OkHttp3源码分析[缓存策略]


BlackSwift (/u/b99b0edd4e77) ＋关注
2016.01.23 01:29* 字数 2019 阅读 16557 评论 33 喜欢 72 赞赏 5
(/u/b99b0edd4e77)

OkHttp系列文章如下

- OkHttp3源码分析[综述] (https://www.jianshu.com/p/aad5aacd79bf)
- OkHttp3源码分析[复用连接池] (https://www.jianshu.com/p/92a61357164b)
- OkHttp3源码分析[缓存策略] (https://www.jianshu.com/p/9cebbbd0eeab)
- OkHttp3源码分析[DiskLruCache] (https://www.jianshu.com/p/23b8aa490a6b)
- OkHttp3源码分析[任务队列] (https://www.jianshu.com/p/6637369d02e7)

本文专门分析OkHttp的缓存策略，应该是okhttp分析中最简单的一篇了

HTTP缓存基础知识

在分析源码之前，我们先回顾一下http的缓存Header的含义

1. Expires

表示到期时间，一般用在response报文中，当超过此事件后响应将被认为是无效的而需要网络连接，反之而是直接使用缓存

```
Expires: Thu, 12 Jan 2017 11:01:33 GMT
```

2. Cache-Control

相对值，单位是秒，指定某个文件被续多少秒的时间，从而避免额外的网络请求。比expired更好的选择，它不用要求服务器与客户端的时间同步，也不用服务器时刻同步修改配置 `Expired` 中的绝对时间，而且它的优先级比 `Expires` 更高。比如简书静态资源有如下的header，表示可以续31536000秒，也就是一年。

```
Cache-Control: max-age=31536000, public
```

3. 修订文件名(Reving Filenames)

如果我们通过设置header保证了客户端可以缓存的，而此时远程服务器更新了文件如何解决呢？我们这时可以通过修改url中的文件名版本后缀进行缓存，比如下文是又拍云的公共CDN就提供了多个版本的JQuery

```
upcdn.b0.upaiyun.com/libs/jquery/jquery-2.0.3.min.js
```

这个方法是最简单的，实践性非常高

4. 条件GET请求(Conditional GET Requests)与304

如缓存果过期或者强制放弃缓存，在此情况下，缓存策略全部交给服务器判断，客户端只用发送 `条件get请求` 即可，如果缓存是有效的，则返回 `304 Not Modifiled` ，否则直接返回body。

请求的方式有两种：

4.1. Last-Modified-Date:

客户端第一次网络请求时，服务器返回了

```
Last-Modified: Tue, 12 Jan 2016 09:31:27 GMT
```

客户端再次请求时，通过发送

```
If-Modified-Since: Tue, 12 Jan 2016 09:31:27 GMT
```

交给服务器进行判断，如果仍然可以缓存使用，服务器就返回 `304`

4.2. ETag

ETag是对资源文件的一种摘要，客户端并不需要了解实现细节。当客户端第一请求时，服务器返回了

```
ETag: "5694c7ef-24dc"
```

客户端再次请求时，通过发送

```
If-None-Match:"5694c7ef-24dc"
```

交给服务器进行判断，如果仍然可以缓存使用，服务器就返回 `304`

> 如果 ETag 和 Last-Modified 都有，则必须一次性都发给服务器，它们没有优先级之分，反正这里客户端没有任何判断的逻辑。

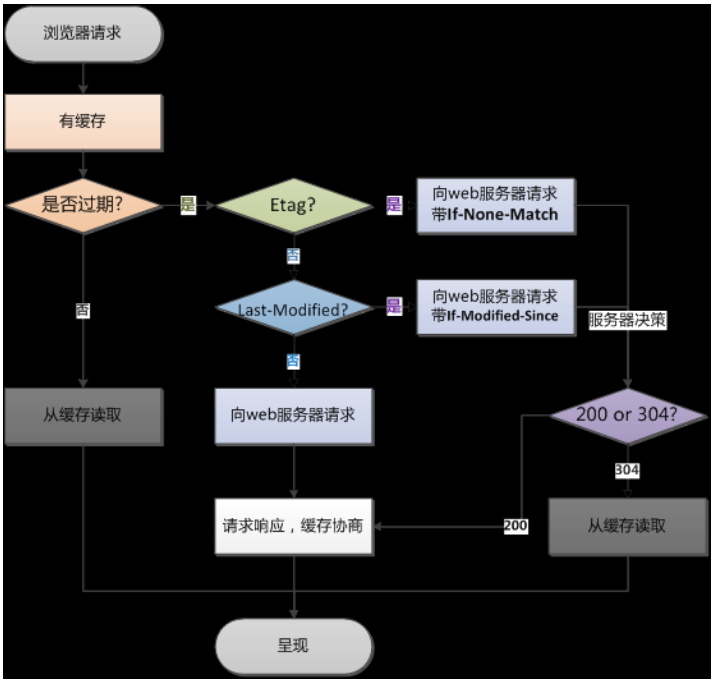
5. 其它标签

- no-cache/no-store: 不使用缓存，no-cache指令的目的是防止从缓存中返回过期的资源。客户端发送的请求中如果包含no-cache指令的话，表示客户端将不会接受缓存过的相应，于是缓存服务器必须把客户端请求转发给源服务器。服务器端返回的相应中包含no-cache指令的话那么缓存服务器不能对资源进行缓存。
- only-if-cached: 只使用缓存
- Date: The date and time that the message was sent
- Age: The Age response-header field conveys the sender's estimate of the amount of time since the response (or its revalidation) was generated at the origin server. 说人话就是CDN反代服务器到原始服务器获取数据延时的缓存时间

> "only-if-cached"标签非常具有诱导性，它只在 (https://link.jianshu.com?t=https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9)请求中使用，表示无论是否有网完全只使用缓存（如果命中还好说，否则返回503错误/网络错误），这个标签比较危险。

> 全部的标签，可以到这里看 (https://link.jianshu.com?t=https://en.wikipedia.org/wiki/List_of_HTTP_header_fields)

以上内容是作为一个服务器开发或者客户端的常识，下图是网上找的总结，注意图中的 ETag 和 Last-Modified 可能有优先级的歧义，你只需要记住它们是没有优先级的。



图源: 浏览器缓存机制 - [吴秦（Tyler）](http://www.cnblogs.com/skynet/)

2. 源码分析

OkHttp中使用了 `CacheStrategy` (https://link.jianshu.com?t=https://github.com/square/okhttp/blob/db9c2db40b0b89a1853715fd52e2748463d9cc9c/okhttp/src/main/java/okhttp3/internal/http/CacheStrategy.java#L31-L31)实现了上文的流程图，它根据之前的缓存结果与当前将要发送Request的header进行策略分析，并得出是否进行请求的结论。

2.1. 总体请求流程分析

CacheStrategy类似一个 `mapping` 操作，将两个值输入，再将两个值输出

Input	**request, cacheCandidate**
↓	↓
CacheStrategy	处理，判断Header信息
↓	↓
Output	networkRequest, cacheResponse

Request:
开发者手动编写并在 `Interceptor` 中递归加工而成的对象（如果读者需要调试分析的话，可以用logging-interceptor (https://link.jianshu.com?t=https://github.com/square/okhttp/tree/master/okhttp-logging-interceptor)进行log操作），我们只需要知道了目前传入的Request中并没有任何关于缓存的Header

cacheCandidate:
也就是上次与服务器交互缓存的Response，可能为null。这里的缓存全部是基于文件系统的Map，key是请求中url的md5，value是在文件中查询到的缓存，页面置换基于 `LRU` 算法 (https://link.jianshu.com?t=https://en.wikipedia.org/wiki/Page_replacement_algorithm#Least_recently_used)，我们现在只需要知道它是一个可以读取 `缓存Header` 的Response即可。

当被 `CacheStrategy` 加工输出后，输出 `networkRequest` 与 `cacheResponse` ，根据是否为空执行不　的请求

networkRequest	**cacheResponse**	**result**

networkRequest	cacheResponse	result
null	null	only-if-cached(表明不进行网络请求，且缓存不存在或者过期，一定会返回503错误)
null	non-null	不进行网络请求，而且缓存可以使用，直接返回缓存，不用请求网络
non-null	null	需要进行网络请求，而且缓存不存在或者过期，直接访问网络
non-null	non-null	Header中含有 `ETag/Last-Modified` 标签，需要在 `条件请求` 下使用，还是需要访问网络

> 以上是对networkRequest/cacheResponse进行findusage查询获得出的结论

基本上与上文的图片完全一致，以上就是OkHttp的缓存策略

> 关于此部分的分析，读者可以在HttpEngine (https://link.jianshu.com?t=https://github.com/square/okhttp/blob/b8c5938ed49502fbae89d0e842389853a208f996/okhttp/src/main/java/okhttp3/internal/http/HttpEngine.java#L82-L82)对象中通过对 `userResponse` 进行findUsage分析得出，源码都是一大堆的if判断

2.2. CacheStrategy的加工过程

`CacheStrategy` 使用Factory模式 (https://link.jianshu.com?t=https://en.wikipedia.org/wiki/Factory_method_pattern)进行构造，参数如下

```
InternalCache responseCache = Internal.instance.internalCache(client);
//cacheCandidate从disklurcache中获取
//request的url被md5序列化为key,进行缓存查询
Response cacheCandidate = responseCache != null ? responseCache.get(request) : null;
//请求与缓存
factory = new CacheStrategy.Factory(now, request, cacheCandidate);
cacheStrategy = factory.get();
//输出结果
networkRequest = cacheStrategy.networkRequest;
cacheResponse = cacheStrategy.cacheResponse;
//进行一大堆的if判断，内容同上表格
.....
```

可以看出 `Factory.get()` 是最关键的缓存策略的判断，我们点入 `get()` 方法 (https://link.jianshu.com?t=https://github.com/square/okhttp/blob/db9c2db40b0b89a1853715fd52e2748463d9cc9c/okhttp/src/main/java/okhttp3/internal/http/CacheStrategy.java#L166-L166)，可以发现是对 `getCandidate()` 的一个封装，我们接着点开 `getCandidate()` ，全是if与数学计算，详细代码如下

```
private CacheStrategy getCandidate() {
  //如果缓存没有命中(即null),网络请求也不需要加缓存Header了
  if (cacheResponse == null) {
    //`没有缓存的网络请求,查上文的表可知是直接访问
    return new CacheStrategy(request, null);
  }

  // 如果缓存的TLS握手信息丢失,返回进行直接连接
  if (request.isHttps() && cacheResponse.handshake() == null) {
    //直接访问
    return new CacheStrategy(request, null);
  }

  //检测response的状态码,Expired时间,是否有no-cache标签
  if (!isCacheable(cacheResponse, request)) {
    //直接访问
    return new CacheStrategy(request, null);
  }

  CacheControl requestCaching = request.cacheControl();
  //如果请求报文使用了`no-cache`标签(这个只可能是开发者故意添加的)
  //或者有ETag/Since标签(也就是条件GET请求)
  if (requestCaching.noCache() || hasConditions(request)) {
    //直接连接,把缓存判断交给服务器
    return new CacheStrategy(request, null);
  }
  //根据RFC协议计算
  //计算当前age的时间戳
  //now - sent + age (s)
  long ageMillis = cacheResponseAge();
  //大部分情况服务器设置为max-age
  long freshMillis = computeFreshnessLifetime();

  if (requestCaching.maxAgeSeconds() != -1) {
    //大部分情况下是取max-age
    freshMillis = Math.min(freshMillis, SECONDS.toMillis(requestCaching.maxAgeSeconds()));
  }

  long minFreshMillis = 0;
  if (requestCaching.minFreshSeconds() != -1) {
    //大部分情况下设置是0
    minFreshMillis = SECONDS.toMillis(requestCaching.minFreshSeconds());
  }

  long maxStaleMillis = 0;
  //ParseHeader中的缓存控制信息
  CacheControl responseCaching = cacheResponse.cacheControl();
  if (!responseCaching.mustRevalidate() && requestCaching.maxStaleSeconds() != -1) {
    //设置最大过期时间,一般设置为0
    maxStaleMillis = SECONDS.toMillis(requestCaching.maxStaleSeconds());
  }

  //缓存在过期时间内,可以使用
  //大部分情况下是进行如下判断
  //now - sent + age + 0 < max-age + 0
  if (!responseCaching.noCache() && ageMillis + minFreshMillis < freshMillis + maxStaleMilli
    //返回上次的缓存
    Response.Builder builder = cacheResponse.newBuilder();
    return new CacheStrategy(null, builder.build());
  }

  //缓存失效, 如果有etag等信息
  //进行发送`conditional`请求,交给服务器处理
  Request.Builder conditionalRequestBuilder = request.newBuilder();

  if (etag != null) {
    conditionalRequestBuilder.header("If-None-Match", etag);
  } else if (lastModified != null) {
    conditionalRequestBuilder.header("If-Modified-Since", lastModifiedString);
  } else if (servedDate != null) {
    conditionalRequestBuilder.header("If-Modified-Since", servedDateString);
  }
  //下面请求实质还说网络请求
  Request conditionalRequest = conditionalRequestBuilder.build();
  return hasConditions(conditionalRequest) ? new CacheStrategy(conditionalRequest,
      cacheResponse) : new CacheStrategy(conditionalRequest, null);
}
```

太长不看的话，大多数常见的情况可以用这个估算

```
now - sent + age < max-age
```

这里有个技巧，对构造函数进行 `findUsage` 查询，就可以看出各个输出是否为空的结果，然后各个击破分析

new CacheStrategy()

3. 结论

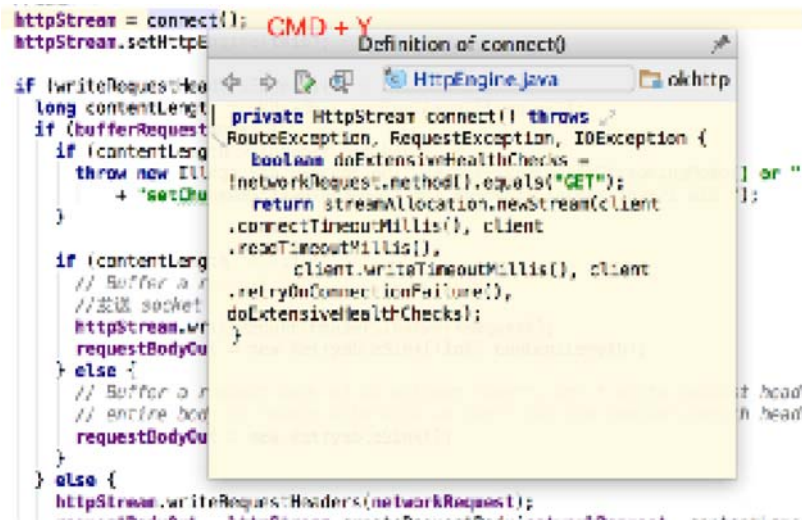
通过上面的分析，我们可以发现，okhttp实现的缓存策略实质上就是大量的if判断集合，这些是根据RFC标准文档写死的，并没有相当难的技巧。

1. Okhttp的缓存是自动完成的，完全由服务器Header决定的，自己**没有必要**进行控制。网上热传的文章在 `Interceptor` 中手工添加缓存代码控制，它固然有用，但是属于Hack式的利用，违反了RFC文档标准，不建议使用，OkHttp的官方缓存控制在注释中(https://link.jianshu.com?t=https://github.com/square/okhttp/blob/d662c1a82851800c46ad8ede2d9d10d10427fdad/okhttp/src/main/java/okhttp3/Cache.java#L79)。如果读者的需求是对象持久化，建议用文件储存或者数据库即可（比如realm）。

2. 服务器的配置非常重要，如果你需要减小请求次数，建议直接找对接人员对 `max-age` 等头文件进行优化；服务器的时钟需要严格NTP同步

3. 充分利用Idea的findUsage的功能，源码的各个跳转条件可以很快分析完成

4. 使用 `CMD + Y` 可以快速预览某个函数，类似于forcetouch功能



Idea quick preview

5. 使用 `CMD + 左键` 可以添加标签，方便跳转代码，如图