Effective HTTP Caching — Part II

msingh Follow
Nov 27, 2018 · 4 min read

In this second part of the series, we will do a deep dive on the `Last-Modified` HTTP header and how it impacts caching by clients.

Last-Modified

The Last-Modified entity-header field indicates the date and time at which the origin server believes the variant was last modified. Example header :

```
Last-Modified: Tue, 15 Nov 1994 12:45:26 GMT
```

Important things to note

- The date and time is as specified/set by *Origin Server (not the clients)*
- The value of the header is also a choice of Origin server. For example in case of static files, it may be the last mod time of the file on disk , for other entities , it can be driven by functional/business logic depending on update to some state of the entity etc.

How does Last-Modified header impact caching?

It works in conjunction with another header `If-Modified-Since` .

Here is what RFC says

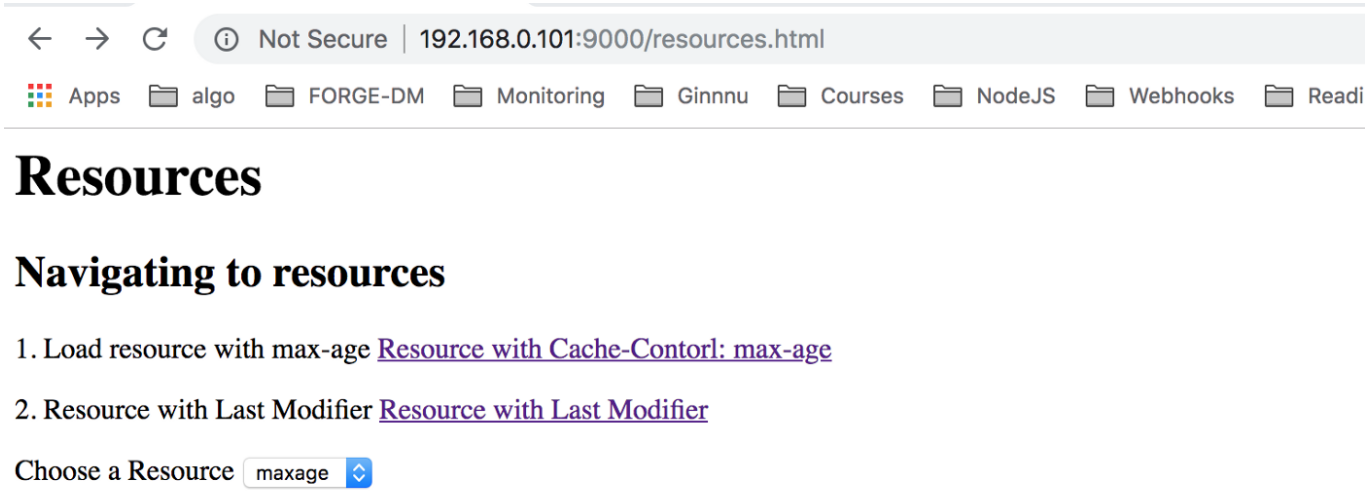
> *The If-Modified-Since request-header field is used with a method to make it conditional: if the requested variant has not been modified since the time specified in this field, an entity will not be returned from the server; instead, a 304 (not modified) response will be returned without any message-body.*

Let's explore that via a live server and Chrome browser as client.

All the code for the examples can be found at here. Follow the instructions in Readme.MD to build and run the server

Once you have the server running -

- Load the main page by going to `http://<IP|localhost>:9000/resources.html`. It will return a page like below —



The `lastmod` Resource

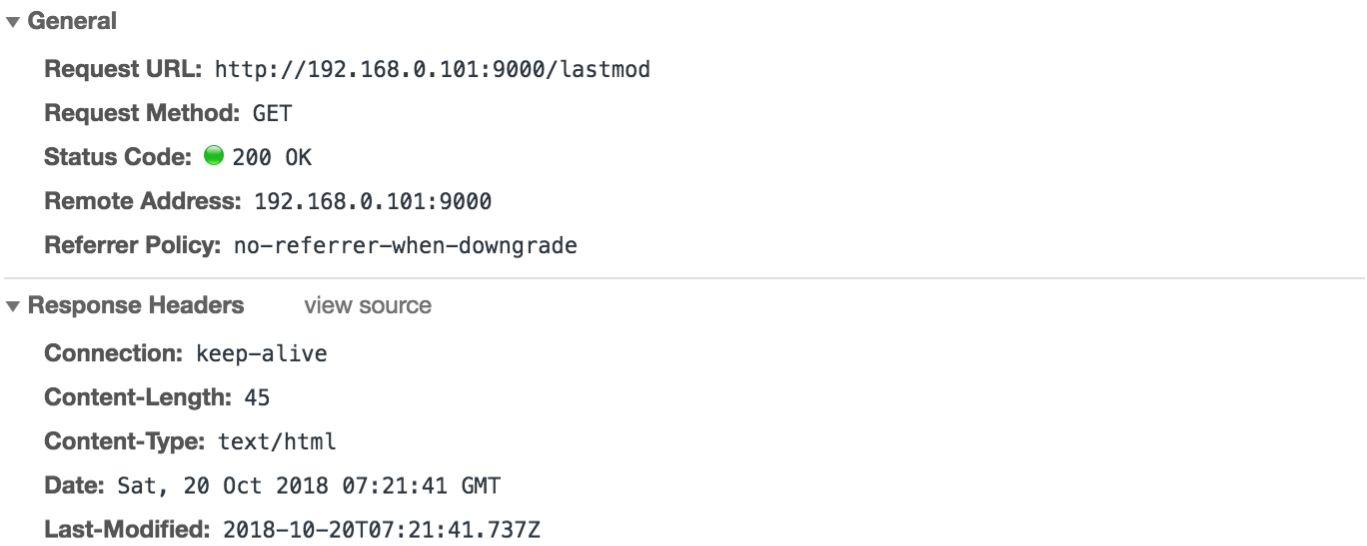
It is handled by `lastModHandler()` in the code. The resource last modified date is updated at `30` minute intervals so our goal is that for all requests that come-in during the interval should get a `304 (not modified)` response header and the whole entity would not be sent in response. This is what the code does :

- Check for `If-Modified-Since` header in the incoming request.
- If the `lastModDate` of the resource is less or equal to the `If-Modified-Since` header then we know that the cache is still fresh and we return 304

- If not, return the resource with `200 OK` status code

Here is how the browser requests look

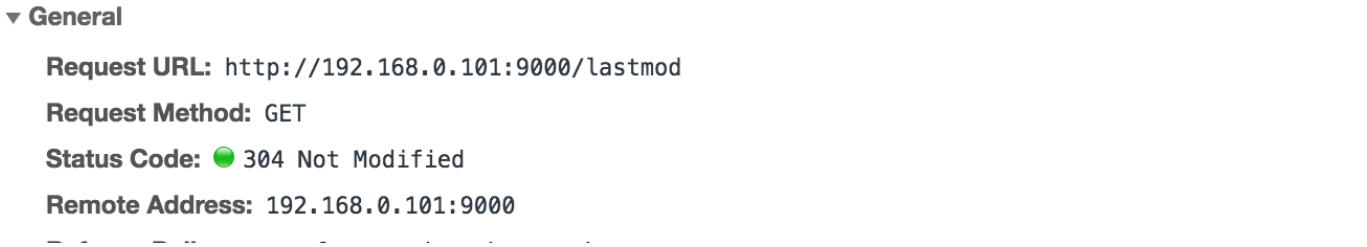
First request :

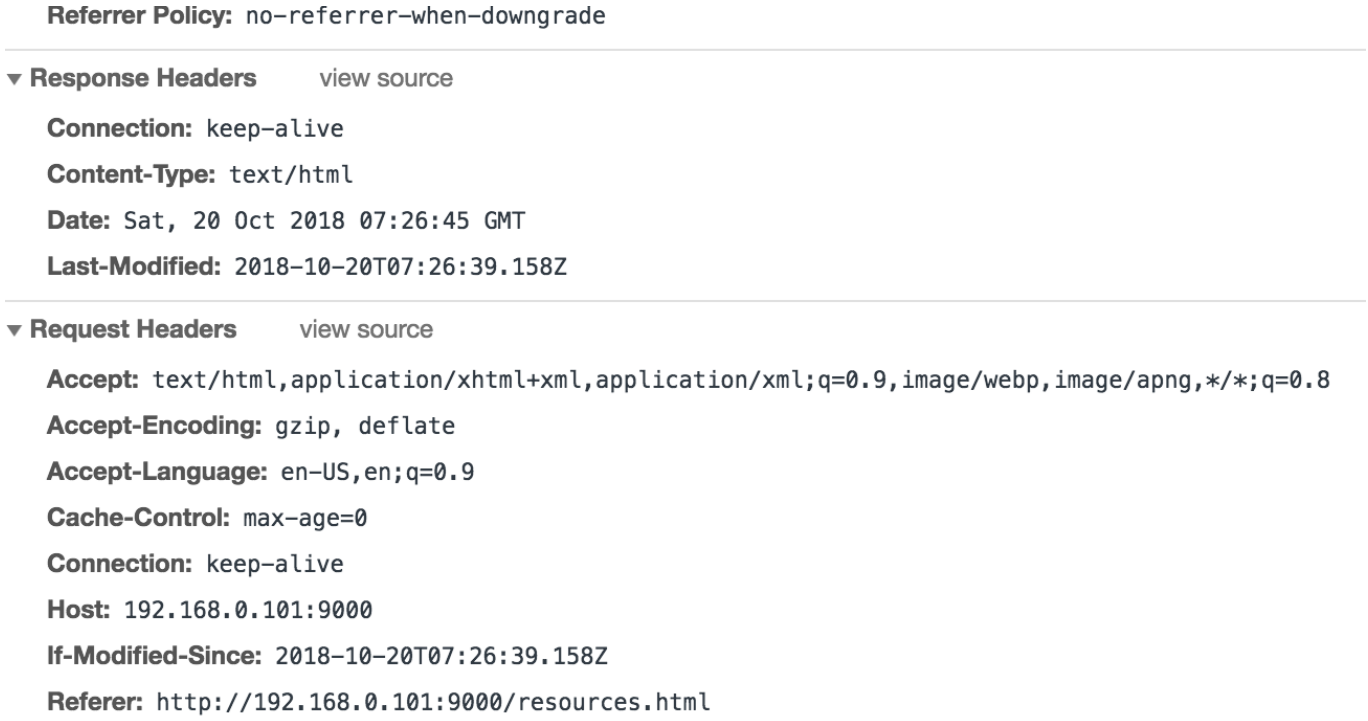


Since this is not a conditional request , we return the resource with 200 OK

Now, refresh the page `http://<IP>:9000/lastmod` or use the dropdown to select `lastmod` on the resources page.

You will see the following request headers





The server log explains why :

> *If modified since header is 1540020399158, Last Mod time for resource is 1540020399158*
>
> *cache is still fresh !, return 304 NOT MODIFIED*

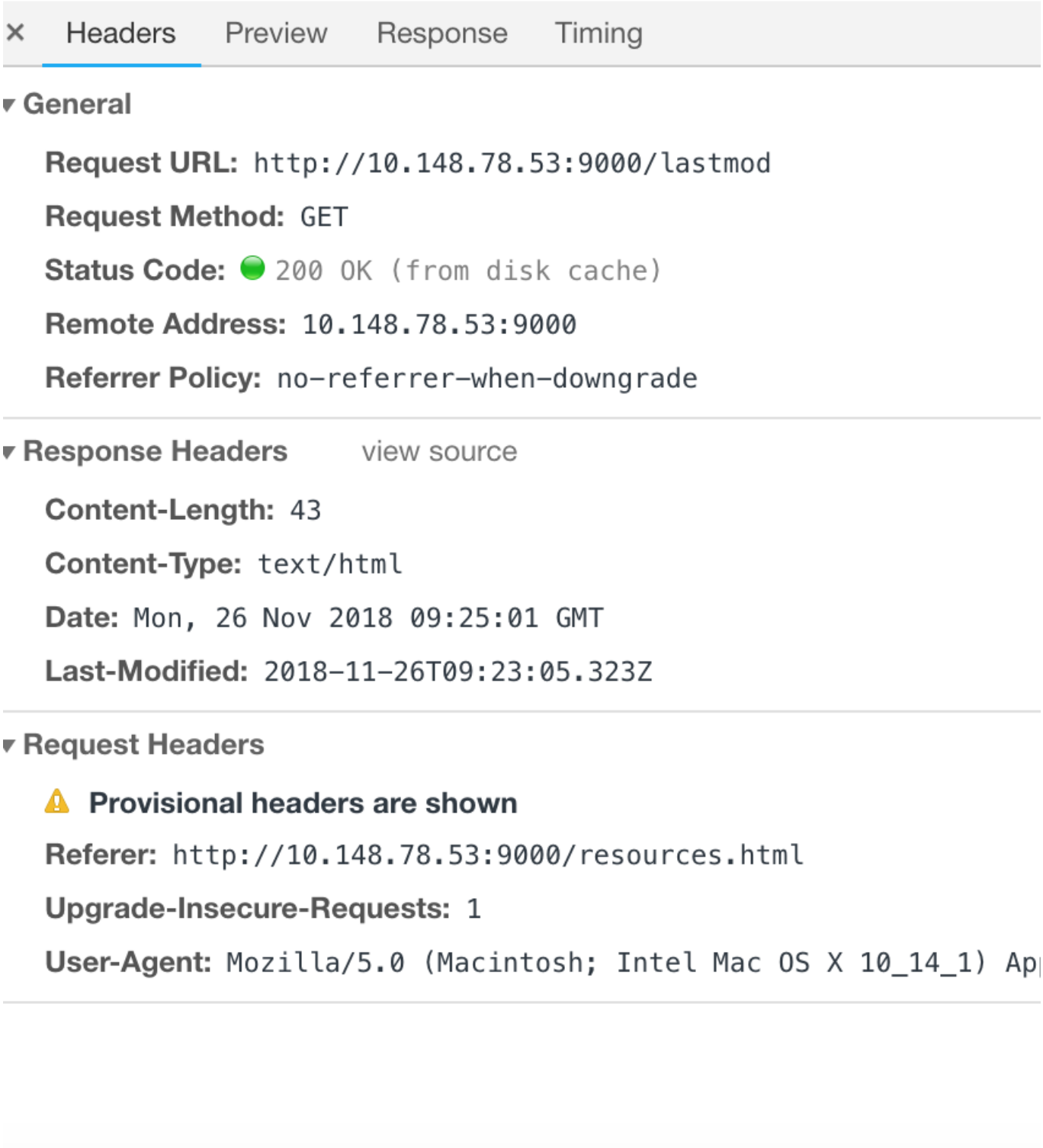
Important

- Recall that the `Last-Modified` time is set by Server. And note that `If-Modified-Since` time is interpreted by the server, whose clock might not be synchronized with the client. So the best strategy for client is to simply reuse the exact value of `Last-Modified` header when making conditional request to remove any ambiguity or clock issues

The Heuristic Freshness Gotcha

RFC7234 allows for something called as “heuristic freshness” for cached responses. Detailed explanation in Section 4.2.2 of the RFC. Simply put, if the response from the Server has a `Last-Modified` header but doesn’t contain any other directive, either an explicit expiration time or a cache control directive which forces Cache to re-validate its request with the Origin Server, then a certain percentage of requests may be simply served by Cache.

If you load `http://<IP>:9000/lastmod` a few times, intermittently you will see the following response from Browser. Note that the response code is 200 instead of 304 and the request was never sent to the Server in this case.



In general, I found this behavior non-intuitive. The best way to avoid this and have the Browser always validate with the Server, is to add the `Cache-Control` directive.

Cache-Control Header

This directive controls how, and for how long, the browser and other intermediate caches can cache the response. Returning `Cache-Control: no-cache` header with the response means Browser can't use the cached response to satisfy a subsequent request without successful revalidation with the origin server.

So, Un-comment the following line in Server code (see comments above) , and re-running the server you will see that the browser no longer applies heuristics for freshness calculation

```
response.setHeader('Cache-Control', 'no-cache');
```

In the **next** part of this series, we will look at some more Cache-Control directives.

You can also explore read about this header here :
https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9

Thanks for reading !

Web Development | Http Request | Browser Cache | Cache Control | Rest Api