ResponseBody body) {
        state = DownloadTaskState.DOWNLOADING;
        byte[] buf = new byte[2048];
        int len;
        FileOutputStream fos = null;
        try {
            Log.d(TAG, "saveFile: body content length: " + body.contentLength());
            srcInputStream = body.byteStream();
            File dir = tmpFile.getParentFile();
            if (dir == null) {
                throw new FileNotFoundException("target file has no dir.");
            }
            if (!dir.exists()) {
                boolean m = dir.mkdirs();
                onInfo("Create dir " + m + ", " + dir);
            }
            File file = tmpFile;
            if (!file.exists()) {
                boolean c = file.createNewFile();
                onInfo("Create new file " + c);
            }
            fos = new FileOutputStream(file);
            long time = System.currentTimeMillis();
            while ((len = srcInputStream.read(buf)) != -1 && !isCancel) {
                fos.write(buf, 0, len);
                int duration = (int) (System.currentTimeMillis() - time);

                int overBytes = len - downloadBytePerMs() * duration;
```

```
            if (overBytes > 0) {
                try {
                    Thread.sleep(overBytes / downloadBytePerMs());
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
            time = System.currentTimeMillis();
            if (isCancel) {
                state = DownloadTaskState.CLOSING;
                srcInputStream.close();
                break;
            }
        }
        if (!isCancel) {
            fos.flush();
            boolean rename = tmpFile.renameTo(targetFile);
            if (rename) {
                setState(DownloadTaskState.DONE);
                onSuccess(url);
            } else {
                setState(DownloadTaskState.ERROR);
                onError(url, new Exception("Rename file fail. " + tmpFile));
            }
        }
    } catch (FileNotFoundException e) {
        Log.e(TAG, "saveFile: FileNotFoundException ", e);
        setState(DownloadTaskState.ERROR);
        onError(url, e);
    } catch (Exception e) {
        Log.e(TAG, "saveFile: IOException ", e);
        setState(DownloadTaskState.ERROR);
        onError(url, e);
    } finally {
        try {
            if (srcInputStream != null) {
                srcInputStream.close();
            }
            if (fos != null) {
                fos.close();
            }
        } catch (IOException e) {
            Log.e(TAG, "saveFile", e);
        }
        if (isCancel) {
            onCancel(url);
        }
    }
}
```

每次读数据的循环，计算读了多少数据和用了多少时间。超过限速后主动sleep一下，达到控制下载度的效果。

要注意不能sleep太久，以免socket关闭。

这里控制的是网络数据流与本地文件的读写速度。

下载进度监听

OkHttp实现下载进度监听，可以从字节流的读写那里入手。也可以使用拦截器，参考官方的例子。

这里用拦截器的方式实现网络下载进度监听功能。

定义回调与网络拦截器

先定义回调。

```
public interface ProgressListener {
    void update(String url, long bytesRead, long contentLength, boolean done);
}
```

自定义ProgressResponseBody。

```
public class ProgressResponseBody extends ResponseBody {

    private final ResponseBody responseBody;
    private final ProgressListener progressListener;
    private BufferedSource bufferedSource;
    private final String url;

    ProgressResponseBody(String url, ResponseBody responseBody, ProgressListener progressL
stener) {
        this.responseBody = responseBody;
        this.progressListener = progressListener;
        this.url = url;
    }

    @Override
    public MediaType contentType() {
        return responseBody.contentType();
    }

    @Override
    public long contentLength() {
        return responseBody.contentLength();
    }

    @Override
    public BufferedSource source() {
        if (bufferedSource == null) {
            bufferedSource = Okio.buffer(source(responseBody.source()));
        }
        return bufferedSource;
    }

    private Source source(final Source source) {
        return new ForwardingSource(source) {
            long totalBytesRead = 0L;
```

```
            @Override
            public long read(Buffer sink, long byteCount) throws IOException {
                long bytesRead = super.read(sink, byteCount);
                // read() returns the number of bytes read, or -1 if this source is exhausted.
                totalBytesRead += bytesRead != -1 ? bytesRead : 0;
                progressListener.update(url, totalBytesRead, responseBody.contentLength(), bytesR
ad == -1);
                return bytesRead;
            }
        };
    }
}
```

定义拦截器。从Response中获取信息。

```
public class ProgressInterceptor implements Interceptor {

    private ProgressListener progressListener;

    public ProgressInterceptor(ProgressListener progressListener) {
        this.progressListener = progressListener;
    }

    @NotNull
    @Override
    public Response intercept(@NotNull Chain chain) throws IOException {
        Response originalResponse = chain.proceed(chain.request());
        return originalResponse.newBuilder()
                .body(new ProgressResponseBody(chain.request().url().url().toString(), originalResp
onse.body(), progressListener))
                .build();
    }
}
```

添加拦截器

在创建OkHttpClient时添加ProgressInterceptor。

```
    OkHttpClient okHttpClient = new OkHttpClient.Builder()
            .connectTimeout(8, TimeUnit.SECONDS)
            .addInterceptor(new ProgressInterceptor(new ProgressListener() {
                @Override
                public void update(String url, long bytesRead, long contentLength, boolean done) {
                    // tellProgress(url, bytesRead, contentLength, done);
                }
            }))
            .build();
```

值得注意的是这里的进度更新非常频繁。并不一定每次回调都要去更新UI。