Stream: Internet Engineering Task Force (IETF)
RFC: 9211
Category: Standards Track
Published: June 2022
ISSN: 2070-1721
Author: M. Nottingham
Fastly

RFC 9211
The Cache-Status HTTP Response Header Field

Abstract

To aid debugging, HTTP caches often append header fields to a response, explaining how they handled the request in an ad hoc manner. This specification defines a standard mechanism to do so that is aligned with HTTP's caching model.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9211.

Table of Contents

1. Introduction 2
1.1. Notational Conventions 3
2. The Cache-Status HTTP Response Header Field 3
2.1. The hit Parameter 4
2.2. The fwd Parameter 4
2.3. The fwd-status Parameter 5
2.4. The ttl Parameter 5
2.5. The stored Parameter 5
2.6. The collapsed Parameter 5
2.7. The key Parameter 5
2.8. The detail Parameter 5
3. Examples 6
4. Defining New Cache-Status Parameters 7
5. IANA Considerations 8
6. Security Considerations 8
7. References 8
7.1. Normative References 8
7.2. Informative References 9
Author's Address 9

1. Introduction

To aid debugging (both by humans and automated tools), HTTP caches often append header fields to a response explaining how they handled the request. Unfortunately, the semantics of these header fields are often unclear, and both the semantics and syntax used vary between implementations.

This specification defines a new HTTP response header field, "Cache-Status", for this purpose with standardized syntax and semantics.

1.1. Notational Conventions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the following terminology from Section 3 of [STRUCTURED-FIELDS] to specify syntax and parsing: List, String, Token, Integer, and Boolean.

This document also uses terminology from [HTTP] and [HTTP-CACHING].

2. The Cache-Status HTTP Response Header Field

The Cache-Status HTTP response header field indicates how caches have handled that response and its corresponding request. The syntax of this header field conforms to [STRUCTURED-FIELDS].

Its value is a List. Each member of the List represents a cache that has handled the request. The first member represents the cache closest to the origin server, and the last member represents the cache closest to the user (possibly including the user agent's cache itself if it appends a value).

Caches determine when it is appropriate to add the Cache-Status header field to a response. Some might add it to all responses, whereas others might only do so when specifically configured to, or when the request contains a header field that activates a debugging mode. See Section 6 for related security considerations.

An intermediary **SHOULD NOT** append a Cache-Status member to responses that it generates locally, even if that intermediary contains a cache, unless the generated response is based upon a stored response (e.g., 304 (Not Modified) and 206 (Partial Content) are both based upon a stored response). For example, a proxy generating a 400 response due to a malformed request will not add a Cache-Status value, because that response was generated by the proxy, not the origin server.

When adding a value to the Cache-Status header field, caches **SHOULD** preserve the existing field value, to allow debugging of the entire chain of caches handling the request.

Each List member identifies the cache that inserted it, and this identifier **MUST** be a String or Token. Depending on the deployment, this might be a product or service name (e.g., "ExampleCache" or "Example CDN"), a hostname ("cache-3.example.com"), an IP address, or a generated string.

Each member of the list can have parameters that describe that cache's handling of the request. While these parameters are **OPTIONAL**, caches are encouraged to provide as much information as possible.

This specification defines the following parameters.

2.1. The hit Parameter

The value of "hit" is a Boolean that, when true, indicates that the request was satisfied by the cache; that is, it was not forwarded, and the response was obtained from the cache.

A response that was originally produced by the origin but was modified by the cache (for example, a 304 or 206 status code) is still considered a hit, as long as it did not go forward (e.g., for validation).

A response that was in cache but not able to be used without going forward (e.g., because it was stale or partial) is not considered a hit. Note that a stale response that is used without going forward (e.g., because the origin server is not available) can be considered a hit.

"hit" and "fwd" are exclusive; only one of them should appear on each list member.

2.2. The fwd Parameter

"fwd", when present, indicates that the request went forward towards the origin; its value is a Token that indicates why.

The following parameter values are defined to explain why the request went forward, from most specific to least:

bypass: The cache was configured to not handle this request.

method: The request method's semantics require the request to be forwarded.

uri-miss: The cache did not contain any responses that matched the request URI.

vary-miss: The cache contained a response that matched the request URI, but it could not select a response based upon this request's header fields and stored Vary header fields.

miss: The cache did not contain any responses that could be used to satisfy this request (to be used when an implementation cannot distinguish between uri-miss and vary-miss).

request: The cache was able to select a fresh response for the request, but the request's semantics (e.g., Cache-Control request directives) did not allow its use.

stale: The cache was able to select a response for the request, but it was stale.

partial: The cache was able to select a partial response for the request, but it did not contain all of the requested ranges (or the request was for the complete response).

The most specific reason known to the cache **SHOULD** be used, to the extent that it is possible to implement. See also [HTTP-CACHING], Section 4.

2.3. The fwd-status Parameter

The value of "fwd-status" is an Integer that indicates which status code (see [HTTP], Section 15) the next-hop server returned in response to the forwarded request. The fwd-status parameter is only meaningful when fwd is present. If fwd-status is not present but the fwd parameter is, it defaults to the status code sent in the response.

This parameter is useful to distinguish cases when the next-hop server sends a 304 (Not Modified) response to a conditional request or a 206 (Partial Content) response because of a range request.

2.4. The ttl Parameter

The value of "ttl" is an Integer that indicates the response's remaining freshness lifetime (see [HTTP-CACHING], Section 4.2.1) as calculated by the cache, as an integer number of seconds, measured as closely as possible to when the response header section is sent by the cache. This includes freshness assigned by the cache through, for example, heuristics (see [HTTP-CACHING], Section 4.2.2), local configuration, or other factors. It may be negative, to indicate staleness.

2.5. The stored Parameter

The value of "stored" is a Boolean that indicates whether the cache stored the response (see [HTTP-CACHING], Section 3); a true value indicates that it did. The stored parameter is only meaningful when fwd is present.

2.6. The collapsed Parameter

The value of "collapsed" is a Boolean that indicates whether this request was collapsed together with one or more other forward requests (see [HTTP-CACHING], Section 4). If true, the response was successfully reused; if not, a new request had to be made. If not present, the request was not collapsed with others. The collapsed parameter is only meaningful when fwd is present.

2.7. The key Parameter

The value of "key" is a String that conveys a representation of the cache key (see [HTTP-CACHING], Section 2) used for the response. Note that this may be implementation specific.

2.8. The detail Parameter

The value of "detail" is either a String or a Token that allows implementations to convey additional information not captured in other parameters, such as implementation-specific states or other caching-related metrics.

For example:

```
Cache-Status: ExampleCache; hit; detail=MEMORY
```

The semantics of a detail parameter are always specific to the cache that sent it; even if a details parameter from another cache shares the same value, it might not mean the same thing.

This parameter is intentionally limited. If an implementation's developer or operator needs to convey additional information in an interoperable fashion, they are encouraged to register extension parameters (see Section 4) or define another header field.

3. Examples

The following is an example of a minimal cache hit:

```
Cache-Status: ExampleCache; hit
```

However, a polite cache will give some more information, e.g.:

```
Cache-Status: ExampleCache; hit; ttl=376
```

A stale hit just has negative freshness, as in this example:

```
Cache-Status: ExampleCache; hit; ttl=-412
```

Whereas this is an example of a complete miss:

```
Cache-Status: ExampleCache; fwd=uri-miss
```

This is an example of a miss that successfully validated on the backend server:

```
Cache-Status: ExampleCache; fwd=stale; fwd-status=304
```

This is an example of a miss that was collapsed with another request:

```
Cache-Status: ExampleCache; fwd=uri-miss; collapsed
```

This is an example of a miss that the cache attempted to collapse, but couldn't:

```
Cache-Status: ExampleCache; fwd=uri-miss; collapsed=?0
```

The following is an example of going through two separate layers of caching, where the cache closest to the origin responded to an earlier request with a stored response, and a second cache stored that response and later reused it to satisfy the current request:

```
Cache-Status: OriginCache; hit; ttl=1100,
              "CDN Company Here"; hit; ttl=545
```

The following is an example of going through a three-layer caching system, where the closest to the origin is a reverse proxy (where the response was served from cache); the next is a forward proxy interposed by the network (where the request was forwarded because there wasn't any response cached with its URI, the request was collapsed with others, and the resulting response was stored); and the closest to the user is a browser cache (where there wasn't any response cached with the request's URI):

```
Cache-Status: ReverseProxyCache; hit
Cache-Status: ForwardProxyCache; fwd=uri-miss; collapsed; stored
Cache-Status: BrowserCache; fwd=uri-miss
```

4. Defining New Cache-Status Parameters

New Cache-Status parameters can be defined by registering them in the "HTTP Cache-Status" registry.

Registration requests are reviewed and approved by a designated expert, per [RFC8126], Section 4.5. A specification document is appreciated but not required.

The expert(s) should consider the following factors when evaluating requests:

- Community feedback
- If the value is sufficiently well defined
- Generic parameters are preferred over vendor-specific, application-specific, or deployment-specific values. If a generic value cannot be agreed upon in the community, the parameter's name should be correspondingly specific (e.g., with a prefix that identifies the vendor, application, or deployment).

Registration requests should use the following template:

Name: [a name for the Cache-Status parameter's key; see Section 3.1.2 of [STRUCTURED-FIELDS] for syntactic requirements]

Type: [the Structured Type of the parameter's value; see Section 3.1.2 of [STRUCTURED-FIELDS]]

Description: [a description of the parameter's semantics]

Reference: [to a specification defining this parameter, if available]

See the registry at <https://www.iana.org/assignments/http-cache-status> for details on where to send registration requests.

5. IANA Considerations

IANA has created the "HTTP Cache-Status" registry at <https://www.iana.org/assignments/http-cache-status> and populated it with the types defined in Section 2; see Section 4 for its associated procedures.

IANA has added the following entry in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" defined in [HTTP], Section 18.4:

Field name: Cache-Status
Status: permanent
Reference: RFC 9211

6. Security Considerations

Attackers can use the information in Cache-Status to probe the behavior of the cache (and other components) and infer the activity of those using the cache. The Cache-Status header field may not create these risks on its own, but it can assist attackers in exploiting them.

For example, knowing if a cache has stored a response can help an attacker execute a timing attack on sensitive data.

Additionally, exposing the cache key can help an attacker understand modifications to the cache key, which may assist cache poisoning attacks. See [ENTANGLE] for details.

The underlying risks can be mitigated with a variety of techniques (e.g., using encryption and authentication and avoiding the inclusion of attacker-controlled data in the cache key), depending on their exact nature. Note that merely obfuscating the key does not mitigate this risk.

To avoid assisting such attacks, the Cache-Status header field can be omitted, only sent when the client is authorized to receive it, or sent with sensitive information (e.g., the key parameter) only when the client is authorized.

7. References

7.1. Normative References

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[HTTP-CACHING] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <https://www.rfc-editor.org/info/rfc9111>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[STRUCTURED-FIELDS] Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <https://www.rfc-editor.org/info/rfc8941>.

7.2. Informative References

[ENTANGLE] Kettle, J., "Web Cache Entanglement: Novel Pathways to Poisoning", September 2020, <https://portswigger.net/research/web-cache-entanglement>.

Author's Address

Mark Nottingham
Fastly
Prahran
Australia
Email: mnot@mnot.net
URI: https://www.mnot.net/