转载请注明链接: https://blog.csdn.net/feather_wch/article/details/51009871

本文介绍OkHttp的介绍、特点、基本使用、进阶使用(拦截器、Cookie管理)等内容。

如果有帮助的话，请点个赞！万分感谢！

> 友情链接：OkHttp源码详解

OkHttp使用详解

版本：2018/8/26-1(21:20)

介绍(4)

1、OkHttp是什么?

> 1. 一个处理网络请求的开源项目
> 2. Android上最火热的轻量级框架
> 3. 由移动支付Square公司贡献

特点

2、OkHttp的特点

1. 支持同步、异步请求
2. 支持GZIP减少数据流量
3. 缓存响应数据：从而减少重复的网络请求
4. 自动重连：处理了代理服务器问题和SSL握手失败问题
5. 支持SPDY：1.共享同一个Socket来处理同一个服务器的请求。 2.若SPDY不可用，则通过连接池来减少请求延时
6. 请求、处理速度快：基于NIO和Okio。
7. API使用方便简单：需要进一步封装。
8. 能够从许多连接问题中，自动恢复。如：服务器配置了多个IP，第一个IP连接失败后okhttp会自动尝试下一个IP

3、OkHttp的特点在源代码中的体现

1. 异步：enqueue开启异步请求，内部有线程池，通过Dispatcher调度，AsyncCall就是异步请求。
2. 同步：execute开启同步请求，内部会直接调用RealCall的getResponseWithInterceptorChain()进行请求，然后返回。
3. GZIP压缩：在HTTP1.1开始，Web客户端可以通过Acceppt-Encoding头来标识对压缩的支持。HttpEngine.java的readReponse()中无论是使用缓存的Response还是网络的Response，会通过unzip(userResponse)解压缩并且返回。
4. 缓存相应数据：OkHttpClient中Cache对reponse进行缓存，内部采用DiskLruCache实现。
5. 自动重连：RealCall的getResponse中，会进行网络请求。调用HttpEngine的sendRequest或者readResponse时，如果出现RouteException或者IOException，会通过HttpEngine的recover进行恢复。然后继续去请求。
6. 支持SPDY---共享同一个Socket来处理同一个服务器的请求。
7. 复用连接池：ConnectionPool会管理连接池的复用，最大idle连接为5个，keepAlive=5分钟。建立连接时会将connection存储到队列中，再次发起请求时，会去队列寻找host一致的connection并且返回，最后进行请求。

4、OkHttp的应用场景

数据量大的重量级网络请求

基本使用(11)

1、OkHttp集成

okio是okhttp的io基础，因此也需要集成

```
//build.gradle
compile 'com.squareup.okhttp3:okhttp:3.2.0'
compile 'com.squareup.okio:okio:1.7.0'
```

2、Okio是什么?

> square基于IO、NIO的一个高效处理数据流的开源库。

3、NIO是什么?

> 非阻塞性IO

GET请求

4、OkHttp的GET请求

```
//1. 通过Builder构建Request
Request.Builder requestBuilder = new Request.Builder().url("https://www.baidu.com/");
//2. 设定方法=GET
requestBuilder.method("GET", null);
//3. 构建Request
Request request = requestBuilder.build();
//4. 创建OKHttpClient客户端
OkHttpClient okHttpClient = new OkHttpClient();
//5. 创建Call请求
Call call = okHttpClient.newCall(request);
//6. 通过call的enqueue将请求入队(enqueue为异步方法， execute为同步方法)
call.enqueue(new Callback() {
    @Override
    public void onFailure(Call call, IOException e) {
    }
    @Override
    public void onResponse(Call call, Response response) throws IOException {
         //TODO 接收返回值并进行处理
        String str = response.body().string();
        //注意！该回调不在UI线程中
        Log.d("HttpActivity", str);
    }
});
```

> 1. 通过RequestBuilder构造Request
> 2. 创建OkHttpClient
> 3. 通过OkHttpClient的newCall方法，构建Call。
> 4. Call.enqueue/execute 进行异步/同步请求

POST请求

5、OkHttp的POST请求(提交表单)

```
//1. 通过FormBody创建RequestBody
        RequestBody requestBody = new FormBody.Builder()
                .add("ip", getIPAddress(this)) //本机IP
                .build();
        //2. 创建Request
        Request request = new Request.Builder()
                .url("http://ip.taobao.com/service/getIpInfo.php")
                .post(requestBody)
                .build();
        //3. 创建OKHttpClient客户端
        OkHttpClient okHttpClient = new OkHttpClient();
        //4. 创建Call请求
        Call call = okHttpClient.newCall(request);
        //5. 通过call的enqueue将请求入队(enqueue为异步方法， execute为同步方法)
        call.enqueue(new Callback() {
            @Override
            public void onFailure(Call call, IOException e) {
            }

            @Override
            public void onResponse(Call call, Response response) throws IOException {
                //TODO 接收返回值并进行处理
                String str = response.body().string();
                //注意！该回调不在UI线程中
                Log.d("HttpActivity", str);
            }
        });
```

> 获取本机IP的方法（需要网络权限）

```
public static String getIPAddress(Context context) {
    NetworkInfo info = ((ConnectivityManager) context
            .getSystemService(Context.CONNECTIVITY_SERVICE)).getActiveNetworkInfo();
    if (info != null && info.isConnected()) {
        if (info.getType() == ConnectivityManager.TYPE_MOBILE) {//当前使用2G/3G/4G网络
            try {
                //Enumeration<NetworkInterface> en=NetworkInterface.getNetworkInterfaces();
                for (Enumeration<NetworkInterface> en = NetworkInterface.getNetworkInterfac
                    NetworkInterface intf = en.nextElement();
                    for (Enumeration<InetAddress> enumIpAddr = intf.getInetAddresses(); enu
                        InetAddress inetAddress = enumIpAddr.nextElement();
                        if (!inetAddress.isLoopbackAddress() && inetAddress instanceof Inet
                            return inetAddress.getHostAddress();
                        }
                    }
                }
            } catch (SocketException e) {
                e.printStackTrace();
            }

        } else if (info.getType() == ConnectivityManager.TYPE_WIFI) {//当前使用无线网络
            WifiManager wifiManager = (WifiManager) context.getSystemService(Context.WIFI_S
            WifiInfo wifiInfo = wifiManager.getConnectionInfo();
            String ipAddress = intIP2StringIP(wifiInfo.getIpAddress());//得到IPV4地址
            return ipAddress;
        }
    } else {
        //当前无网络连接,请在设置中打开网络
    }
    return null;
}

/**
 * 将得到的int类型的IP转换为String类型
 */
public static String intIP2StringIP(int ip) {
    return (ip & 0xFF) + "." +
            ((ip >> 8) & 0xFF) + "." +
            ((ip >> 16) & 0xFF) + "." +
            (ip >> 24 & 0xFF);
}
```

1. 通过FormBody构造requestBody
2. 通过requestBuilder和requestBody构造Request
3. 创建OkHttpClient
4. 通过OkHttpClient的newCall方法，构建Call。
5. Call.enqueue/execute 进行异步/同步请求

6、OkHttp的POST请求(提交Json)

> 相比于表单方式，只是构造RequestBody的方式不同。

```
//MediaType  设置Content-Type 标头中包含的媒体类型值
RequestBody requestBody = FormBody.create(MediaType.parse("application/json; charset=utf-8")
                , json); //Json
```

上传文件

7、OkHttp上传文件

```
//1. 创建媒体类型
    public static final MediaType MEDIA_TYPE_MARKDOWN = MediaType.parse("text/x-markdown; chars
    public void upload(String url, String filename){
        String filepath = "";
        //2. 获取到SD卡根目录中的文件
        if(Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)){
            filepath = Environment.getExternalStorageDirectory().getAbsolutePath();
        }else{
            return;
        }
        File file = new File(filepath, filename);
        //3. 创建请求(传入需要上传的File)
        Request request = new Request.Builder()
                .url(url) //"https://www.baidu.com"
                .post(RequestBody.create(MEDIA_TYPE_MARKDOWN, file))
                .build();
        //4. 异步上传文件(同步上传需要用execute())
        new OkHttpClient().newCall(request).enqueue(new Callback() {
            @Override
            public void onFailure(Call call, IOException e) {
            }
            @Override
            public void onResponse(Call call, Response response) throws IOException {
                //TODO 接收返回值并进行处理.注意！该回调不在UI线程中
                String str = response.body().string();
                Log.d("HttpActivity", str);
            }
        });
    }
```

异步下载

8、OkHttp异步下载

```
public void download(String url, final String filename){
    //1. 创建请求(目标的url)
    Request request = new Request.Builder()
            .url(url) //"https://.../xxxx.jpg"
            .build();
    //2. 异步下载文件
    new OkHttpClient().newCall(request).enqueue(new Callback() {
        @Override
        public void onFailure(Call call, IOException e) {
        }
        @Override
        public void onResponse(Call call, Response response) throws IOException {
            //3. 得到Response中的流
            InputStream inputStream = response.body().byteStream();
            //4. 创建保存网络数据的文件
            String filepath = "";
            if(Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)){
                filepath = Environment.getExternalStorageDirectory().getAbsolutePath();
            }else{
                filepath = Environment.getDataDirectory().getAbsolutePath();
            }
            File file = new File(filepath, filename);
            //5. 下载数据到文件中
            if(null != file){
                FileOutputStream fileOutputStream = new FileOutputStream(file);
                byte[] buffer = new byte[2048];
                int len = 0;
                while((len = inputStream.read(buffer)) != -1){
                    fileOutputStream.write(buffer, 0, len);
                }
                fileOutputStream.flush();
            }
            Log.d("HttpActivity", "Downloaded");
        }
    });
}
```

多份数据上传

9、OKHttp同时上传多份数据(字符串、图片等等)

```
//1. 创建媒体类型
public static final MediaType MEDIA_TYPE_PNG = MediaType.parse("image/png");
public void multiUpload(String url){
    //2. 请求主体：同时上传字符串数据和图片数据
    RequestBody requestBody = new MultipartBody.Builder()
            .setType(MultipartBody.FORM)
            .addFormDataPart("title", "some jpg") //上传的字符串(key, value)
            .addFormDataPart("image"  //* key值
                    ,"feather.jpg" //* 文件名
                    , RequestBody.create(MEDIA_TYPE_PNG, new File("/sdcard/feather.jpg")) //*
                ) //同时也上传图片
            .build();
    //3. 创建请求
    Request request = new Request.Builder()
            .header("Authorization", "Client-ID" + "...")
            .url(url) //"https://www.baidu.com"
            .post(requestBody)
            .build();
    //4. 异步上传数据和文件(同步上传需要用execute())
    new OkHttpClient().newCall(request).enqueue(new Callback() {
        @Override
        public void onFailure(Call call, IOException e) {
        }
        @Override
        public void onResponse(Call call, Response response) throws IOException {
            //TODO 接收返回值并进行处理.注意！该回调不在UI线程中
            String str = response.body().string();
            Log.d("HttpActivity", str);
        }
    });
}
```

超时时间与缓存

10、OkHttp设置连接、读取、写入的超时时间，以及设置缓存

```
int cacheSize = 10 * 1024 * 1024;
File sdcacheFile = getExternalCacheDir();
//1. 需要通过Builder创建OkHttpClient客户端---才能设置超时和缓存
OkHttpClient.Builder builder = new OkHttpClient.Builder()
        .connectTimeout(15, TimeUnit.SECONDS)
        .writeTimeout(20, TimeUnit.SECONDS)
        .readTimeout(20, TimeUnit.SECONDS)
        .cache(new Cache(sdcacheFile.getAbsoluteFile(), cacheSize));
OkHttpClient okHttpClient = builder.build();
```

取消请求

11、OkHttp如何取消请求

1. `OkHttp3` 的 `Callback` 的回调方法里面有个参数是 `Call` 这个call可以单独取消相应的请求，随便在onFailure或者onResponse方法内部执行 `call.cancel()` 都可以。
2. 如果想 取消所有的请求 ， 则可以 `okHttpClient.dispatcher().cancelAll();` 进行取消。

进阶使用(11)

Interceptors

1、什么是Interceptors拦截器?

1. 是一种强大的机制，可以监视、重写、重试call请求。
2. 通常情况下，用于添加、移除、转换请求和响应的头部信息。

2、最简单拦截器的实现方法

```
//会拦截 请求和返回的数据
public class SignInInterceptor implements Interceptor{
    @Override
    public Response intercept(Chain chain) throws IOException {
        Request request = chain.request();

        long t1 = System.nanoTime();
        Logger.getGlobal().info(String.format("Sending request %s on %s%n%s",
                request.url(), chain.connection(), request.headers()));

        Response response = chain.proceed(request);

        long t2 = System.nanoTime();
        Logger.getGlobal().info(String.format("Received response for %s in %.1fms%n%s",
                response.request().url(), (t2 - t1) / 1e6d, response.headers()));

        return response;
    }
}
```

3、拦截器的分类和在流程上的区别

1. 应用拦截器：拦截OkHttp核心和应用间的请求与响应。不需要关心重定向和重试的中间响应。
2. 网络拦截器：拦截OkHttp核心和网络之前的请求与响应。
   

4、添加应用拦截器: addInterceptor

```
OkHttpClient client = new OkHttpClient.Builder()
    .addInterceptor(new SignInInterceptor())
    .build();
```

5、添加网络拦截器: addNetworkInterceptor

```
OkHttpClient client = new OkHttpClient.Builder()
    .addNetworkInterceptor(new SignInInterceptor())
    .build();
```

6、拦截器的应用场景

> 1. 可以在拦截器中进行请求体压缩(如果Web服务端支持请求体的压缩)。
> 2. 可以将域名替换为IP地址
> 3. 可以在在请求头中添加host属性
> 4. 可以在请求头中添加应用相关的公共参数：设备ID、版本号等

7、应用拦截器是在哪里进行处理的?

> 1. RealCall的getResponseWithInterceptorCahin()中
> 2. 大致流程：从拦截器链中取出拦截器，依次进行递归调用。

```
 //RealCall.java
private Response getResponseWithInterceptorChain(){...}
```

Cookie管理

8、如何持久化Cookie?

> 1. 手动保存Cookie，从Response中取出Header里面的Cookie，保存到本地(SharePreference)。发送请求时利用 `Interceptor拦截器` 添加到头部(保存Cookie也可以采用拦截器拦截，然后本地保存)。为了对应不同域名，可以将域名作为key对Cookie进行保存。
> 2. OkHttp3的Cookie管理。

9、OkHttp3新增的CookieJar

> 1. okhttp3中对Cookie管理提供了额外的支持
> 2. 实现CookieJar接口中的两个方法(发送请求/接到响应)，实现Cookie的内存or本地缓存。

CookieJar的使用

10、CookieJar的实现

> 1-实现

```
public class MyCookieJar implements CookieJar{

    private static HashMap<String, List<Cookie>> mCookieStore = new HashMap<>();

    @Override
    public void saveFromResponse(HttpUrl url, List<Cookie> cookies) {
        mCookieStore.put(url.host(), cookies);
    }

    @Override
    public List<Cookie> loadForRequest(HttpUrl url) {
        List<Cookie> cookies = mCookieStore.get(url.host());
        return cookies != null ? cookies : new ArrayList<Cookie>();
    }
}
```

> 2-使用

```
OkHttpClient client = new OkHttpClient.Builder()
                            .cookieJar(new MyCookieJar())
                            .build();
```

11、CookieJar的saveFromeResponse()在哪里被调用?

> 1. HttpClient.readResponse()->receiveHeaders(networkResponse.headers())
> 2. receiveHeaders()中从response的headers中解析出Cookie

12、CookieJar的loadForRequest()在哪里被调用?

> 1. HttpClient.sendRequest()->networkRequest(): 向request添加headers的时候
> 2. 调用loadForRequest()获取到cookie列表，添加到header中。

```
List<Cookie> cookies = client.cookieJar().loadForRequest(request.url());
if (!cookies.isEmpty()) {
  result.header("Cookie", cookieHeader(cookies));
}
```

参考资料

1. OkHttp拦截器-官方github-wiki
2. OKHTTP结合官网示例分析两种自定义拦截器的区别
3. http中的204和205
4. HTTP协议探索之Cache-Control
5. Http ETag是什么?