HTTP-Header und Cache-Richtlinien

Matthias Hotz

Wie funktioniert eigentlich HTTP-Caching? Welche Informationen im HTTP-Header spielen für das Caching eine Rolle, wie arbeiten die verschiedenen Anweisungen zusammen, wo wird gecacht und wo liegen die Grenzen? Warum ist das für SEO relevant? Experte Matthias Hotz klärt auf.

Der HTTP-Header ist ein wichtiger Bestandteil des HTTP-Protokolls. Zum einen überträgt ein Client (zum Beispiel ein Browser) darin Informationen an den Server bzw. Host. Auf der anderen Seite sendet der Server darin Informationen an den Client. Das Ganze kann man sich wie eine diplomatische Verhandlung vorstellen. Ein Client teilt dem Server beispielsweise mit, welche Dateien er akzeptiert, welche Sprache er spricht, welchen Zeichensatz er verwendet etc.

Der Server antwortet daraufhin ebenso und nennt zum Beispiel die Größe der angefragten Datei, die Sprache, in der der Inhalt gesendet wird, oder auch den bekannten HTTP-Statuscode einer Datei (zum Beispiel 200, 404 oder 301).

Man kann darin Canonicals oder auch Informationen zu robots (x-robots) übertragen, ebenso wie Anweisungen zum Caching der Dateien, die abgerufen werden. Dabei hat jede Datei, die im Ladezyklus einer Website verwendet wird, ihren eigenen HTTP-Header. Also nicht nur die HTML-Datei, sondern auch CSS, Fonts, Bilder, JavaScript etc.

Caching kann an diversen Stellen der Übertragung stattfinden

Wenn man Dateien im Netz abruft, gibt es grundsätzlich drei Stellen, an denen ein Cache erzeugt werden kann: Server, Client und auf dem Weg dazwischen. Die Cache-Richtlinien im HTTP-Header gelten für alle involvierten Server und Dienste: Eine Anfrage eines Clients an einen Server wird oft nicht „nur“ vom Server beantwortet. Dazwischen liegen zum Beispiel ein oder mehrere Proxyserver der Telefonanbieter/ISPs bzw. ein CDN wie Cloudflare oder Amazon Cloud Front. Alle diese Dienste haben die Möglichkeit, Daten zu cachen, und tun dies in der Regel auch, um den Traffic der Netzwerke zu verringern und die Auslieferung an den Client/User zu beschleunigen.

Wo kann man den HTTP-Header sehen?

Im Browser (in diesem Beispiel Chrome) geht man auf „Quelltext untersuchen“ und dort auf den Tab „Netzwerk“. Dieser ist dann in der Regel erst einmal leer. Nun aktualisiert man die Seite und in der Tabelle sollten jetzt alle Dateien auftauchen, die die Website lädt. Dann wählt man eine Datei aus und in dem kleinen Fenster, das sich rechts neben der Tabelle befindet, sieht man nun sowohl den Anfrage-Header als auch den Antwort-Header.

Ein Tool, mit dem man das online machen kann, ist zum Beispiel websniffer.com.

Die Caching-Anweisung im HTTP-Header sagt, wie, wo, wann und wie lange

Ob gecacht werden darf, bestimmt der HTTP-Header. Die gängigste Anweisung lautet hier „Cache-Control“ und diese kann sowohl vom Client als auch vom Server gesendet werden. „Cache-Control: no-store“ legt zum Beispiel fest, dass absolut niemand in der Anfrage-Antwort-Kette diese Datei speichern darf. Sendet

DER AUTOR

Matthias Hotz ist Head Of Technical SEO bei der diva-e in München. Er betreut diverse Kunden im Enterprise-Umfeld und entwickelt das interne SEO-Tool der diva-e: OnePro-Seo.

Foto: Mykola Lishchyshyn / gettyimages.de

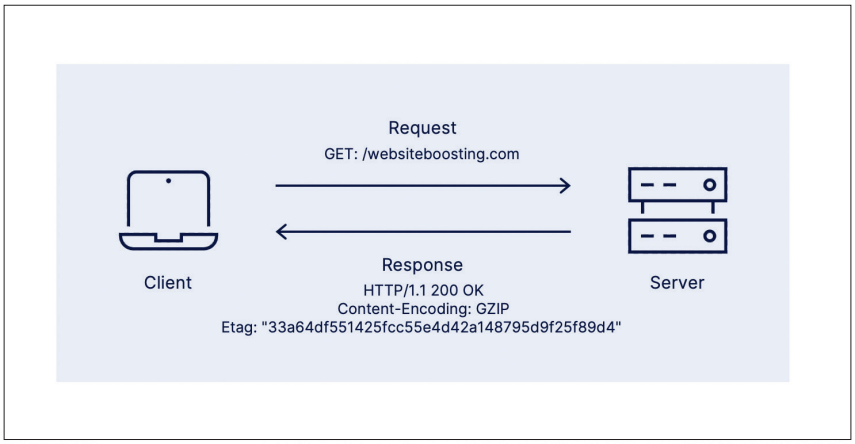

Abb. 1: Request und Response

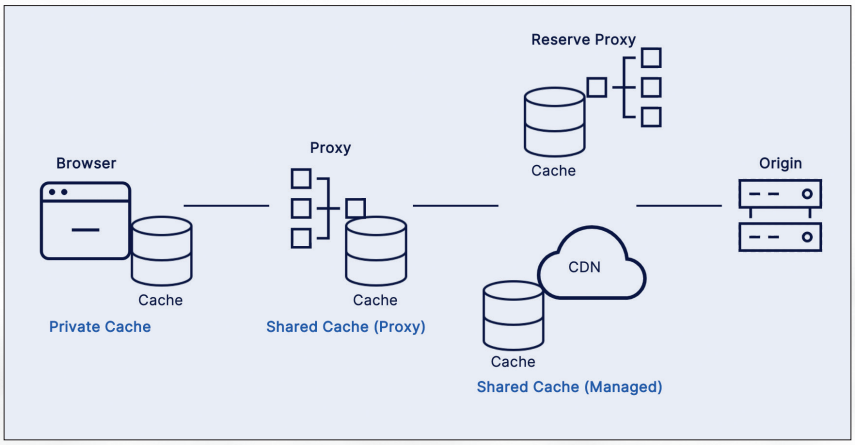

Abb. 2: Caching an verschiedenen Stellen

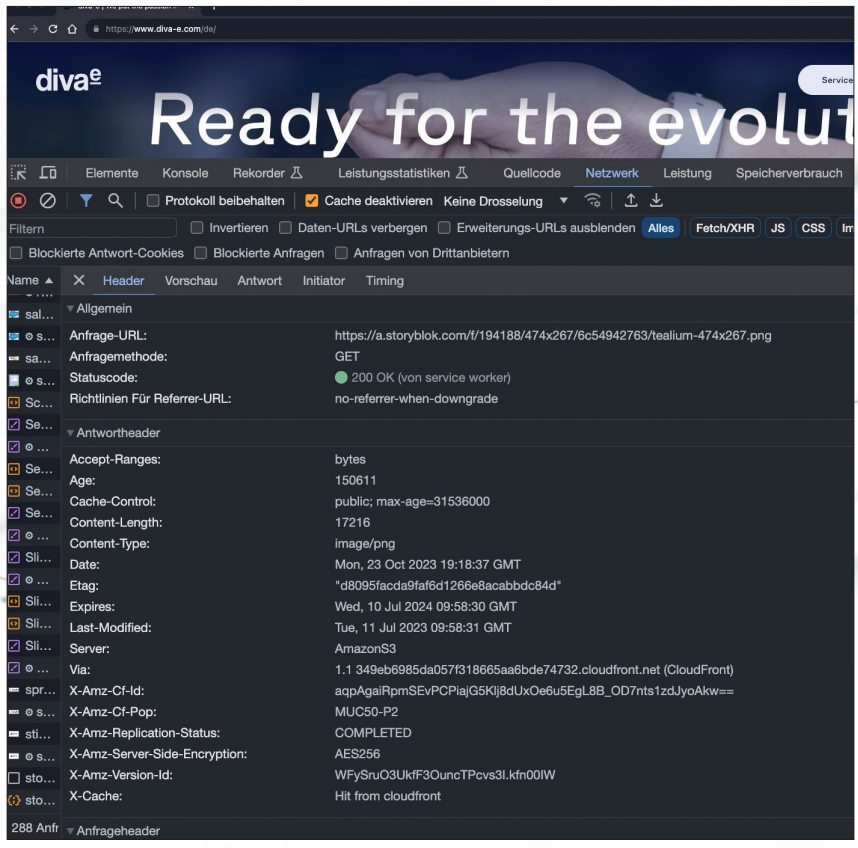

Abb. 3: Antwort-Header inspizieren

ein Client „no-store“, bedeutet das für die involvierten Server, dass sie keinen Cache anlegen dürfen. Senden Server „no-store“, darf der Client keinen Cache anlegen.

Einige Beispiele:

» **„Cache-Control: max-age=120“** bedeutet, dass der Client und die Server dazwischen die Datei für maximal 120 Sekunden speichern dürfen, bevor sie neu angefragt werden muss.

» **„Cache-Control: public“** bedeutet, dass es in Ordnung ist, diese Datei auf dem Weg zum Client und auch auf weiteren Servern zu speichern (Proxys oder CDNs).

» **„Cache-Control: private“** bedeutet, dass nur der Browser die Antwort speichern darf. Alle Server, die sich dazwischen befinden, dürfen dies nicht. Dies ist beispielsweise dann relevant, wenn Daten User-spezifisch sind oder etwa Bankdaten enthalten.

» **„Cache-Control: no-cache“** bedeutet, dass Server zwischen dem eigentlichen Host und dem Client die Datei speichern dürfen. Allerdings muss diese bei einer erneuten Abfrage revalidiert werden. Die cachenden Server müssen also prüfen, ob die Datei, die sie gespeichert haben, noch aktuell ist.

» **„Cache-Control: private, no-cache, no-store, must-revalidate“** verbindet verschiedene Anweisungen, die redundant bzw. unterschiedlich strikt sind. Hier zählt am Ende die Anweisung, welche strikter ist. In diesem Fall gilt „no-store“.

Wenn es multiple Cache-Richtlinien für eine Anweisung gibt, zum Beispiel zwei max-age- oder Expires-Werte, dann ist die komplette Direktive invalide.

Neben den oben genannten Anweisungen für „Cache-Control“ existieren diverse weitere Anweisungen.

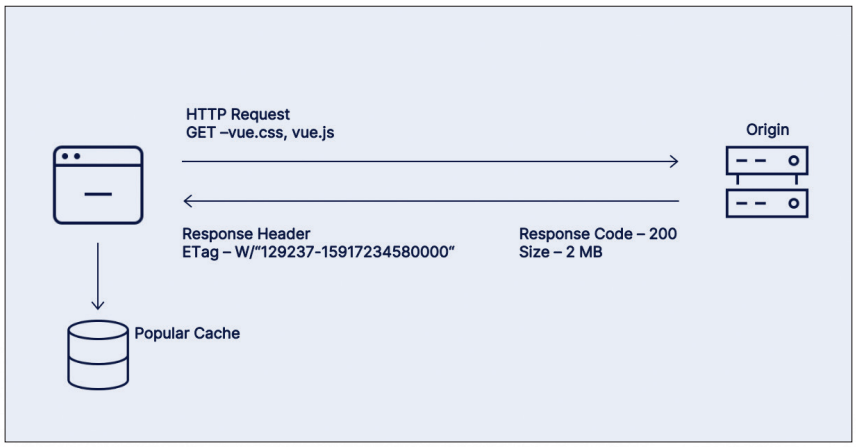

Abb. 4: Schaubild zur Funktionsweise des ETags

Liste der Anweisungen für den „Cache-Control"-Header

» **immutable (Response)**
kann „für immer" gecacht werden, zum Beispiel ein Font oder in Verbindung mit einem Far Future Expires (siehe unten).

» **max-age (Request und Response)**
ist die maximale Anzahl an Sekunden, die eine Datei aus dem Cache genommen werden darf, bevor sie neu abgerufen werden muss.

» **max-stale (Request)**
wird von den meisten Browsern nicht unterstützt, wird angewendet, wenn max-age abgelaufen ist, bedeutet, der Client akzeptiert bis zu einer gewissen Anzahl von Sekunden veraltete Antworten.

» **min-fresh (Request)**
wird von den meisten Browsern nicht unterstützt, bedeutet, dass der Client eine gespeicherte HTTP-Antwort zulässt, die für eine bestimmte Anzahl von Sekunden aktuell bleibt.

» **must-revalidate (Response)**
Eine zuvor abgerufene Datei muss nach einer gewissen Anzahl von Sekunden frisch validiert werden, bevor sie wiederum verwendet wird (siehe ETag).

» **must-understand (Response)**
Die Datei darf nur gespeichert werden, wenn der Client sie auch verarbeiten kann, spielt in der Regel nur bei API-Zugriffen eine Rolle (SOAP).

» **no-cache (Request und Response)**
Datei darf gespeichert werden, muss aber bei einer erneuten Verwendung revalidiert werden (siehe ETag).

» **no-store (Request und Response)**
Anfrage/Antwort darf nicht gespeichert werden.

» **no-transform (Request und Response)**
Anfrage/Antwort darf nicht verändert werden, also ein CDN wie zum Beispiel Cloudflare darf ein Bild nicht zusätzlich komprimieren, bevor es an den Client verschickt wird.

» **only-if-cached (Request)**
Client erwartet nur eine Antwort, wenn es auf dem Server eine gecachte Version gibt, sollte es keine geben, antworten Server mit einem HTTP 504 (Timeout).

» **private (Response)**
Antwort darf nur im Cache des Browsers gespeichert werden.

» **proxy-revalidate (Response)**
Eine zuvor abgerufene Datei muss nach einer gewissen Anzahl von Sekunden frisch validiert werden, bevor sie wiederum verwendet wird, Anweisung explizit für Proxyserver und CDNs.

» **public (Response)**
Antwort darf auch in einem öffentlichen Cache gespeichert werden, zum Beispiel Proxyserver oder CDNs.

» **s-maxage (Response)**
die maximale Anzahl an Sekunden, die eine Datei aus dem Cache genommen werden darf, bevor sie neu abgerufen werden muss, Anweisung für Proxyserver und CDNs

» **stale-if-error (Request und Response)**
Wenn der Server eine HTTP-500-, -502-, -503-, -504-Antwort sendet, kann die zuvor heruntergeladene Cache-Version benutzt werden.

» **stale-while-revalidate (Response)**
Eine gecachte Version kann so lange verwendet werden, bis sie frisch validiert wurde. Beispielsweise kann ein vorhandenes Bild angezeigt werden, während im Hintergrund geprüft wird, ob es eine neue Version gibt.

Weitere HTTP-Header, die beim Caching eine Rolle spielen

ETag

Um Caching besonders effektiv zu machen, gibt es den ETag. Dieser enthält eine eindeutige Zeichenkette für die angefragte Datei. Ändert sich nun die Datei auf dem Server, ändert sich auch die Zeichenkette, die der Server mitsendet. Ein Client, der die darunterliegende Datei gespeichert hat, muss also nur die Zeichenkette an den Server senden. Sind beide Zeichenketten weiterhin identisch, wird er die Datei nicht erneut herunterladen, weil der Server daraufhin einen HTTP-304-Not-Modified-Antwortcode sendet. Der ETag ohne den „Cache-Control"-Header ist allerdings nutzlos. Nur in Kombination mit beispielsweise einer „must-revalidate"-Anweisung hat sein Einsatz Sinn.

Der ETag kann auch für das Tracking von Usern missbraucht werden. Bekommt jeder Client/User einen einzigartigen ETag für eine Datei, wäre es möglich, den User zu verfolgen. Darum sollte man sicherstellen, dass der String des ETags nur für die angefragte Datei eindeutig ist.

Expires

Der Expires-Header kann vom Server gesendet werden. Darin steht ein Datum mit Uhrzeit, das in der Regel in der Zukunft liegt. Dem Client wird damit mitgeteilt, ab wann die Datei veraltet ist und diese beim Server neu angefragt werden muss. Der Expires-Header ist mittlerweile eher eine Fallback-Lösung für ältere Browser. Wenn ein „Cache-Control max-age“ ebenfalls gesendet wird, ignorieren moderne Clients die Expires-Anweisung.

Last Modified

Hiermit kann dem Client mitgeteilt werden, wann die Datei das letzte Mal aktualisiert wurde. Da vor der eigentlichen Datei immer der HTTP-Header übertragen wird, kann der Client selbst entscheiden, ob er die Datei neu herunterlädt.

Far Future Expires

Eine gängige Technik für statische Dateien sind Far-Future-Expires-Header. Man setzt hierbei die „max-age“- bzw. Expires-Anweisung sehr weit in die Zukunft (zum Beispiel ein Jahr) oder nutzt die „immutable“-Anweisung. Allerdings erhalten die Dateien einen Fingerprint in der URL. Eine CSS-Datei hat in der ersten Anfrage an den Server den Dateinamen */style.css?v=1*. Diese wird vom Browser dann gecacht. Ändert man nun etwas an der CSS-Datei, bekommt sie den Dateinamen */style.css?v=2* und somit ist sie für den Client eine unbekannte Datei und wird neu heruntergeladen.

Die Grenzen des HTTP-Cachings

HTTP-Caching kann die Ladezeiten einer Website massiv beschleunigen. Allerdings erst ab dem Second View bzw. mit gefülltem Browser-Cache. Da allerdings HTML-Dokumente normalerweise nicht gecacht werden oder sehr kurze Verfallszeiten haben, sollte man darauf achten, dass man einen schlanken HTML-Quelltext ausliefert.

Es empfiehlt sich auch, HTML-Dokumente, die keine User-spezifischen Daten enthalten, für einen kurzen Zeitraum (zum Beispiel 120 Sekunden) zu cachen. Besucht ein User dann dieselbe Seite in diesen zwei Minuten wieder, muss sie nicht neu geladen werden.

Auch hilft HTTP-Caching nur sehr begrenzt bei einer schlechten Time To First Byte (TTFB) oder Interaction To Next Paint (INP) und Cumulative Layout Shift (CLS), da diese Werte vor bzw. nach der Verwendung des Cache liegen.

Alle gängigen Webserver (Nginx, Apache etc.) unterstützen HTTP-Caching, haben aber teils unterschiedliche Basiseinstellungen. Mit diversen CDNs (Cloudflare) können Cache-Einstellungen ebenfalls getroffen und verändert werden.

Was hat das mit SEO zu tun?

Eine Relevanz für SEO ergibt sich durch Page Performance und Core Web Vitals – Metriken, die Google verwendet, um Seiten für das Ranking zu bewerten.

Google Chrome sammelt Daten zur Ladegeschwindigkeit und den Core Web Vitals. Diese fließen in den Chrome User Experience Report (CrUX Report). Die gesammelten Daten dienen unter anderem als Grundlage für das Reporting der CWV in der Search Console und innerhalb von Page Speed Insight (Testing Tool).

Da die Daten direkt im Chrome entstehen und die meisten Anwender innerhalb der User Journey diverse URLs einer Domain besuchen, kann ein effizientes HTTP-Caching die im CrUX erfassten Core Web Vitals massiv verbessern, auch wenn die Website im First View (mit leerem Cache) eher schlechte Werte erzielt.