HTTP Headers

Both HTTP requests and responses have *Name*: *Value* header lines followed pairs followed by an empty line. Extensible via `X-`headers.

- Content type and MIME types.
- Content negotiation.
- Host, referer, content-length.
- Keep-alive and persistent connections.
- Caching.

Playing with Headers

- This simple *ruby script* echoes all HTTP requests (including headers and body) back to the client.
- It is currently set up to run on `http://zdu.binghamton.edu:2000`.
- Run using *Restlet Chrome Client*.

Content Type

`Content-Type:` *MIME-type*`;` `charset=`*char-set*

char-set Character encoding used.

MIME-type Specifies type of content.

Can also have `boundary=`*Boundary* to distinguish parts of a multi-part response. *Boundary* is a string of *robust* characters starting with `--` which delimits end of message body.

MIME Type

- MIME stands for **Multimedia Mail Extensions**.
- Consists of `type/subtype`.
- Examples:

`text/plain` Default text, displayed as is.

`text/css` Used for CSS stylesheets.

`text/html` Used for HTML.

`image/gif`, `image/jpeg`, `image/png`, `image/svg+xml` Image formats.

`multipart/form-data` Can be used for submitting form data.

MIME Type

Main MIME `type`'s:

text	Human readable text.
image	Any kind of image, not including video.
audio	Audio file.
video	Video file.
application	Binary data.
multipart	Multipart documents broken into distinct parts possibly with distinct MIME types.

Character Encodings

Character encodings map abstract character sequences into a bit sequence. Content character set specified by `charset` attribute in `Content-Type` header. Example character encodings:

ASCII `charset=us-ascii` 7-bit fixed length encoding.

ISO 8859-1 `charset=iso-8859-1` fixed length encoding for 191 characters from the Latin script *Latin alphabet no. 1*. This is the default `charset`.

UTF-8 `charset=utf-8` Variable length encoding of Unicode character set. Uses one to four 8-bit bytes. Backward compatible with ASCII. Default encoding on modern Unix systems.

UTF-16 `charset=utf-16` Variable length encoding of Unicode character set; uses 2 or 4 bytes. Byte order matters; variants `utf-16le` and `utf-16be` are more specific. Used by modern Windows.

Confusable Characters

- Confusable characters often form the basis for incorrect URLs used in phishing attacks; for example, confusing `1` for `l` and `0` for `O`.
- Since Unicode encodes characters, not glyphs, there are many distinct characters which have similar looking glyphs. For example, Latin *B* and Greek Beta *B* look the same but have lowercase Latin *b* and Greek beta β.
- Another source of confusion is that some characters have adornments; for example, dotted versus undotted-I in Turkish.

Content Negotiation

- Obtain different representations of the same resource using *content negotiation*.
- Client sends headers describing types of content it can accept.
- Server chooses appropriate response type.

Content Negotiation Headers

`Accept` MIME-types accepted. `Accept: text/*, image/png, image/gif`.

`Accept-Charset` Character sets accepted. `Accept-Charset: ISO-8859-1`. Abandoned in favor of default UTF-8.

`Accept-Language` Accepted human languages. `Accept-Language: en, de`.

`Accept-Encoding` Different encodings. `Accept-Encoding: compress, gzip`.

`User-Agent` Specifies the browser as in `Firefox/35.0.1`. Not formally meant for content-negotiation but often used to serve different representations to desktop vs mobile browsers.

Host Header

`Host:` *DNS-name*

- Mandatory header for HTTP/1.1 requests. Results in 400 `BAD REQUEST` status if not sent.
- *DNS-name* is the name of the server host.
- Makes it possible for the same IP address to be used for multiple domains. This is often referred to as *virtual hosts*.

Referer Header

`Referer:` *URL*

- Provides *URL* of resource which made this request.
- Not sent for a HTTP request from a HTTPS resource.
- Can be a privacy problem.

Content Length

`Content-Type:` *Length*

- *Length* is the number of bytes in the message body.
- Important to allow receiver to figure out the end of the body.
- Can be difficult to generate for dynamic responses as it requires buffering the entire response until content length can be figured out.

Keep-Alive Connections

- HTTP / HTTPS run over network protocol TCP/IP.
- TCP/IP has high overhead of handshakes to set up (*slow-start*) and tear down connections.
- HTTP 1.0 and early versions of HTTP 1.1 had keep-alive connections. Request **must** include:

  ```
  Connection: Keep-Alive
  ```

  If honored by server, then response includes same header.
- Connections are kept alive only if requested, default is to tear-down after each request-response pair.
- On a keep-alive connection, if must be possible for the receiver to figure out the end of a message body; hence needs something like a `Content-Length` header.

Persistent Connections

- HTTP/1.1 deprecated keep-alive connections in favor of default **persistent connections**.
- In HTTP/1.1, client assumes that connection stays up until it receives a `Connection: close` header in response.
- When client wants to ensure that connection is closed, it should send `Connection: close` header with last request.
- Same restriction as keep-alive in necessity of determining end of message bodies.

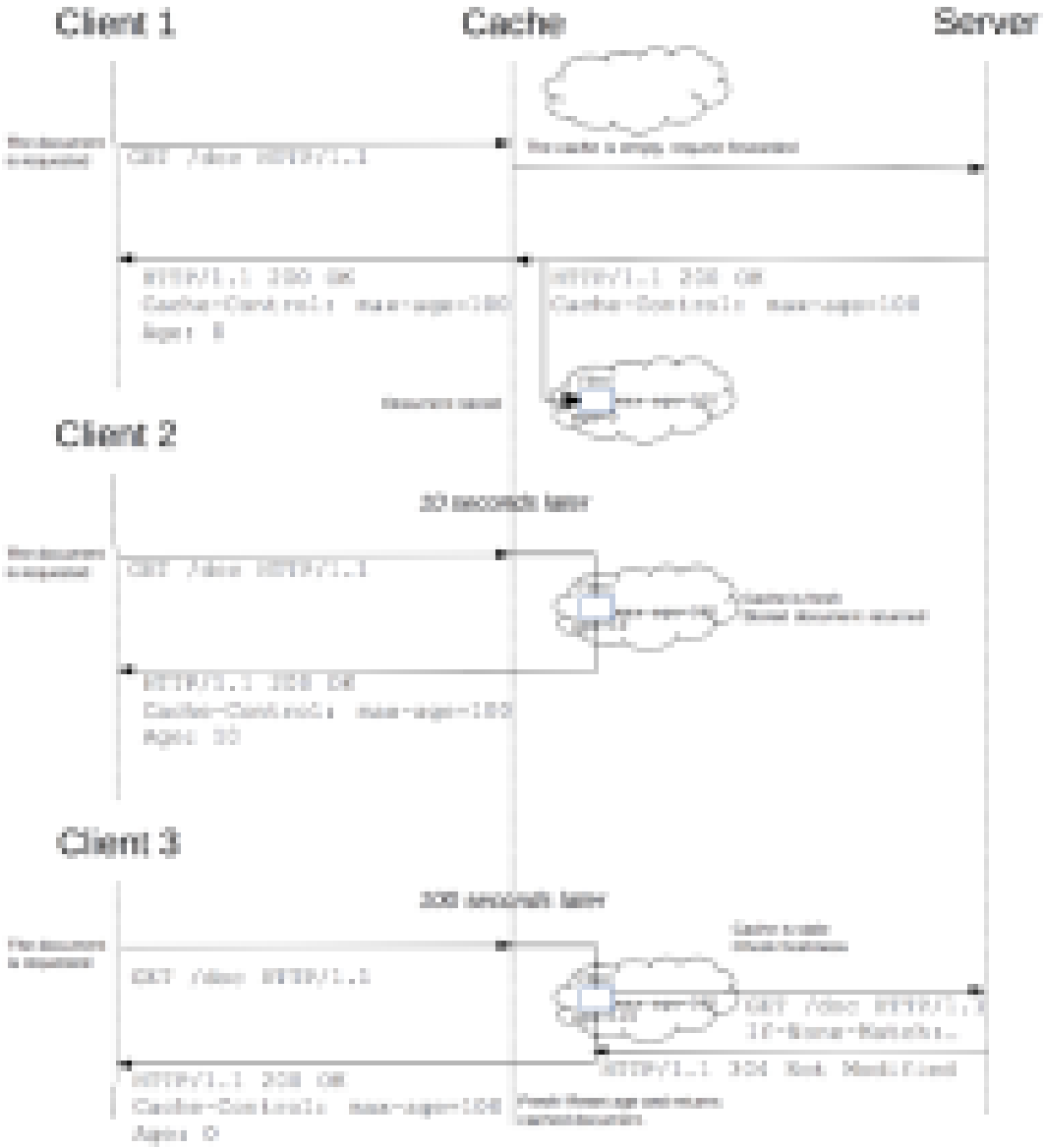
Caching Overview

Responses to specific URLs cached in intermediate stores:

- Motivation: improve performance by reducing response time and network bandwidth.
- Ideally, subsequent request for the **same** URL should be served entirely from cache without server being contacted at all.
- If not possible to serve request from cache, it is better to only have a validation request sent to the server. A successful validation would allow serving cached response.
- Full request-response transmission between client and server only when resource is not cached, or validation fails.
- Caches consists of a map from method-URL keys to the content (in practice, many caches only cache `GET`'s and simply use the URL as a key).

Cache Interaction Diagram

MDN Caching Tutorial



Terminology

User Agent The client which initiates a request. May be a browser, spider or web-service client.

Origin Server The server on which a resource resides.

Equivalent Responses Two responses are *bitwise equivalent* if their bodies (and selected headers) are bitwise identical. They are *semantically equivalent* if there is no difference between the responses with respect to application semantics.

Expiration Time, Freshness Lifetime The time for which a response is regarded as valid (*expiration*: absolute time, *freshness*: relative to response generation time).

Stale Response is stale if past its expiry time. Caches should return stale responses only if disconnected from origin server (must include a `Warning` header).

Validation The process used by a cache to check whether a cached entity is still valid.

Types of Caches

Private Cache A cache dedicated to a single user.

Shared Cache A cache which stores responses shared among multiple users.

User Agent Cache A **private** cache maintained by the user agent. AKA *browser cache*.

Intermediate Cache A **shared** cache between the origin server and user agent. This kind of cache may be operated without any involvement by either the client or server organizations.

Gateway Cache A **shared** cache operated by the server organization. It involves setting up the server DNS so that requests pass through the shared cache. **Content Delivery Networks** provide gateway caches which are distributed geographically across the Internet. Example CDN's include Akamai and Cloudflare.

Cacheable HTTP Methods

- Responses to `GET` and `HEAD` requests are cacheable by default.
- To cope with misuse of HTTP methods, responses to `GET` requests to a URL with query parameters are not cacheable.
- Responses to other methods are not cacheable by default.
- In fact, if a cache notices non-error response beging returned for a `PUT`, `POST` or `DELETE` method to a URL which it has cached, then it is required to invalidate that cache entry.
  Caches are required to **write-through** any unsafe requests to the origin server.
  No guarantee that some other cache may still have that entry cached. Absolutely necessary to configure responses so that this cannot happen.

Cache-Control Header on Responses

Modern way of describing caching policy for a resource. Header consists of `Cache-Control:` label followed by one-or-more comma separated cache-control directives:

`max-age` The max time (in seconds) after which a response will be considered stale.

`must-revalidate` Do not return stale responses without revalidation.

`no-cache` Never return cached response without revalidation.

`no-store` Never store any part of the request or response. Used for private sensitive data. Not sufficient to ensure security.

`public` May be stored on shared caches.

`private` Must not be stored on shared caches.

May also be used in requests, check docs for exact semantics.

Example Cache Control Headers

`Cache-Control: no-cache, no-store, must-revalidate`
: Prevent caching.

`Cache-Control: public, max-age=86400` Cache static assets for 1 day.

`Cache-Control: private, max-age=900` Return response from private cache for up to 15 minutes. No revalidation necessary.

`Cache-Control: private, max-age=86400, no-cache` Cache for 1 day only in private caches but revalidate before returning cached responses.

Other Cache Control Methods

`Expires` Header Similar to Cache-Control `max-age` directive, but uses an absolute expiration date-time (must be in UTC). Still widely used as it will likely be recognized by HTTP/1.0 caches, but Cache-Control has much more flexibility and should be used by newer web sites.

`Pragma` Header A HTTP/1.0 header with value `no-cache`. Similar to `Cache-Control: no-cache`. Use only for backward compatibility with HTTP/1.0.

HTML meta http-equiv Tags put inside HTML. Usually will not work as it requires caches to parse HTML which they will not do.

Conditional Get

- A *conditional get* is simply a `GET` request with a header which requires the origin server to check a condition.
- If the condition is **true**, then the response will include a body.
- If the condition is **false**, the response will simply consists of headers without a response body.
- Improves performance by not transmitting response body unless absolutely necessary.

Validators

Validators are simply header values returned along with a response. Two kinds of validators:

Strong Validators If the resource (entity body or any entity headers) changes in any way, then the value of a strong validator must change.

Weak Validators The validator value may not change even if the entity changes.

Example Validator Headers

Last-Modified Time resource was last modified on origin server. If static file then time of last file modification; if dynamic resource, then latest time of last modification of any component.

Etag An opaque string generated by origin server; string changes iff resource content changes.

Conditional Get Based on Modification Time

- When a response is returned, it contains a `Last-Modified` header containing the GMT time specifying when the resource was last modified on the origin server. If the resource is represented by a static file, then the `Last-Modified` time can be time of last file modification; if the resource is built dynamically, then latest time of last modification of any component.
- When a cache wants to validate a cache entry, it send a request with a `If-Modified-Since` header containing the value of the cached `Last-Modified` header.

Conditional Get Based on Modification Time Continued

- If the content has not changed since the time in the Last-Modified value, the server responds with a 304 `NOT MODIFIED` status **without a response body**.
- If the content has changed since the time in the Last-Modified value, the server responds with a 200 status. The response will have a body containing the representation of the changed resource.
- Not reliable, since granularity of time is 1 second.

Conditional Get Based on Entity Tags

- When a response is returned, it contains a `ETag` header containing an opaque string which serves as an identity for a specific version of a resource. The identity string should change whenever the resource changes. Can be implemented as a hash over the contents, or simply an integer which is incremented each time the resource changes.
- When a cache wants to validate a cache entry, it send a request with a `If-None-Match` header containing the value of the cached `ETag` header.

Conditional Get Based on Entity Tags Continued

- If the resource still has the same `ETag` value, the server responds with a 304 `NOT MODIFIED` status **without a response body**.
- If the resource has a different `ETag` value, the server responds with a 200 status. The response will have a body containing the representation of the changed resource.
- Reliable if any bitwise change in the resource results in a different `ETag` value. Can also be used to implement semantic equality rather than bitwise equality.
- Preferred over validation based on `Last-Modified` header.

HEAD Method

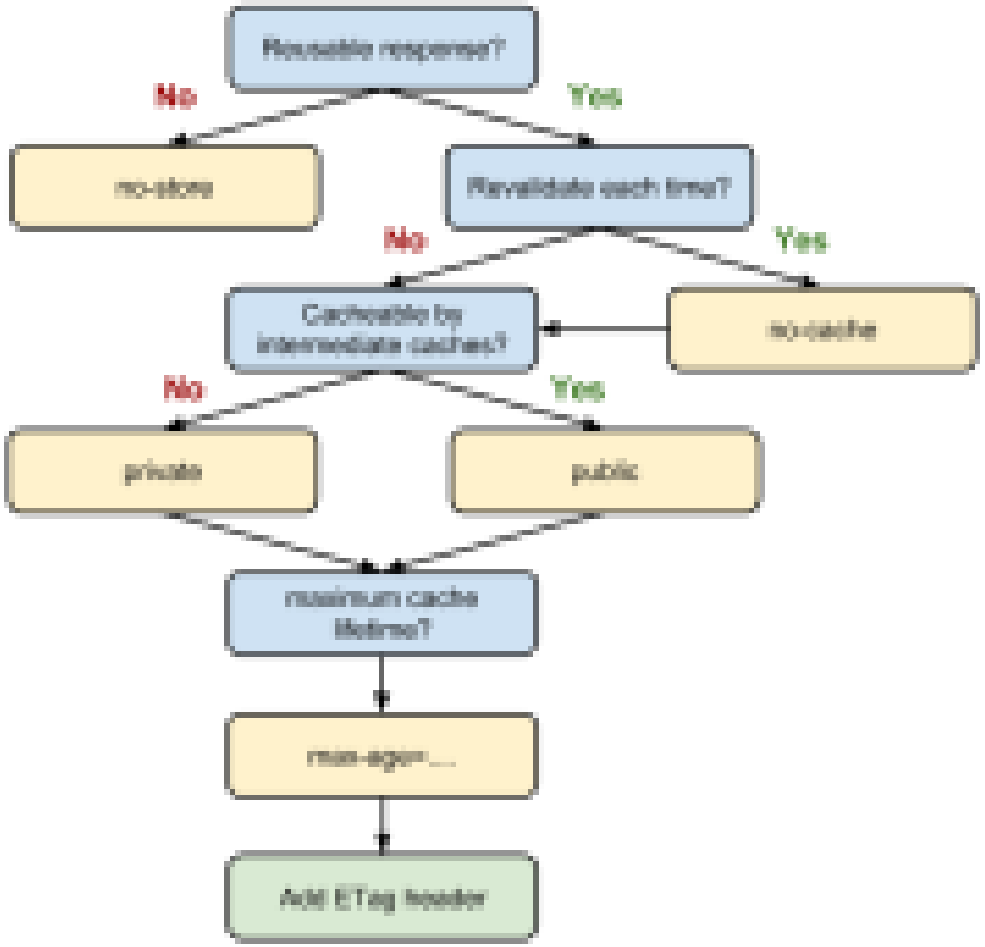
- `HEAD` method can be used purely for validation with either `Last-Modified` or `ETag` headers.
- Response status for `HEAD` method will be 304 `NOT MODIFIED` if resource has not changed.
- If validation fails, response will include all headers which would have been returned for `GET`, but no body. An additional `GET` method is required to get the new version of the resource.

Changing URLs with Infinite Cache Time Pattern

- A containing asset like a HTML page will typically contain many static assets like images, stylesheets and scripts.
- Typically. the static assets do not change often.
- Serve HTML page with `no-cache` or a very short cache time (say 10 seconds). Serve contained assets with an effectively infinite cache expiry time like 1 year.
- If a static asset changes, serve new version under a different URL and update link within HTML page containing asset (can often be done automatically using a *content-management system*).
- Once the updated HTML page is fetched from the origin server, it will automatically bring in the new version of the static asset even though the old version may still be cached under the old URL.
- Advantage of this pattern is that static assets are often served from the browser cache, but still effectively updated for clients in a timely manner.

Caching Decision Diagram

From *Google Developers HTTP Caching Tutorial*



Cookies

- Cookies are merely some limited size content.
- Cookies are sent from server to client using `Set-Cookie` header.
- Cookies are sent from client to server using the `Cookie` header.
- Cookies can be used for session management, user info, shopping carts, personalization, tracking.
- All cookies currently accessible can be read using `document.cookie`; returns semicolon-separated *name=value* pairs. Can parse easily using regex but usually better to use a library.
- Can create/update a single cookie by assigning to `document.cookie` a cookie *name=value* followed by optional cookie attributes following semicolon.

Set-Cookie Header

`Set-Cookie` response header contains following semi-colon separated fields:

name=value This is required. The *name* is the name of the cookie and *value* is its value. Example: `cart_id=a23b-c3f4-a83e`.

`Expires=`*DateTime* or `Max-Age=`*relTime* The time at which the cookie expires. If neither specified, or time is in the past, then cookie is a **session cookie** and expires when browser is closed. `Expires` is supported across all browsers, `Max-Age` on modern browsers.

`Secure` If set, then the cookie is sent only on https connections. Some modern browsers require that only https responses set this cookie.

`HttpOnly` If set, then the cookies cannot be accessed via JavaScript API's within the browser. Prevents *cross-site scripting* (XSS) attacks.

Set-Cookie Header Continued

`Domain=`*domain* The domain to which the cookie should be sent. For example, if *domain* is specified as `binghamton.edu`, then cookie will be sent to both `binghamton.edu` and `cs.binghamton.edu`.

`Path=`*path* The path and subpaths for which the cookie should be sent. For example, if *path* is `/cgi-bin`, then cookie will be sent for `/cgi-bin` as well as `/cgi-bin/carts`.

Cookie Header

`Cookie:` *name*=*value*; *name*=*value* ..

- Sent by client to server.
- Contains previously received *name* / *value* pairs for cookies which apply to request URL (domain and path).

References

- *Ilya Grigorik*, *HTTP Caching*, *Google Developers*.
- *HTTP caching*, Mozilla Developer Network.
- RFC-7234 *Hypertext Transfer Protocol (HTTP/1.1): Caching>*
- RFC-7232 *Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests*
- Brian Totty, Marjorie Sayer, Sailu Reddy, Anshu Aggarwal, David Gourley, *HTTP: The Definitive Guide*, Safari Books Online, *Persistent Connections*
- *HTTP headers*, Mozilla Developer Network.