HenCoder Plus 讲义

从 OkHttp 的原理来看 HTTP

OkHttp 使用方法简介

1. 创建一个 OkHttp 的实例

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

2. 创建 Request

```
Request request = new Request.Builder()
        .url("http://hencoder.com")
        .build();
```

3. 创建 Call 并发起网络请求

```
client.newCall(request).enqueue(new Callback() {
    @Override
    public void onFailure(Call call, IOException e) {

    }

    @Override
    public void onResponse(Call call, Response response)
throws IOException {
        Log.d("okhttp response",
response.body().string());
    }
});
```

OkHttp 源码总结

- `OkHttpClient` 相当于配置中心，所有的请求都会共享这些配置（例如出错是否重试、共享的连接池）。`OkHttpClient` 中的配置主要有：
  - `Dispatcher dispatcher`：调度器，用于调度后台发起的网络请求，有后台总请求数和单主机总请求数的控制。
  - `List<Protocol> protocols`：支持的应用层协议，即 HTTP/1.1、HTTP/2 等。
  - `List<ConnectionSpec> connectionSpecs`：应用层支持的 Socket 设置，即使用明文传输（用于 HTTP）还是某个版本的 TLS（用于 HTTPS）。
  - `List<Interceptor> interceptors`：大多数时候使用的 Interceptor 都应该配置到这里。
  - `List<Interceptor> networkInterceptors`：直接和网络请求交互的 Interceptor 配置到这里，例如如果你想查看返回的 301 报文或者未解压的 Response Body，需要在这里看。
  - `CookieJar cookieJar`：管理 Cookie 的控制器。OkHttp 提供了 Cookie 存取的判断支持（即什么时候需要存 Cookie，什么时候需要读取 Cookie，但没有给出具体的存取实现。如果需要存取 Cookie，你得自己写实现，例如用 `Map` 存在内存里，或者用别的方式存在本地存储或者数据库。
  - `Cache cache`：Cache 存储的配置。默认是没有，如果需要用，得自己配置出 Cache 存储的文件位置以及存储空间上限。
  - `HostnameVerifier hostnameVerifier`：用于验证 HTTPS 握手过程中下载到的证书所属者是否和自己要访问的主机名一致。
  - `CertificatePinner certificatePinner`：用于设置 HTTPS 握手过程中针对某个 Host 额外的的 Certificate Public Key Pinner，即把网站证书链中的每一个证书公钥直接拿来提前配置进 OkHttpClient 里去，作为正常的证书验证机制之外的一次额外验证。
  - `Authenticator authenticator`：用于自动重新认证。配置之后，在请求收到 401 状态码的响应是，会直接调用 `authenticator`，手动加入 `Authorization` header 之后自动重新发起请求。
  - `boolean followRedirects`：遇到重定向的要求是，是否自动 follow。
  - `boolean followSslRedirects` 在重定向时，如果原先请求的是 http 而重定向的目标是 https，或者原先请求的是 https 而重定向的目标是 http，是否依然自动 follow。（记得，不是「是否自动 follow HTTPS URL

重定向的意思，而是是否自动 follow 在 HTTP 和 HTTPS 之间切换的重定向）
    - `boolean retryOnConnectionFailure`：在请求失败的时候是否自动重试。注意，大多数的请求失败并不属于 OkHttp 所定义的「需要重试」，这种重试只适用于「同一个域名的多个 IP 切换重试」「Socket 失效重试」等情况。
    - `int connectTimeout`：建立连接（TCP 或 TLS）的超时时间。
    - `int readTimeout`：发起请求到读到响应数据的超时时间。
    - `int writeTimeout`：发起请求并被目标服务器接受的超时时间。（为什么？因为有时候对方服务器可能由于某种原因而不读取你的 Request）
- `newCall(Request)` 方法会返回一个 `RealCall` 对象，它是 `Call` 接口的实现。当调用 `RealCall.execute()` 的时候，`RealCall.getResponseWithInterceptorChain()` 会被调用，它会发起网络请求并拿到返回的响应，装进一个 `Response` 对象并作为返回值返回；`RealCall.enqueue()` 被调用的时候大同小异，区别在于 `enqueue()` 会使用 `Dispatcher` 的线程池来把请求放在后台线程进行，但实质上使用的同样也是 `getResponseWithInterceptorChain()` 方法。
- `getResponseWithInterceptorChain()` 方法做的事：把所有配置好的 `Interceptor` 放在一个 `List` 里，然后作为参数，创建一个 `RealInterceptorChain` 对象，并调用 `chain.proceed(request)` 来发起请求和获取响应。
- 在 `RealInterceptorChain` 中，多个 `Interceptor` 会依次调用自己的 `intercept()` 方法。这个方法会做三件事：
    1. 对请求进行预处理
    2. 预处理之后，重新调用 `RealIntercepterChain.proceed()` 把请求交给下一个 `Interceptor`
    3. 在下一个 `Interceptor` 处理完成并返回之后，拿到 Response 进行后续处理

    > 当然了，最后一个 `Interceptor` 的任务只有一个：做真正的网络请求并拿到响应
- 从上到下，每级 `Interceptor` 做的事：
    - 首先是开发者使用 `addInterceptor(Interceptor)` 所设置的，它们会按照开发者的要求，在所有其他 `Interceptor` 处理之前，进行最早的预处理工作，以及在收到 Response 之后，做最后的善后工作。如果你有统一的 header 要添加，可以在这里设置；

- 然后是 `RetryAndFollowUpInterceptor`：它会对连接做一些初始化工作，并且负责在请求失败时的重试，以及重定向的自动后续请求。它的存在，可以让重试和重定向对于开发者是无感知的；
- `BridgeInterceptor`：它负责一些不影响开发者开发，但影响 HTTP 交互的一些额外预处理。例如，Content-Length 的计算和添加、gzip 的支持（Accept-Encoding: gzip）、gzip 压缩数据的解包，都是发生在这里；
- `CacheInterceptor`：它负责 Cache 的处理。把它放在后面的网络交互相关 `Interceptor` 的前面的好处是，如果本地有了可用的 Cache，一个请求可以在没有发生实质网络交互的情况下就返回缓存结果，而完全不需要开发者做出任何的额外工作，让 Cache 更加无感知；
- `ConnectInterceptor`：它负责建立连接。在这里，OkHttp 会创建出网络请求所需要的 TCP 连接（如果是 HTTP），或者是建立在 TCP 连接之上的 TLS 连接（如果是 HTTPS），并且会创建出对应的 `HttpCodec` 对象（用于编码解码 HTTP 请求）；
- 然后是开发者使用 `addNetworkInterceptor(Interceptor)` 所设置的，它们的行为逻辑和使用 `addInterceptor(Interceptor)` 创建的一样，但由于位置不同，所以这里创建的 `Interceptor` 会看到每个请求和响应的数据（包括重定向以及重试的一些中间请求和响应），并且看到的是完整原始数据，而不是没有加 Content-Length 的请求数据，或者 Body 还没有被 gzip 解压的响应数据。多数情况，这个方法不需要被使用，不过如果你要做网络调试，可以用它；
- `CallServerInterceptor`：它负责实质的请求与响应的 I/O 操作，即往 Socket 里写入请求数据，和从 Socket 里读取响应数据。

问题和建议?

课上技术相关的问题，都可以去群里和大家讨论，对于比较通用的、有价值的问题，可以去我们的知识星球提问。

具体技术之外的问题和建议，都可以找丢物线（微信：diuwuxian），丢丢会为你解答技术以外的一切。

更多内容：

- 网站：https://hencoder.com；https://kaixue.io
- 各大搜索引擎、微信公众号、微博、知乎、掘金、哔哩哔哩、YouTube、西瓜视频、抖音、快手、微视：统一账号「**扔物线**」，我会持续输出优质的技术内容，欢迎大家关注。
- 哔哩哔哩快捷传送门：https://space.bilibili.com/27559447

  大家如果喜欢我们的课程，还请去扔物线的哔哩哔哩，帮我素质三连，感谢大家！