Increasing Application Performance with HTTP Cache Headers

Last updated 17 January 2019

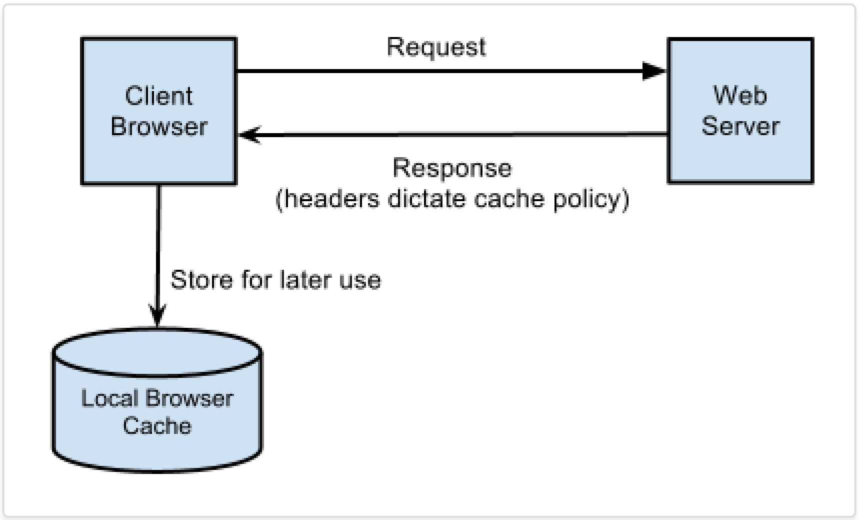
Table of Contents

The modern day developer has a wide variety of techniques and technologies available to improve application performance and end-user experience. One of the most frequently overlooked technologies is that of the HTTP cache.

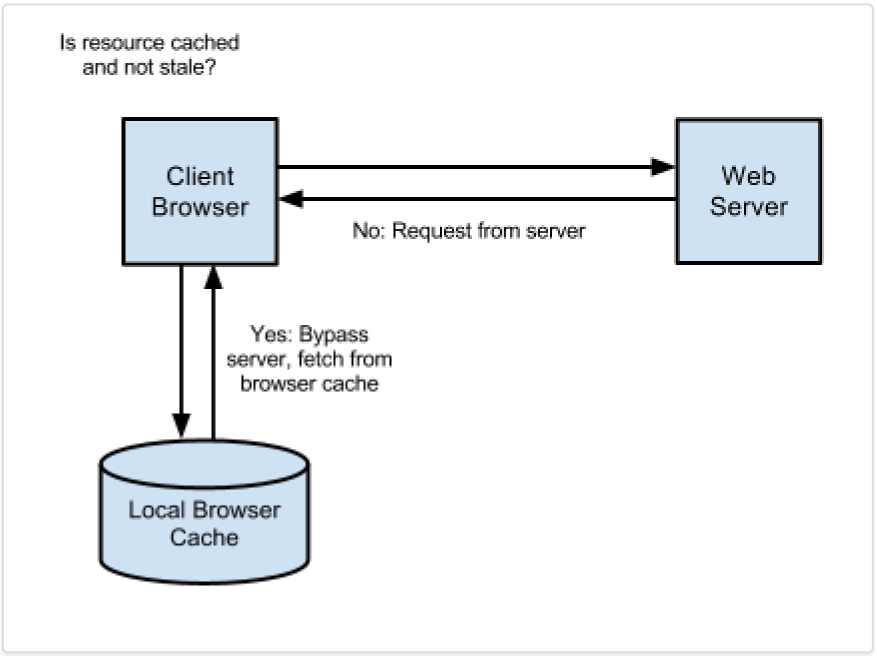
HTTP caching is a universally adopted specification across all modern web browsers, making its implementation in web applications simple. Appropriate use of these standards can benefit your application greatly, improving response times and reducing server load. However, incorrect caching can cause users to see out-of-date content and hard to debug issues. This article discusses the specifics of HTTP caching and in what scenarios to employ an HTTP cache header based strategy.

Overview

HTTP caching occurs when the browser stores local copies of web resources for faster retrieval the next time the resource is required. As your application serves resources it can attach cache headers to the response specifying the desired cache behavior.



When an item is fully cached, the browser may choose to not contact the server at all and simply use its own cached copy:


Is resource cached and not stale?
Client Browser
Web Server
No: Request from server
Yes: Bypass server, fetch from browser cache
Local Browser Cache

For instance, once CSS stylesheets from your application are downloaded by the browser there's no need to download them again during the user's session. This holds true for many asset types like javascript files, images and even infrequently changing dynamic content. In these instances it is beneficial for the users browser to cache this file locally, and use that copy whenever the resource is requested again. An application using HTTP cache headers is able to control this caching behavior and alleviate server-side load.

It is intuitive to think about the end-user's browser as the primary consumer of HTTP cache headers. However, these HTTP cache headers are available to be, and are acted upon by, every intermediate proxy and cache between the source server and the end user.

HTTP cache headers

There are two primary cache headers, `Cache-Control` and `Expires`.

Cache-Control

Without the cache-control header set, no other caching headers will yield any results.

The `Cache-Control` header is the most important header to set as it effectively 'switches on' caching in the browser. With this header in place, and set with a value that enables caching, the browser will cache the file for as long as specified. Without this header the browser will re-request the file on each subsequent request.

`public` resources can be cached not only by the end-user's browser but also by any intermediate proxies that may be serving many other users as well.

```
Cache-Control:public
```

`private` resources are bypassed by intermediate proxies and can only be cached by the end-client.

```
Cache-Control:private
```

The value of the `Cache-Control` header is a composite one, indicating whether the resource is public or private while also indicating the maximum amount of time it can be cached before considered stale. The `max-age` value sets a timespan for how long to cache the resource (in seconds).

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

While the `Cache-Control` header turns on client-side caching and sets the `max-age` of a resource the `Expires` header is used to specify a specific point in time the resource is no longer valid.

Expires

When accompanying the `Cache-Control` header, `Expires` simply sets a date from which the cached resource should no longer be considered valid. From this date forward the browser will request a fresh copy of the resource. Until then, the browsers local cached copy will be used:

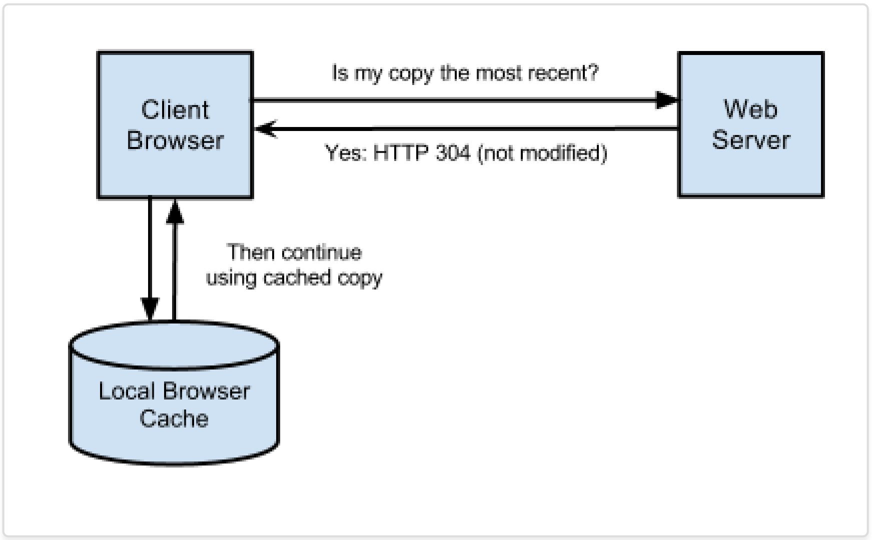
> If both `Expires` and `max-age` are set `max-age` will take precedence.

```
Cache-Control:public
Expires: Mon, 25 Jun 2012 21:31:12 GMT
```

While `Cache-Control` and `Expires` tells the browser *when* to next retrieve the resource from the network a few additional headers specify *how* to retrieve the resource from the network. These types of requests are known as conditional requests.

Conditional requests

Conditional requests are those where the browser can ask the server if it has an updated copy of the resource. The browser will send some information about the cached resource it holds and the server will determine whether updated content should be returned or the browser’s copy is the most recent. In the case of the latter an HTTP status of 304 (not modified) is returned.


Client Browser
Is my copy the most recent?
Yes: HTTP 304 (not modified)
Web Server
Then continue using cached copy
Local Browser Cache

Though conditional requests do invoke a call across the network, unmodified resources result in an empty response body – saving the cost of transferring the resource back to the end client. The backend service is also often able to very quickly determine a resource's last modified date without accessing the resource which itself saves non-trivial processing time.

Time-based

A time-based conditional request ensures that only if the requested resource has changed since the browser's copy was cached will the contents be transferred. If the cached copy is the most up-to-date then the server returns the 304 response code.

To enable conditional requests the application specifies the last modified time of a resource via the `Last-Modified`response header.

```
Cache-Control:public, max-age=31536000
Last-Modified: Mon, 03 Jan 2011 17:45:57 GMT
```

The next time the browser requests this resource it will only ask for the contents of the resource if they're unchanged since this date using the `If-Modified-Since` *request* header

```
If-Modified-Since: Mon, 03 Jan 2011 17:45:57 GMT
```

If the resource hasn't changed since `Mon, 03 Jan 2011 17:45:57 GMT` the server will return with an empty body with the `304` response code.

Content-based

The `ETag` (or Entity Tag) works in a similar way to the `Last-Modified` header except its value is a digest of the resources contents (for instance, an MD5 hash). This allows the server to identify if the cached contents of the resource are different to the most recent version.

This tag is useful when for when the last modified date is difficult to determine.

```
Cache-Control:public, max-age=31536000
ETag: "15f0fff99ed5aae4edffdd6496d7131f"
```

On subsequent browser requests the `If-None-Match` *request* header is sent with the ETag value of the last requested version of the resource.

```
If-None-Match: "15f0fff99ed5aae4edffdd6496d7131f"
```

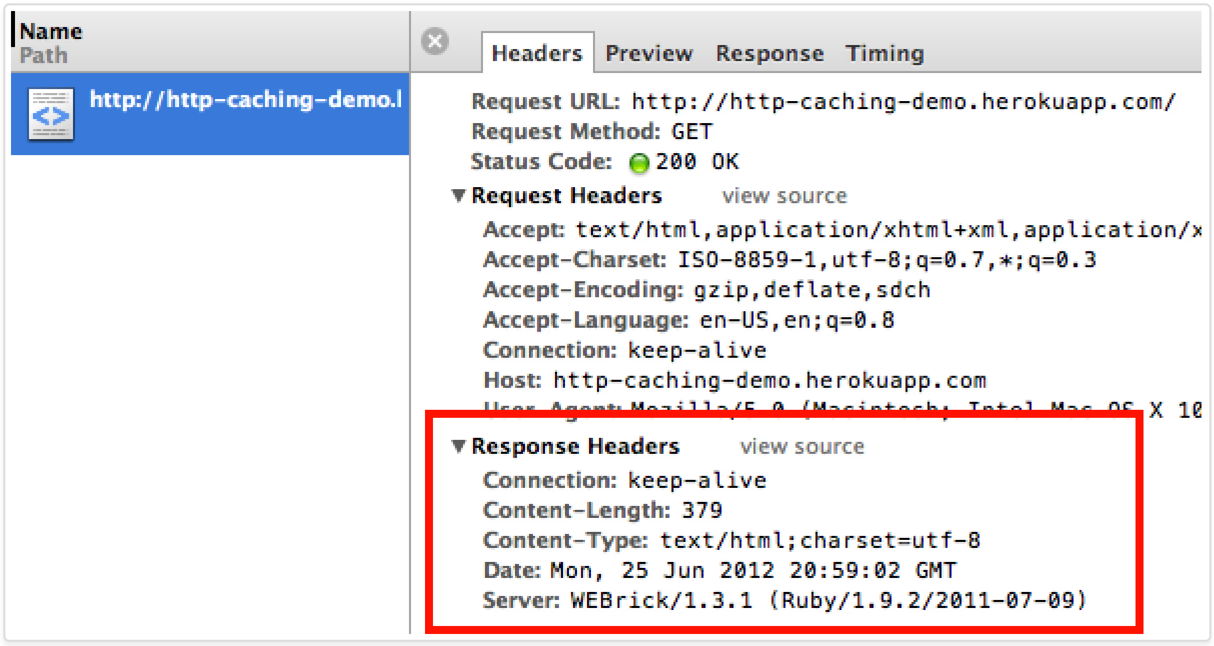
As with the `If-Modified-Since` header, if the current version has the same ETag value, indicating its value is the same as the browser's cached copy, then an HTTP status of 304 is returned.

Visibility

Most modern browsers include robust request/response visualization and introspection tools. The Web Inspector found in both Chrome and Safari shows the response and request headers in the 'Network' tab.

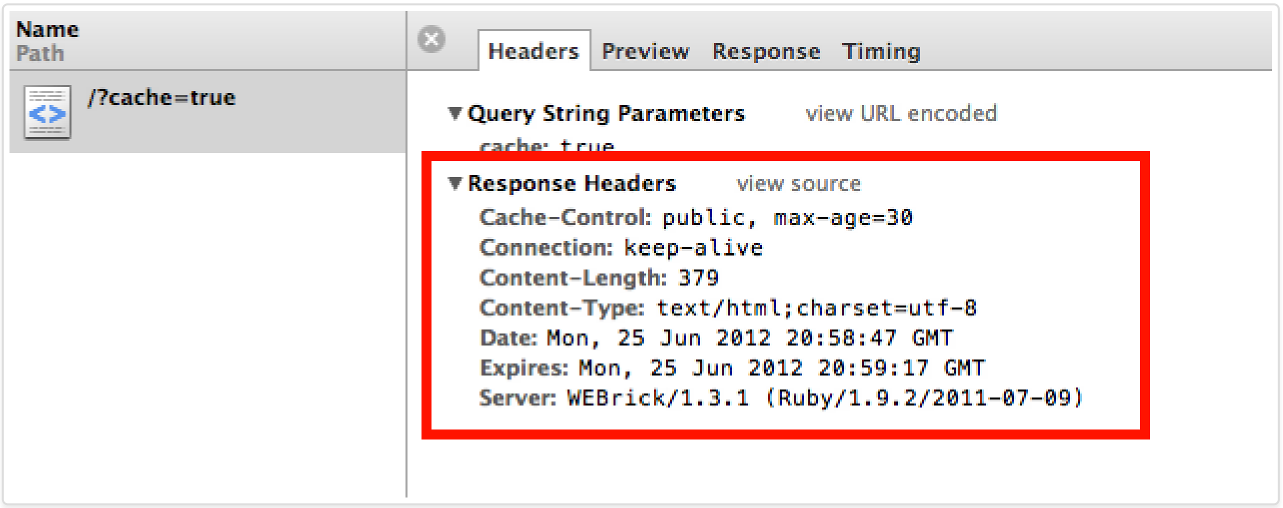
Code for a sample application that uses the various cache headers can be found **on GitHub** and can be seen running at **http://http-caching-demo.herokuapp.com/**.

An initial request to http://http-caching-demo.herokuapp.com/ shows the default set of headers returned by the application (with no cache directives).



```
Request URL: http://http-caching-demo.herokuapp.com/
Request Method: GET
Status Code: 200 OK
▼Request Headers    view source
  Accept: text/html,application/xhtml+xml,application/x
  Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
  Accept-Encoding: gzip,deflate,sdch
  Accept-Language: en-US,en;q=0.8
  Connection: keep-alive
  Host: http-caching-demo.herokuapp.com
  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10
▼Response Headers    view source
  Connection: keep-alive
  Content-Length: 379
  Content-Type: text/html;charset=utf-8
  Date: Mon, 25 Jun 2012 20:59:02 GMT
  Server: WEBrick/1.3.1 (Ruby/1.9.2/2011-07-09)
```

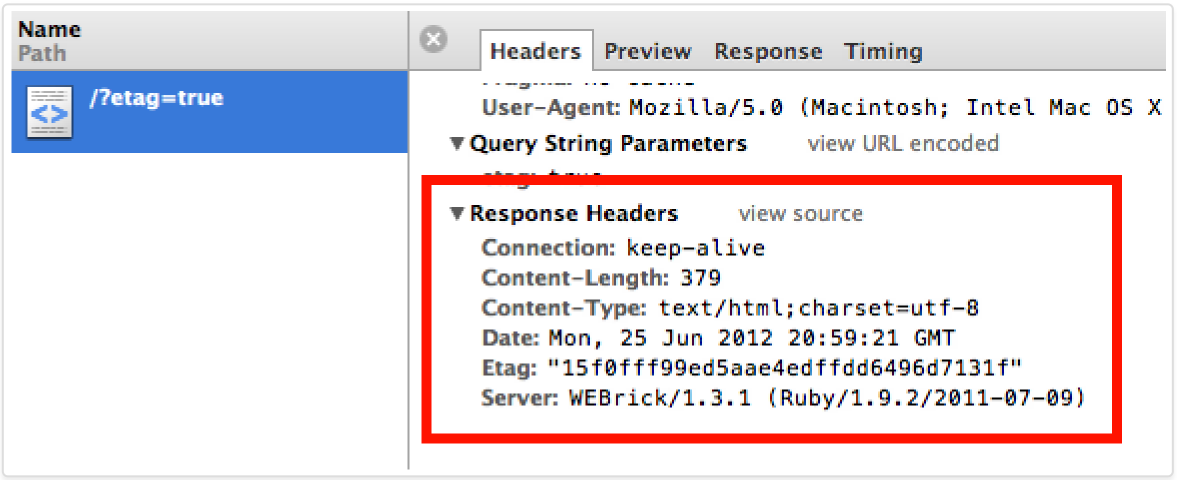
By adding the `cached` query parameter, http://http-caching-demo.herokuapp.com/?cache=true, the application turns on caching with both `Cache-Control` and `Expires` headers (both of which are set to 30 seconds in the future).


Name
Path
/?cache=true

Headers Preview Response Timing

```
▼Query String Parameters      view URL encoded
  cache: true
▼Response Headers      view source
  Cache-Control: public, max-age=30
  Connection: keep-alive
  Content-Length: 379
  Content-Type: text/html;charset=utf-8
  Date: Mon, 25 Jun 2012 20:58:47 GMT
  Expires: Mon, 25 Jun 2012 20:59:17 GMT
  Server: WEBrick/1.3.1 (Ruby/1.9.2/2011-07-09)
```

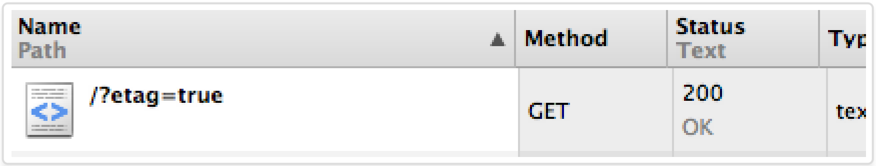
Adding an `etag` parameter to the request, http://http-caching-demo.herokuapp.com/?etag=true, causes the sample app to specify the ETag digest of the JSON contents.


Name
Path
/?etag=true

Headers Preview Response Timing

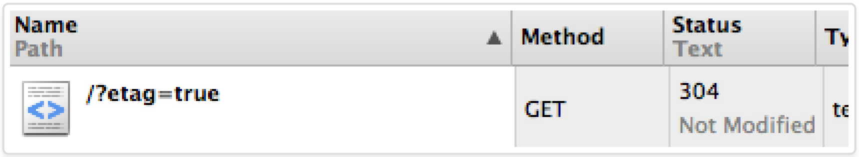
```
  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
▼Query String Parameters      view URL encoded
  etag: true
▼Response Headers      view source
  Connection: keep-alive
  Content-Length: 379
  Content-Type: text/html;charset=utf-8
  Date: Mon, 25 Jun 2012 20:59:21 GMT
  Etag: "15f0fff99ed5aae4edffdd6496d7131f"
  Server: WEBrick/1.3.1 (Ruby/1.9.2/2011-07-09)
```

On deeper inspection the ETag-based conditional request executes as expected. With the initial request the browser can be seen downloading the file from the server:



Name Path	Method	Status Text	Typ
/?etag=true	GET	200 OK	tex

However, on subsequent requests you are able to see the server responding to the browsers ETag check with a HTTP status of 304 (not modified) which causes the browser to use its own cached copy:



Name Path	Method	Status Text	Ty
/?etag=true	GET	304 Not Modified	te

Use-cases

Static assets

Under normal usage, the starting point for any developer should be to add as an aggressive caching strategy to the files in the application that will not change. Normally this will include static files that are served by the application such as images, CSS file and Javascript files. As these files are typically re-requested on each page, a large performance improvement can be had with little effort.

In these instances, you should set the Cache-Control header, with a max-age value of a year in the future from the time of the request. It is recommended that Expires should be set to a similar value.

1 year is 31536000 seconds

```
Cache-Control:public; max-age=31536000
Expires: Mon, 25 Jun 2013 21:31:12 GMT
```

It is not generally a good idea to go any further than this as greater time periods are not supported by the RFC and may be ignored.

Dynamic content

Dynamic content is much more nuanced. For each and every resource, the developer must assess how heavily it can be cached and what the implications might be of serving stale content to the user. Two examples would be the contents of a blog RSS feed (which will not change more than once every few hours), to the JSON packets which drive a user’s Twitter timeline (updating once every few seconds). In these cases it would be reasonable to cache the resources for as long as you believe possible without causing issues for the end user.

Private content

Private content (ie. that which can be considered sensitive and subject to security measures) requires even more assessment. Not only do you as the developer need to determine the cacheability of a particular resource, but you also need to consider the impact of having intermediary caches (such as web proxies) caching the files which may be outside of the users control. If in doubt, it is a safe option to not cache these items at all.

Should end-client caching still be desirable you can ask for resources to only be cached privately (i.e only within the end-user’s browser cache):

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

Cache prevention

Highly secure or variable resources often require no caching. For instance, anything involving a shopping cart checkout process. Unfortunately, merely omitting cache headers will not work as many modern web browsers cache items based on their own internal algorithms. In such cases it is necessary to tell the browser to explicitly to not cache items.

In addition to `public` and `private` the `Cache-Control` header can specify `no-cache` and `no-store` which informs the browser to not cache the resources under any circumstances.

> Both values are required as IE uses `no-cache`, and Firefox uses `no-store`.

```
Cache-Control:no-cache, no-store
```

Implementation

Once the concepts behind HTTP caching are understood the next step is to implement them in your application. Most modern web frameworks make this a trivial task.

Language/framework	Tutorials
Ruby/Rails	• HTTP Caching in Ruby with Rails • Using Rack::Cache with Memcached to provide an HTTP cache store
Java	• HTTP Header Caching in JAX-RS