HTTP

1. 概念与原理

> http（超文本传输协议）是一个基于`请求与响应模式`的、`无状态`的、`应用层`的协议，常基于TCP的连接方式；

1.1 请求

> http请求由三部分组成，分别是：`请求行`、`消息报头`、`请求正文`；
> 报头与正文之间以空行分隔；

1.1.1 请求行

> 格式：Method Request-URI HTTP-Version CRLF

- 以一个方法符号开头 - get、post、delete、head等
  - **head** - 与GET方法一样，都是向服务器发出指定资源的请求。只不过服务器将不传回资源的请求体部分。它的好处在于，使用这个方法可以在不必传输全部内容的情况下，就可以获取其中“关于该资源的信息”（元信息或称元数据）。
  - **TRACE** - 回显服务器收到的请求，主要用于测试或诊断
  - **PTIONS** - 这个方法可使服务器传回该资源所支持的所有HTTP请求方法。用‘*’来代替资源名称，向Web服务器发送OPTIONS请求，可以测试服务器功能是否正常运作
  - **CONNECT** - HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。通常用于SSL加密服务器的链接（经由非加密的HTTP代理服务器）
- URI
- 协议的版本

1.2 响应

> HTTP响应也是由三个部分组成，分别是：状态行、消息报头、响应正文

1.2.1 状态行

> 格式：HTTP-Version Status-Code Reason-Phrase CRLF
>
> 注：Reason-Phrase - 状态代码的文本描述

- 状态码
  - 1xx：指示信息–表示请求已接收，继续处理
  - 2xx：成功–表示请求已被成功接收、理解、接受
  - 3xx：重定向–要完成请求必须进行更进一步的操作
  - 4xx：客户端错误–请求有语法错误或请求无法实现
  - 5xx：服务器端错误–服务器未能实现合法的请求

1.3 报头

- 每一个报头域都是由`名字` + `冒号`（:）+ `空格` + `值`组成，消息报头域的名字是大小写无关的

1.3.1 通用头域

- **Cache-control**
- **Date** - 显示的GMT时间
- **Pragma**
- **Connection** - 连接状态

  > - **请求：**
  >
  >   `close`（告诉WEB服务器或者代理服务器，在完成本次请求的响应后，断开连接，不要等待本次连接的后续请求了）。
  >
  >   `keepalive`（告诉WEB服务器或者代理服务器，在完成本次请求的响应后，保持连接，等待本次连接的后续请求）。
  > - **响应：**
  >
  >   `close`（连接已经关闭）。
  >
  >   `keepalive`（连接保持着，在等待本次连接的后续请求）。 Keep-Alive：如果浏览器请求保持连接，则该头部表明希望 WEB 服务器保持连接多长时间（秒）。例如：`Keep-Alive：300`

1.3.2 请求头域

- **Accept-Encoding** - BD百科 - 维基
  - 当支持多种类型的Accept-Encoding时，可以设置优先级，如：Accept-Encoding: gzip;q=1.0, identity; q=0.5, *;q=0 // 按顺序支持 gzip , identity；
    - identity - 不转换内容。这是内容编码的默认值
  - 服务器将在HTTP响应中添加一个Content-Encoding或Transfer-Encoding字段表明使用的方案，用逗号分隔

    > 扩展 - **TE**：
    >
    > 网页客户端在HTTP请求的头部通告其支持的压缩方案为一个标记列表（tokens）。对于Content-Encoding，这个列表称作Accept-Encoding；对于Transfer-Encoding，该字段被称为**TE**
- **Accept-Language**

  语言缩写列表
- range

  If-Range - 条件范围请求

1.3.3 响应头域

- Accept-Ranges

如果响应头中Accept-Ranges的值不是 `none`，表示服务器支持范围请求

> 范围请求：HTTP 协议范围请求允许服务器只发送 HTTP 消息的一部分到客户端。范围请求在传送大的媒体文件，或者与文件下载的断点续传功能搭配使用时非常有用

- 范围请求的响应：
  1. 在请求成功的情况下，服务器会返回 `206 Partial Content` 状态码。
  2. 在请求的范围越界的情况下（范围值超过了资源的大小），服务器会返回 `416 Requested Range Not Satisfiable`（请求的范围无法满足） 状态码。
  3. 在不支持范围请求的情况下，服务器会返回 `200 OK` 状态码

1.4 缓存

1.4.1 概念和原理

- **ETag**是HTTP1.1中才加入的一个属性，用来帮助服务器控制Web端的缓存验证。参考博文

> **原理**：当浏览器请求服务器的某项资源(A)时，服务器根据A算出一个哈希值(3f80f-1b6-3e1cb03b)并通过 ETag 返回给浏览器，浏览器把"3f80f-1b6-3e1cb03b" 和 A 同时缓存在本地，当下次再次向服务器请求A时，会通过类似 `If-None-Match: "3f80f-1b6-3e1cb03b"` 的请求头把ETag发送给服务器，服务器再次计算A的哈希值并和浏览器返回的值做比较，如果发现A发生了变化就把A返回给浏览器(200)，如果发现A没有变化就给浏览器返回一个304未修改。这样通过控制浏览器端的缓存，可以节省服务器的带宽，因为服务器不需要每次都把全量数据返回给客户端。
>
> 注：HTTP中并没有指定如何生成ETag，哈希是比较理想的选择。

ETag配置示例

- 弱实体

```
String body = "<a href=''>点击访问当前链接</a>";

//弱实体，“W/”开头，如： W/"etag值"
String etag = "W/\"" + md5(body) + "\"";
MultiValueMap<String, String> headers = new HttpHeaders();
headers.add("ETag", etag);
```

> 此处我们使用了弱实体W\“343sda”，弱实体（“343sda”）只要内容语义没变即可，比如内容的gzip版和非gzip版可以使用弱实体验证；而强实体指字节必须完全一致（gzip和非gzip情况是不一样的），因此建议首先选择使用弱实体。

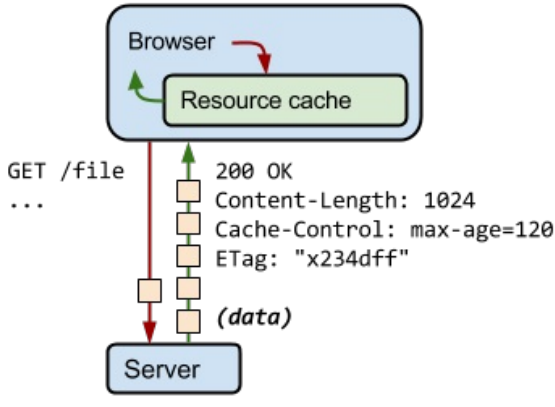
nginx在生成etag时使用的算法是Last-Modified + Content-Length计算的：

```
ngx_sprintf(etag->value.data,"\"%xT-%xO\"",

                                r->headers_out.last_modified_time,

                                r->headers_out.content_length_n)
```

- 博文 - HTTP缓存



- 流程

- 第一次请求：

- 再次请求

1.4.2 服务器配置示例

- 配置参考

1.4.3 Cache-Control

> HTTP规范允许服务器返回一系列不同的Cache-Control指令来控制浏览器或其他中继缓存会以何种方式将该响应缓存多久;
>
> Cache-Control头被定义为HTTP/1.1规范的一部分，并取代了之前用于定义响应缓存策略的头信息（比如Expires）。
>
> **Note** that HTTP/1.0 caches might not implement Cache-Control and might only implement Pragma: no-cache (see section 14.32).

1.4.3.1 缓存指令

- `no-cache` - response/request

  表示对相同的URL进行后续请求时，返回的响应在未经服务器检查其是否被修改之前，不能使用。因此，如果提交一个适当的验证令牌（ETage），no-cache会增加一个往返来验证已缓存响应，但却可以在响应未发生更改的时候，避免下载。

  - response中的no-cache可以指定不缓存的field：no-cache[="field-name"]

- `no-store` - response/request

  它直接禁止浏览器和所有的中继缓存储存任何版本的返回响应——比如包含了个人隐私或者银行信息的响应。每当用户请求这个资源，都会向服务器发送一个请求，并下载到完整的响应

- `public` - response

  响应可以被缓存，即使它有与之相连的HTTP认证，甚至响应状态代码不是正常可缓存的。大多数时候，“public”不是必需的，因为显式缓存信息（像是“max-age”）就指明了该缓存是可以缓存的

- `private` - response

  格式：private[="field-name"]

  可以由浏览器缓存，但是通常只针对单个用户，因此 `不允许` 由任何 `中继缓存` 进行保存——比如说，一个带有用户个人信息的HTML页面可以让用户浏览器缓存，但是不能让CDN缓存

- `max-age` - response/request

  缓存时长，单位秒（s）；不能设置成多于一年，因为RFC上限制了最大只能是一年，超过一年的情况不同的浏览器处理策略不同，有些直接就忽略了Cache-Control。

- `min-fresh` - request

  取 `max-age` > `当前时间` + `min-fresh` 的缓存

- `max-stale` - request

  允许获取过期时间不超过 `max-stale` 的缓存

- `must-revalidation` / `proxy-revalidation` - response

  如果缓存的内容失效，请求必须发送到服务器/代理以进行重新验证

- `no-transform` - response/request

  客户端缓存文件时，不得对实体数据做任何改变

- `only-if-cached`

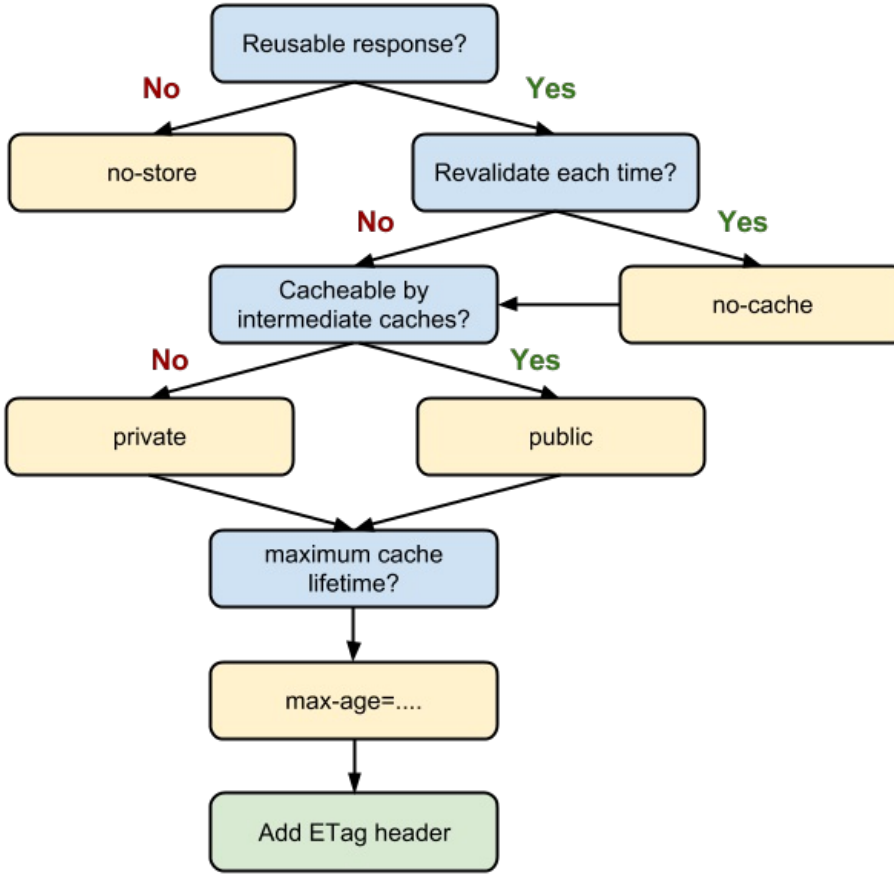
- s-maxage

```
"s-maxage" "=" delta-seconds
```

- **兼听则明**
  下图是摘自另外一篇博客

字段名称	说明
public	表明任何情况下都得缓存该资源*（即使是需要 HTTP 认证的资源）*
Private [= “field-name”]	表明返回报文中全部或部分*（若指定了 field-name 则为 field-name 的字段数据）*仅开放给某些用户*（服务器指定的 share-user，如代理服务器）*做缓存使用，其他用户则不能缓存这些数据
no-cache	不直接使用缓存，要求向服务器发起*（新鲜度校验）*请求
no-store	所有内容都不会被保存到缓存或 Internet 临时文件中
no-transform	告知客户端缓存文件时不得对实体数据做任何改变
only-if-cached	告知*（代理）*服务器客户端希望获取缓存的内容*（若有）*，而不用向原服务器发去请求
must-revalidate	当前资源一定是向原服务器发去验证请求的，若请求失败会返回 504*（而非代理服务器上的缓存）*
proxy-revalidate	与 must-revalidate 类似，但仅能应用于共享缓存*（如代理）*
max-age=delta-seconds	告知客户端该资源在 delta-seconds 秒内是新鲜的，无需向服务器发请求
s-maxage=delta-seconds	同 max-age，但仅应用于共享缓存*（如代理）*
cache-extension	自定义扩展值，若服务器不识别该值将被忽略掉

1.4.3.2 最优Cache-Control策略



1.4.3.3 缓存的缺点及解决方案

- max-age导致修改难以被及时发现
  - 可以在url中添加上文件指纹，比如：style.**x234dff**.css

1.4.3.4 其他

- 优先级：

> **Pragma -> Cache-Control -> Expires**

  - **Pragma**的优先级是高于**Cache-Control**
  - 如果Pragma头部和Expires头部同时存在，则起作用的会是Pragma
  - `Expires` 是由**服务器**定制失效时刻，**客户端**（客户端时间）判断是否失效

> Pragma ，在HTTP1.0中规定的通用首部，用来兼容只支持HTTP/1.0的服务器；它的值是空或者no-cache，空时代表可以缓存，no-cache时代表可以缓存，与Cache-Control:no-cache等价，但是当2个header同时出现时，pragma具有更高的优先级。

- expires

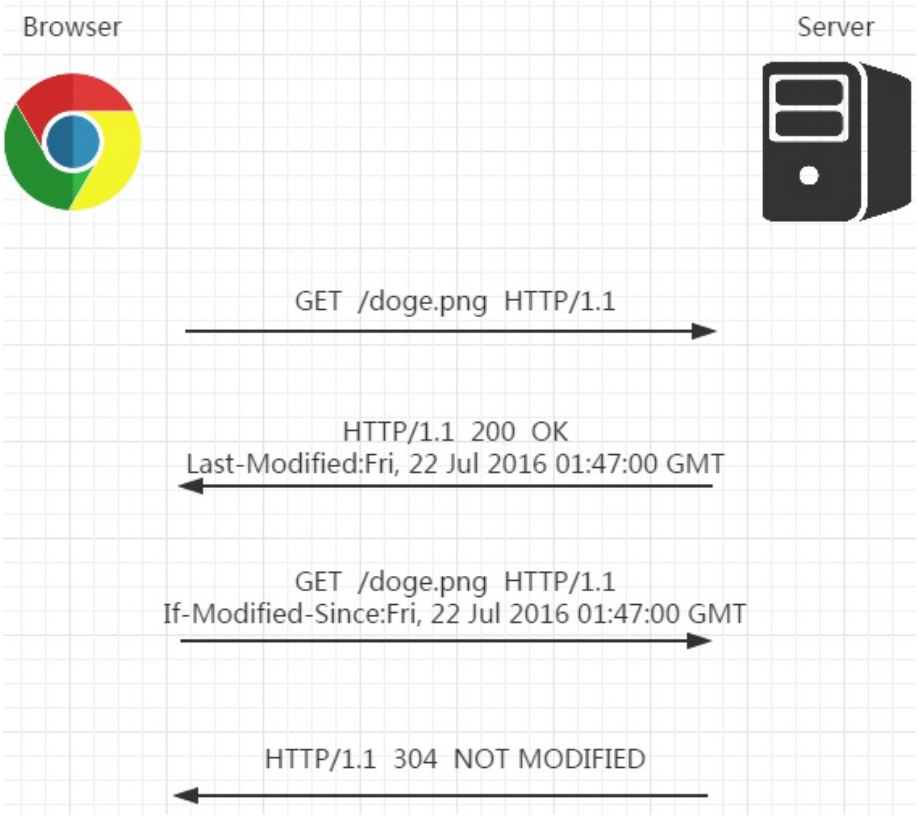
> 在直接输入地址时，若资源未到Expires指定的过期时间，则直接使用缓存，不去服务器验证。但在刷新（F5）时还是要去服务器端验证是否过期。

- Last-Modified与ETag同时使用时，浏览器在验证时会同时发送If-Modified-Since和If-None-Match，按照http/1.1规范，如果同时使用If-Modified-Since和If-None-Match则服务端必须两个都验证通过后才能返回304；且nginx就是这样做的。

1.4.4 HTML Meta标签控制缓存

了解即可

1.4.5 Last-Modified/If-Modified-Sinc的概念



- `Last-Modified`：标示这个响应资源的最后修改时间。web服务器在响应请求时，告诉浏览器资源的最后修改时间。这个是在 响应头 中设置的。

- `If-Modified-Since`：当资源过期时（使用Cache-Control标识的max-age），发现资源具有Last-Modified声明，则再次向web服务器请求时带上头 If-Modified-Since，表示请求时间。web服务器收到请求后发现有头If-Modified-Since 则与被请求资源的最后修改时间进行比对。若最后修改时间较新，说明资源又被改动过，则响应整片资源内容（写在响应消息包体内），HTTP 200；若最后修改时间较旧，说明资源无新修改，则响应HTTP 304 (无需包体，节省浏览)，告知浏览器继续使用所保存的cache。这个是在 请求头 中设置的。

- `If-Unmodified-Since`: Last-Modified-value

  该值告诉服务器，若Last-Modified没有匹配上（资源在服务端的最后更新时间改变了），则应当返回 `412` (Precondition Failed) 状态码给客户端。

1.4.6 Etag/If-None-Match

- **Etag/If-None-Match也要配合Cache-Control使用**
  - **Etag**：web服务器 响应 请求时，告诉浏览器当前资源在服务器的唯一标识（生成规则由服务器决定）。Apache中，ETag的值，默认是对文件的索引节（INode），大小（Size）和最后修改时间（MTime）进行Hash后得到的。
  - **If-None-Match**：当资源过期时（使用Cache-Control标识的max-age），发现资源具有Etage声明，则再次向web服务器 请求 时带上头If-None-Match （Etag的值）。web服务器收到请求后发现有头If-None-Match 则与被请求资源的相应校验串进行比对，决定返回200或304。
  - **If-Match: ETag-value**：告诉服务器如果没有匹配到ETag，或者收到了"*"值而当前并没有该资源实体，则应当返回 `412` (Precondition Failed) 状态码给客户端

- **既生Last-Modified何生Etag？**你可能会觉得使用Last-Modified已经足以让浏览器知道本地的缓存副本是否足够新，为什么还需要Etag（实体标识）呢？HTTP1.1中Etag的出现主要是为了解决几个Last-Modified比较难解决的问题：
  - Last-Modified标注的最后修改只能精确到秒级，如果某些文件在1秒钟以内，被修改多次的话，它将不能准确标注文件的修改时间如果某些文件会被定期生成，当有时内容并没有任何变化，但Last-Modified却改变了，导致文件没法使用缓存有可能存在服务器没有准确获取文件修改时间，或者与代理服务器时间不一致等情形
  - Etag是服务器自动生成或者由开发者生成的对应资源在服务器端的唯一标识符，能够更加准确的控制缓存。Last-Modified与ETag一起使用时，服务器会优先验证ETag。

- yahoo的Yslow法则中则提示**谨慎设置Etag**：需要注意的是 分布式系统 里多台机器间文件的last-modified必须保持一致，以免负载均衡到不同机器导致比对失败，Yahoo建议分布式系统尽量关闭掉Etag(每台机器生成的etag都会不一样，因为除了 last-modified、inode 也很难保持一致)

1.4.7 静态资源优化方案

1. 配置超长时间的本地缓存 —— 节省带宽，提高性能
2. 采用内容摘要作为缓存更新依据 —— 精确的缓存控制
3. 静态资源CDN部署 —— 优化网络请求
4. 更资源发布路径实现非覆盖式发布 —— 平滑升级

1.4.8 优缺点

头部	优势和特点	劣势和问题
Expires	1、HTTP 1.0 产物，可以在HTTP 1.0和1.1中使用，简单易用。 2、以时刻标识失效时间。	1、时间是由服务器发送的(UTC)，如果服务器时间和客户端时间存在不一致，可能会出现问题。 2、存在版本问题，到期之前的修改客户端是不可知的。
Cache-Control	1、HTTP 1.1 产物，以时间间隔标识失效时间，解决了Expires服务器和客户端相对时间的问题。 2、比Expires多了很多选项设置。	1、HTTP 1.1 才有的内容，不适用于HTTP 1.0 。 2、存在版本问题，到期之前的修改客户端是不可知的。
Last-Modified	1、不存在版本问题，每次请求都会去服务器进行校验。服务器对比最后修改时间如果相同则返回304，不同返回200以及资源内容。	1、只要资源修改，无论内容是否发生实质性的变化，都会将该资源返回客户端。例如周期性重写，这种情况下该资源包含的数据实际上一样的。 2、以时刻作为标识，无法识别一秒内进行多次修改的情况。 3、某些服务器不能精确的得到文件的最后修改时间。
ETag		

	1、可以更加精确的判断资源是否被修改，可以识别一秒内多次修改的情况。 2、不存在版本问题，每次请求都回去服务器进行校验。	1、计算ETag值需要性能损耗。 2、分布式服务器存储的情况下，计算ETag的算法如果不一样，会导致浏览器从一台服务器上获得页面内容后到另外一台服务器上进行验证时发现ETag不匹配的情况。

1.4.9 响应头字段

- **Age**

  一般用于代理层（如CDN），大家在访问京东一些页面时会发现有一个Age响应头，然后强制刷新(Ctrl+F5)后会发现其不断的变化；其表示此内容在代理层从缓存到现在经过了多长时间了，即在代理层缓存了多长时间了。

- **Vary**

  一般用于代理层（如CDN），用于代理层和浏览器协商什么情况下使用哪个版本的缓存内容（比如压缩版和非压缩版），即什么情况下后续请求才能使用代理层缓存的该版本内容，比如如下响应是告知浏览器Content-Encoding:gzip，即缓存代理层缓存了gzip版本的内容；那么后续的请求在请求时Accept-Encoding头部中包含gzip时才能使用改代理层缓存。

- **Via**

  一般用于代理层（如CDN），表示访问到最终内容经过了哪些代理层，用的什么协议，代理层是否缓存命中等等；通过它可以进行一些故障诊断。

  > via: cache24.l2cn126[59,200-0,C], cache39.l2cn126[54,0], cache6.cn598[0,200-0,H], cache4.cn598[1,0]