目录

[okHttp的工作机制 1](#_Toc32448)

[简单说一下okhttp。 4](#_Toc14181)

[Okhttp 基本实现原理 5](#_Toc27415)

[如何使用OkHttp进行异步网络请求，并根据请求结果刷新UI 5](#_Toc11413)

[OkHttp对于网络请求都有哪些优化，如何实现的 6](#_Toc12813)

[OkHttp框架中都用到了哪些设计模式 6](#_Toc25787)

[Okhttp 网络缓存如何实现？ 6](#_Toc24736)

[Okhttp 网络连接怎么实现复用？ 7](#_Toc22931)

# okHttp的工作机制

Okhttp是http的二次封装，它的工作机制与浏览器非常相似，将http的请求头和请求体封装成header和body，然后组建为专门的请求类Request，这样框架使用者就可以很方便的使用Request类来存储一次网络请求的地址、请求方式、请求头、请求体等数据。

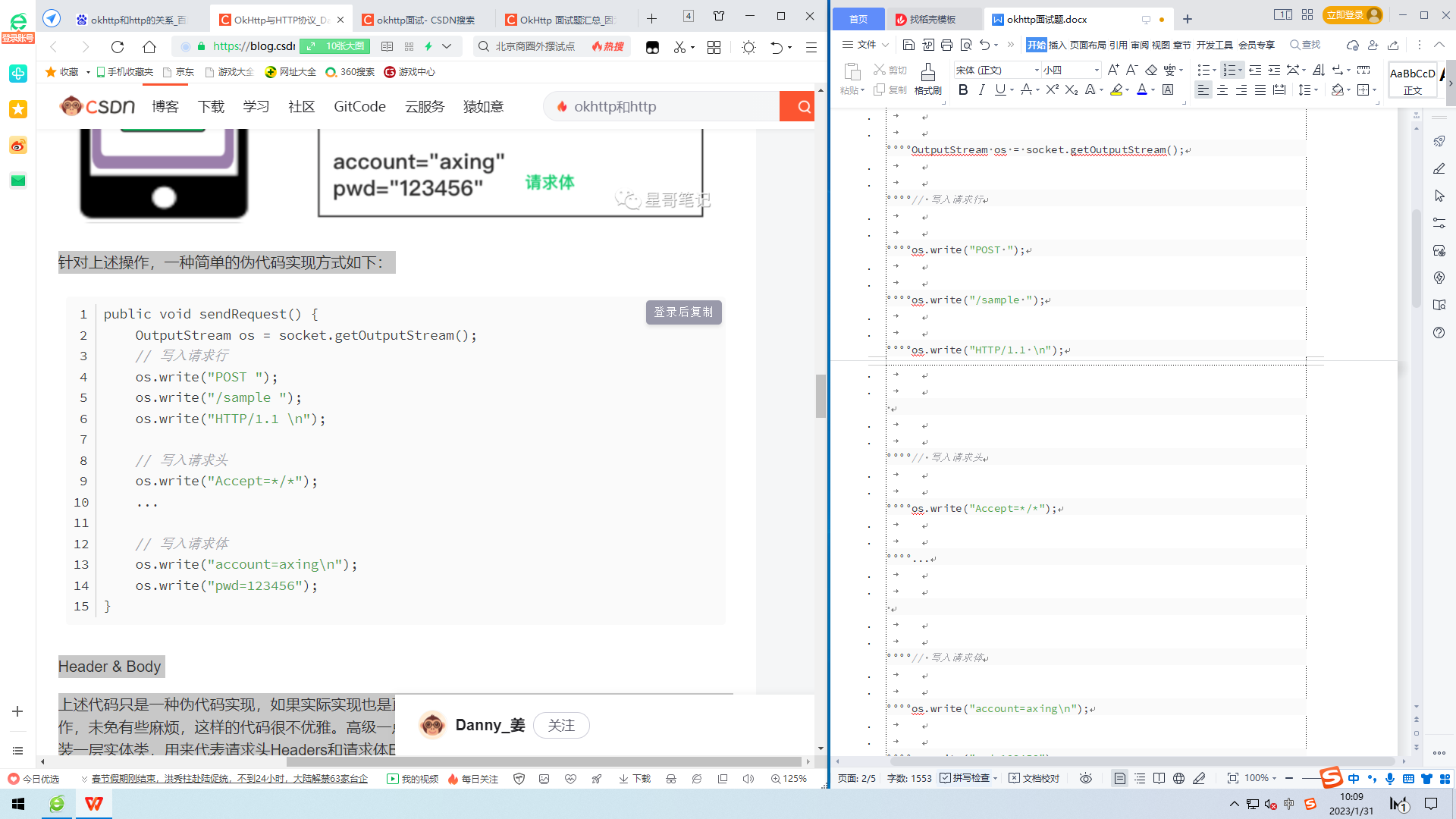
同样，响应消息也有专门的封装类Response。

接着将SocketAPI的操作封装到HttpCodec类中，finishRequest和openResponseBody，分别调用Socket发送请求数据，以及读取响应结果中的信息。

OkHttp的工作机制与浏览器非常相似

同浏览器一样，OkHttp需要将用户的各种操作转化为相对应的http请求对象。比如下图中用户点击了"登录"操作，OkHttp会将其转化为图中右方的请求格式，并发送给服务端。





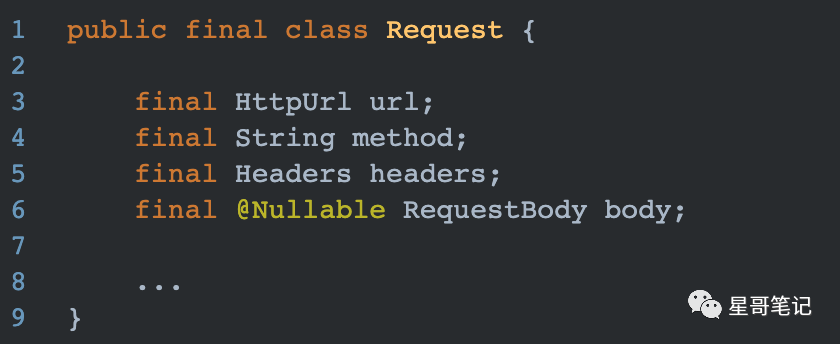
Header & Body

上述代码只是一种伪代码实现，如果实际实现也是直接使用OutputStream依次将所有数据进行写入操作，未免有些麻烦，这样的代码很不优雅。高级一点的工程师立马能想到应该在OutputStream之上封装一层实体类，用来代表请求头Headers和请求体Body。

实际上，在OkHttp中也正是使用了这种方式，Heads和Body如下所示：

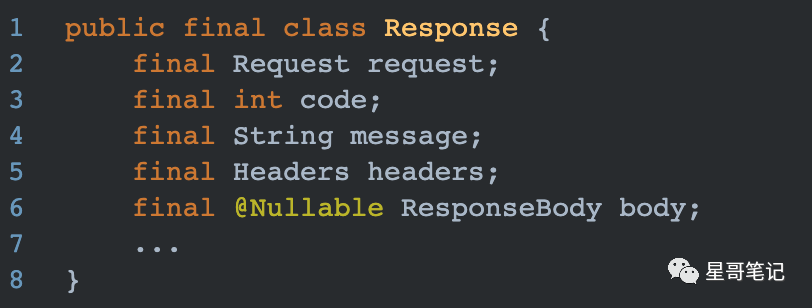
Request & Response

有了Header和Body之后，就可以创建出专门用来发送消息的请求类Request，如下所示：



这样框架使用者就可以很方便的使用Request类来存储一次网络请求的地址、请求方式、请求头、请求体等数据。

同样，对于响应消息也需要一个专门的封装类Response来表示，如下



请求发送和接收

不管是Request还是Response，都需要将数据提交给TCP协议层，并以流的形式在互联网中进行传播。在Java SDK中，有一个专门负责完成这一操作的类Socket，因此我们还需要手动调用Socket API实现网络通信。在OkHttp框架中，这部分操作被封装在HttpCodec类中，finishRequest和openResponseBody，分别代表调用Socket发送请求数据，以及读取响应结果中的信息

# 简单说一下[okhttp](https://so.csdn.net/so/search?q=okhttp&spm=1001.2101.3001.7020)。

1、支持SPDY、HTTP2.0 SPYD：[HTTP](https://baike.baidu.com/item/HTTP?fromModule=lemma_inlink" \t "_blank)协议的增强，包括多路复用、请求优先级以及HTTP报头压缩

2、无缝支持GZIP来减少数据流量 BridgeIntercept

3、支持同步、异步（异步使用较多）

4、缓存响应数据来减少重复的网络请求CacheInterceptor

5、可以从很多常用的连接问题中自动恢复 RetryAndFollowUpInterceptor：重试重定向拦截器

1、支持SPDY、HTTP2.0

基于[TCP](https://baike.baidu.com/item/TCP/33012?fromModule=lemma_inlink" \t "_blank)的会话层协议

SPDY并不是一种用于替代HTTP的协议，而是对[HTTP](https://baike.baidu.com/item/HTTP?fromModule=lemma_inlink" \t "_blank)协议的增强。新协议的功能包括数据流的多路复用、请求优先级以及HTTP报头压缩。

2、无缝支持GZIP来减少数据流量

你发送的数据和接受的收据在传递过程中都是经过gzip压缩的，并且这基本上不需要你手动处理的，框架自动会帮你处理好。

首先，我们给request的请求头添加了("Accept-Encoding", "gzip")的键值对，说明发送的请求数据是经过gzip压缩的；

其次，我们在处理从服务器返回的数据时也会判断：

"gzip".equalsIgnoreCase(networkResponse.header("Content-Encoding")

这两个步骤都是在BridgeIntercept的intercept方法中设置的。

3、支持同步、异步（异步使用较多）  
4、缓存响应数据来减少重复的网络请求

CacheInterceptor.java    
5、可以从很多常用的连接问题中自动恢复

RetryAndFollowUpInterceptor：重试重定向拦截器

a、请求失败后重新尝试连接：从Retry这个单词理解，但是在OKHttp中并不是所有的请求失败后（即返回码不是200）都会去重新连接，而是在发生 RouteException 或者 IOException 后再根据一些策略进行一些判断，如果可以恢复，就重新进请求

b、继续请求：FollowUp本意是跟进的意思，主要有以下几种类型可以继续发起请求：部分以3或4开头的返回码的情况下可以继续发送请求。

注意：其中FollowUp的次数受到限制，OKHTTP内部限制次数为20次以内

# Okhttp 基本底层原理

OkHttp 主要是通过 5 个[拦截器](https://so.csdn.net/so/search?q=%E6%8B%A6%E6%88%AA%E5%99%A8&spm=1001.2101.3001.7020" \t "_blank)和1个线程池中的3个双端队列（2 个异步队列，1 个同步队列）工作。内部实现通过一个责任链模式完成，将网络请求的各个阶段封装到各个链条中，实现了各层的解耦。

OkHttp 的底层是通过 Socket 发送 HTTP 请求与接受响应，但是 OkHttp 实现了连接池的概念，即对于同一主机的多个请求，可以公用一个 Socket 连接。

实现流程：

1. OkhttpClient 负责为Request 创建 Call；
2. RealCall 为Call的具体实现，其enqueue() 异步请求接口通过Dispatcher()调度器利用ExcutorService实现，而最终进行网络请求时和同步的execute()接口一致，都是通过 getResponseWithInterceptorChain() 函数实现
3. getResponseWithInterceptorChain() 中利用 Interceptor 链条，责任链模式 分层实现缓存、透明压缩、网络 IO 等功能；最终将响应数据返回给用户。

Dispatcher类：  
Interceptor类：

Dispatcher类：

Dispatcher分发器，通过维护一个线程池executorService来分发Request。  
线程池内部维护着三个队列：同步请求队列 runningSyncCalls、异步请求队列 runningAsyncCalls、异步缓存队列 readyAsyncCalls，整体可以参照生产者消费者模式来理解：Dispatcher是生产者，executorService是消费者池，runningSyncCalls、runningAsyncCalls和readyAsyncCalls是消费者，用来消费请求Call。

Interceptor类：

官网：拦截器是Okhttp中提供的一种强大机制，它可以实现网络监听、请求、以及响应重写、请求失败重试等功能。  
RetryAndFollowUpInterceptor、BridgeIntercept、CacheIntercept、ConnectIntercept、CallServerIntercept

RetryAndFollowUpInterceptor：重试和失败重定向拦截器，负责监控请求是否失败

BridgeInterceptor：桥接拦截器，添加请求头、cookie信息

CacheInterceptor：缓存拦截器，命中缓存则不会发起网络请求。

ConnectInterceptor：连接拦截器，维护一个连接池，负责连接复用，创建和释放连接以及创建连接上的socket流。

CallServerInterceptor：请求服务器拦截器，发起网络请求。将 request写进 IO 流当中，并且从 IO 流中读取响应 Response

# 如何使用OkHttp进行异步网络请求，并根据请求结果刷新UI

第一步，创建一个OkHttpClient对象 OkHttpClient mClient = new OkHttpClient.Builder().build();  
第二步，创建携带请求信息的Request对象 Request request = new Request.Builder().url("http://www.baidu.com").get().build();//.post(requestBody).build()  
第三步，创建Call对象  Call call = mClient.newCall(request);  
第四步，call.enqueue(callback)

第五步，使用onFailure和onResponse处理结果

第六步，调用RunOnUIThread（runnable）来更新UI

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

Request request = new Request.Builder().

url("https://github.com/cozing").

build();

Call call = client.newCall(request);

try {

//1.同步请求调用的方法是call.execute()，内部采用的是线程阻塞（一直等待直到线程返回结果）方式直接将结果返回到Response

Response response = call.execute();

//2.异步请求调用的方法是call.enqueue(Callback callback)，该方法需要传入一个Callback等待结果回调的接口

call.enqueue(new Callback() {

@Override

public void onFailure(Call call, IOException e) {

Log.w("cozing", "请求失败");

}

@Override

public void onResponse(Call call, Response response) throws IOException {

Log.w("cozing", "请求成功");

}

});

} catch (IOException e) {

e.printStackTrace();

}

# OkHttp对于网络请求都有哪些优化，如何实现的（几个拦截器）

a、通过连接池来减少请求延时:

我们知道，在okhttp中，我们每次的request请求都可以理解为一个connection，而每次发送请求的时候我们都要经过tcp的三次握手，然后传输数据，最后在释放连接。在高并发或者多个客户端请求的情况下，多次创建就会导致性能低下。如果能够connection复用的话，就能够很好地解决这个问题了。能够复用的关键就是客户端和服务端能够保持长连接，并让一个或者  
多个连接复用。怎么保持长连接呢？

在BridgeInterceptor的intercept()方法中requestBuilder.header("Connection", "Keep-Alive")，我们在request的请求头添加了("Connection", "Keep-Alive")的键值对，这样就能够保持长连接。而连接池ConnectionPoll就是专门负责管理这些长连接的类。需要注意的是，我们在初始化 ConnectionPoll的时候，会设置 闲置的connections 的最大数量为5个，每个最长的存活时间为5分钟。

b、无缝支持GZIP来减少数据流量

c、缓存响应数据来减少重复的网络请求

d、可以从很多常用的连接问题中自动恢复

# OkHttp框架中都用到了哪些设计模式

* 构造者模式（OkhttpClient,Request 等各种对象的创建）
* 工厂模式（在 Call 接口中，有一个内部工厂 Factory 接口。）
* 单例模式（Platform 类，已经使用 Okhttp 时使用单例）
* 策略模式（在 CacheInterceptor 中，在响应数据的选择中使用了策略模式，选择缓存数据还是选择网络访问。）
* 责任链模式（拦截器的链式调用）
* 享元模式（Dispatcher 的线程池中，不限量的线程池实现了对象复用）

# Okhttp 网络缓存如何实现？

OKHttp 默认只支持 get 请求的缓存。

* 第一次拿到响应后根据头信息决定是否缓存。
* 下次请求时判断是否存在本地缓存，是否需要使用对比缓存、封装请求头信息等等。
* 如果缓存失效或者需要对比缓存则发出网络请求，否则使用本地缓存。

# Okhttp 网络连接怎么实现复用？

HttpEngine 在发起请求之前，会先调用nextConnection()来获取一个Connection对象，如果可以从ConnectionPool中获取一个Connection对象，就不会新建，如果无法获取，就会调用createnextConnection()来新建一个Connection对象，这就是 Okhttp 多路复用的核心，不像之前的网络框架，无论有没有，都会新建Connection对象。