OkHttp3.8源码分析

管理配置

对外提供统一的接口和调用方式，对内提供配置选项，基于外观者模式，涉及类：
OkHttpClient

请求

描述一个请求，是什么，包括什么，如何生成，如何自定义。主要涉及的类：
Request;
HttpUrl;
RequestBody;
Headers;
CacheControl;

任务分发

主要描述了如何进行、记录、取消请求，包括同步请求、异步请求如何处理，如何管理多线程，记录进行中、未进行的请求。主要涉及类：
Call;
RealCall;
AsyncCall(runnable,维护RealCall);
Dispatcher;

缓存

描述了一个请求的是否可以从本地获取响应，网络响应是否需要缓存，包括如何定义缓存策略，如何将策略设置到一次网络连接过程，如何缓存数据，如何读取等问题。主要涉及类：
Cache;
Cache-Strategy;
Cache-Control;
InternalCache ;
DiskLruCache;
CacheRequest

连接

描述了本地-远程网络通信，包括定位服务器位置，如何建立Socket连接，是否使用代理，是否使用Https，使用的Http协议的版本，如何与Socket通信，如何管理这些连接；主要涉及类：
Connection;
Address;
Route;
Protocal;
Socket;
ConnectionPool ;
RealConnection;
StreamAllocation;

响应

描述了一个完整的网络响应，包括协议，状态码，状态信息，响应头，响应体。并维护缓存响应的引用。
主要涉及类：
Response;
ResponseBody;
Headers;
CacheControl;

流程管理

通过Chain和拦截器搭配，在整个请求-响应 网络连接过程中，对请求或者响应做观察、修改等操作，主要包括网络请求拦截器，创建连接拦截器，请求桥联转换拦截器，缓存拦截器，重连跟踪拦截器，日志拦截器，以实现一个请求获得响应的整个流程，具体见各类详解,主要涉及类：
Chain;
RealInterceptorChain;
RetryAndFollowUpInterceptor ;
BridgeInterceptor ;
CacheInterceptor ;
ConnectInterceptor ;
CallServerInterceptor

Call Factory:
工厂接口：
Call newCall(Request request);

OkHttpClient：
外观类，主要是管理连接各配置，生成Call，websocket。
建造者模式：维持建成对象的不可变性。
方便克隆对象，复用一些配置。
newCall(request)方法，创建一个Call

Call
接口，是一次可以执行的请求，其对象代表了一个请求/响应对，只能执行一次。方法：
1、Request request():获取原始请求；
2.Response execute()；同步执行请求，阻塞直到有结果或者错误返回；
3.void enqueue(Callback responseCallback):异步请求；
还有是否执行，取消，是否取消，克隆等方法；
内置工厂

RealCall：
真正负责任务分发执行的类，每次同步执行Request时，向Dispatcher中注册；异步执行时，也注册，并且由Dispatcher的线程池来管理执行线程；核心方法：Response getResponseWithInterceptorChain();通过一系列内置的，自定义的拦截器以及Chain，输入Request，得到Response

AsyncCall：
内部类，Runnable类，内部维护RealCall，用于异步执行。

Dispatcher类：
任务记录以及线程管理类，内部维持 待进行Call，正在进行的同步Call，异步执行的Call的列表，维持一个缓存线程池来管理异步执行任务，可以集中取消Call，设置最大并发请求数量和单域名最大请求数量

RealInterceptorChain:
真正的拦截链，包含了必要的数据：
List<Interceptor> interceptors;--拦截器列表
private final StreamAllocation streamAllocation;链接/流的分配；
private final HttpCodec httpCodec;
private final RealConnection connection;--真正的连接，创建（HTTP或者HTTPS，包含通过tunnel的情况）socket开连接socket;
private final int index;
private final Request request;
核心方法proceed（...）搭配interceptor，进行一系列流程

执行顺序

HttpLoggingInterceptor：
log日志拦截器，主要是打印一次请求过程中的日志，包括NONE,BASIC,HEADER,BODY四种级别

RetryAndFollowUpInterceptor：
重试和跟踪拦截器，主要是从失败的请求中恢复重试，并在有必要时跟踪重定向；重试和校验欠数跟浏览器有关；

BridgeInterceptor：
桥联客户端代码和网络代码，首先基于用户端的请求创建一个网络请求，然后将其处理传递给网络，最后将结果构建成结果回传给客户端

CacheInterceptor：
缓存拦截器，从缓存里为请求提供响应；将响应写入缓存中

ConnectInterceptor：
连接拦截器，开启一个通向服务器的连接，处理后推给下一个拦截器

CallServerInterceptor：
链上最后一个拦截器，向服务器发起网络请求

Interceptor:
接口，可观察、修改请求、响应，可用来增删改改header。
Response intercept(Chain chain)

Chain：
接口，代表了一个请求-响应 子链接

request

response

chain

RealInterceptorChain

拦截器，观察，修改请求和响应

sever

Http连接

Connection Socket Stream

连接管理模块

CallServer拦截器，观察，修改请求和响应

Request request();

Response proceed(Request request)

Connection connection()

Builder：
构建者：
url,URL,HttpUrl-->HttpUrl;
header：增删改查；
method:GET,HEAD,POST,DELETE,PUT;
public Builder cacheControl(CacheControl cacheControl) {
String value = cacheControl.toString();
if (value.isEmpty()) return
removeHeader("Cache-Control");
return header("Cache-Control", value);
}

Request：
请求类
final HttpUrl url;
final String method;
final Headers headers;
RequestBody body;
final Object tag;
private volatile CacheControl cacheControl;

CacheControl:
设置Cache-Control header的类，比在Header里直接设置要更全面灵活。为HTTP请求和响应配置有关缓存的头部信息

Headers：
自定义请求头，String[] = {"键"，"值"，"键"，"值"...}
方法主要是怎么基于String[]增删改header

HttpCodec：
接口，编码解码器。
编码HTTP请求，解码HTTP响应。
向请求写入header，将请求转为输出流（比如写入到socket，对外提供流Sink，以供数据传输）；
读取响应的headers，获取响应的响应体等
Http 2.0 --Http2Codec
Http 1.x --Http1Codec

Response:
请求：Request request;
协议：Protocol protocol;
状态码：int code;
状态信息：String message;
握手信息： Handshake handshake;
响应头：Headers headers;
响应体： ResponseBody body;
网络响应：Response networkResponse;
缓存响应：Response cacheResponse;
优先响应：Response priorResponse;
发送网络请求时的时间戳：long sentRequestAtMillis;
接收到网络请求时的时间戳：long receivedResponseAtMills;
缓存控制：CacheControl cacheControl;

ResponseBody：
响应体，由服务器发给客户端的一次性的流，包含原始的字节数据，每一个响应体都基于遵向web服务器的一个活跃连接。body基于有限的资源，如socket或者file，所以用完之后要关闭，否则会造成资源泄漏。可以通过close()方法来关闭。
body流只能消费一次，可以接受超大空间的响应资源，若需要，则可以自己将body流保存，比如通过byte(),string(),source(),charStream()等。
主要方法：
内容类型：contentType（）；
内容长度 contentLength()；
bytes()，将响应体转化为byte数组，一次性载入内存；
string()，将响应体转为字符串，一次性载入内存，字符集UTF-8或者BOM；
source()，将响应转为source，方便流读取；
另外还有byteStream()，charStream()结果读取方式。

可以自定义，比如监听进度，推荐装饰者模式。

Builder:
建造者，主要目的：链式调用，对象创建后的属性不变性，复用对象

CallBack:
接口，结果回调，两个方法：
1.onFailure,可能是取消、连接问题或者超时造成；
2.onResponse,说明传输层是成功的，但未必代表你完了成功的app层面的请求。结果可以从Response.body读取

ConnectionPool：
连接管理池，用来管理Connection复用的类，相同地址的请求可能会复用一个连接；
内部数据结构：双端队列

Connection：
接口，The sockets and streams of an HTTP, HTTPS, or HTTPS+HTTP/2 connection，可用来描述多个请求-响应的数据交换，可能连接到真实服务器或者代理。
每个连接可以承载一些流，http1.x承载0-1个流，http 2可承载任意数量的流，0流被认为是空闲流，用来复用，比新建成本低。
methods:
Socket socket();
Route route();
Handshake handshake();
Protocol protocol();

RouteDatabase：
失败路线的黑名单，用来让client从错误中学习的，当尝试连接一个地址的错误连接，则直接使用替换路线

Route：
到达一个抽象源服务连接使用的具体路线，当创建一个连接时，客户端有多个选项：ip地址，代理。
成员：
final Address address;
final Proxy proxy;
final InetSocketAddress inetSocketAddress;

Address：
地址，连接服服务器连接的说明类，至少包含主域名，端口信息，对于有代理的则需要有代理信息；有安全要求的，如https，则要有ssl生成工厂，主域名校验器，证书固定器

HandShake
TLS handshake的记录，对于Https，本地和远程服务器即为端手的两端，该类对象描述了复杂的端手过程,需要使用ConnectionSpec来为新端手设置policy:
成员：
Tls版本，加密算法套件，证书列表等

Protocol：
协议，包括短连接的HTTP1.0,长链接的HTTP1.1，以及HTTP2.0（长链接，服务器推送，单socket多请求）

RealConnection:
真正的连接，主要职责是创建HTTP、HTTPS、基于代理或者tunnel的Socket，并连接之；然后与Socket建立流的关系，通过okio的Sink和Source，来进行socket的数据传输、交换。连接成功后，可基于流对外提供HTTP的编码解码器--Codec

StreamAllocation:
主要用来协调三方面的关系（Connections，Streams和Calls），该类的实例代表了正在通过一个或多个连接使用一个或多个流的Call。核心方法：
HttpCodec newStream(OkHttpClient client, boolean doExtensiveHealthChecks)与findConnection（。。。）：先尝试从连接池中复用一个带流的连接，若找不到，则新建一个真正的连接--RealConnection，并连接，添加到连接池中；然后获得该连接下流的编码解码器，以供刚处使用。

Http上传多内容时的格式--MultiPart：

其它Http header略，【】里面的是注释。

Content-Type:multipart/form-data;boundary=--------------------------7d33a816d302b6
【注释，声明内容类型，声明边界的代号，即声明boundary用什么代表，Http协议中使用 \r\n 表示换行】
【以下是第一部分part】
--boundary【换行】
Content-Disposition:form-data;name="the name";filename="path"【换行】【表单提交，可无，也可无name或者filename】
key:value【键值对列表，可无，每个都换行】
Content-Type:image/*【可无，换行】
Content-Length:1000【可无，换行】
【换行】
body
【换行】
【注释，以下第二部分】
--boundary【换行，boundary前面有--】
Content-Disposition:form-data;name="the name"【换行】【表单提交】
Content-Length:1000【可无，换行】
【换行】
body
【换行】
--boundary--【结束】

HttpUrl:
Http还者Https协议下的统一资源定位符URL，用来描述资源在web上的位置。虽然Java也提供了URL和URI，但是HttpUrl可以定位一些其它问题。
HttpUrl url = new HttpUrl.Builder()
* scheme("https")
* host("www.google.com")
* addPathSegment("search")
* addQueryParameter("q", "polar bears")
* build();
* System.out.println(url);
* }
https://www.google.com/search?q=polar%20bears

</pre> ></pre> ></pre> >
协议：域名：端口：路径：请求参数（一般为一系列键值对）</pre> ></pre> ></pre> ></pre> >

主要方法：组成，拆分URL，编解码

RequestBody：
抽象请求体，由媒体类型，大小和具体内容决定。
结合okio库实现请求体的构建，方法主要是：获取媒体类型，获取媒体大小，向缓冲槽(BufferedSink)里写入数据。库里提供了这些类型：
字符串（比如json,xml，text）byte[]型，ByteString，文件型File，表单型(FormRequestBody)，多部分型（MultiPartBody），还可继承自定义，比如上传时监控进度，推荐装饰器

Builder:
建造者，主要目的：链式调用，对象创建后的属性不变性，复用对象

CacheStrategy:
基于一个请求和缓存的响应，根据CacheControl的规则以及现有的request，缓存过的响应来判断是否使用网络，是否使用缓存或者同时使用。

CacheRequest：
接口，缓存请求，包括：
Sink body()；
abort()方法。
主要目的是，提供缓存该请求下的数据操作，包括完成或者废弃，记录写入、命中次数。

InternalCache：
内部缓存接口，定义了请求-响应的缓存的增删改查操作；委托给了Cache

Cache:
缓存HTTP/HTTPS响应到文件系统，用来节省时间宽带等，具体规则由CacheControl来提供，具体缓存技术基于LruDiskCache，该类主要是基于缓存规则下，将request-response和文件系统的entry互换，进行请求-响应缓存的增删改查操作，属于针对okhttp的业务类

CacheRequestImpl：
lrucache的entry编辑器输出流的封装，方便保持输出的原子性和一致性

DiskLruCache：
基于磁盘存储的缓存，有限定存储空间大小，每个键值对都有一个键（Ukey,符合限定正则标准）和若干个值（对应文件系统或者流），对外提供最近最少原则下的数据的增删改查操作。内部包含一个LinkedHashMap，用来维护数据访问顺序；维护日志文件，记录每次操作的必要信息，包括每个键值对记录，并通过日志文件，在每次创建该类时，同步磁盘和内存信息，用于后续增删改查时进一步方便同步数据。
静态方法创建，需要指定存储文件，存储限额空间，每个元素值的最目等。
edit（key,long）：提供key的编辑器；
Snapshot get(String key)：获取key的快照，方便读取数据；
trimToSize()：根据LRU和maxsize来保证存储限额的有效性。

Entry:
键值对抉射，主要是key-文件系统的若干（指时文件，文件，文件大小），可以提供值的快照--Snapshot，包含一个对应的编辑器--Editor

Snapshot：
键值对快照，主要是key以及value对应各个文件的流。方便访问，且能保证文件发生修改还者删除时，不会影响在读的数据

Editor:
每个键值对的编辑器，提供数据的创建和修改线程安全操作；每次编辑结束时，需要有commit()进行完成操作或者有abort()进行终止操作。读写操作是通过提供流的方式进行：
newSource（）输入流；
newSink（）输出流

LruCache：
声明最大可用的缓存空间和每个数据元素所占用的存储，在每次踏入新数据时，根据最近最少使用原则，来控制数据存储总量，并提供增删改查操作：
trimToSize（max）控制已存储大小；
sizeOf(K key, V value)，返回每个键值对占据存储

LinkedHashMap:
主要是记录数据的访问顺序，以确定最近最少使用的数据

基于循环双链表+HashMap的数据结构，可以根据accessable标志位，来记录数据踏入顺序或者数据访问顺序；每次增改查元素时，进行链表移动节点操作