Network Working Group
Internet-Draft
Updates: 7234 (if approved)
Intended status: Standards Track
Expires: April 1, 2018

M. Nottingham
Fastly

September 28, 2017

HTTP Variants
draft-nottingham-variants-00

Abstract

This specification introduces the HTTP "Variants" response header field to communicate what representations are available for a given resource.

Note to Readers

RFC EDITOR: please remove this section before publication

The issues list for this draft can be found at https://github.com/mnot/I-D/labels/variant.

The most recent (often, unpublished) draft is at https://mnot.github.io/I-D/variant/.

Recent changes are listed at https://github.com/mnot/I-D/commits/gh-pages/variant.

See also the draft's current status in the IETF datatracker, at https://datatracker.ietf.org/doc/draft-nottingham-variant/.

Table of Contents

1. Introduction

HTTP proactive content negotiation ([RFC7231], Section 3.4.1) is starting to be used more widely again. The most widely seen use - determining a response's content-coding - is being joined by renewed interest in negotiation for language and other, newer attributes (for example, see [I-D.ietf-httpbis-client-hints]).

Successfully reusing negotiated responses that have been stored in a HTTP cache requires establishment of a secondary cache key

([RFC7234], Section 4.1) using the Vary header ([RFC7231], Section 7.1.4), which identifies the request headers that form the secondary cache key for a given response.

HTTP's caching model allows a certain amount of latitude in normalising request header fields to match those stored in the cache, so as to increase the chances of a cache hit while still respecting the semantics of that header. However, this is often inadequate; even with understanding of the headers' semantics to facilitate such normalisation, a cache does not know enough about the possible alternative representations available on the origin server to make an appropriate decision.

For example, if a cache has stored the following request/response pair:

```
GET /foo HTTP/1.1
Host: www.example.com
Accept-Language: en;q=1.0, fr;q=0.5

HTTP/1.1 200 OK
Content-Type: text/html
Content-Language: fr
Vary: Accept-Language
Transfer-Encoding: chunked

[French content]
```

Provided that the cache has full knowledge of the semantics of "Accept-Language" and "Content-Language", it will know that a French representation is available and might be able to infer that an English representation is not available. But, it does not know (for example) whether a Japanese representation is available without making another request, thereby incurring possibly unnecessary latency.

This specification introduces the HTTP "Variants" response header field to enumerate the available variant representations on the origin server, to provide clients and caches with enough information to properly satisfy requests - either by selecting a response from cache or by forwarding the request towards the origin.

"Variants" is best used when content negotiation takes place over a constrained set of representations; since each variant needs to be listed in the header field, it is ill-suited for open-ended sets of representations. Likewise, it works best for content negotiation over header fields whose semantics are well-understood, since it requires a selection algorithm to be specified ahead of time.

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 specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] with a list extension, defined in Section 7 of [RFC7230], that allows for compact definition of comma-separated lists using a '#' operator (similar to how the '*' operator indicates repetition).

Additionally, it uses the "field-name", "OWS" and "token" rules from [RFC7230].

2. The "Variants" HTTP Header Field

The "Variant" HTTP response header field is used to indicate what other representations are available for a given resource at the time that the response is produced.

```
Variants        = 1#variant
variant         = field-name *( OWS ";" OWS available-value )
available-value = token
```

Each "variant" indicates a response header field that carries a value that clients might proactively negotiate for; each parameter on it indicates a value for which there is an available representation on the origin server.

So, given this example header field:

Variants: Content-Encoding;gzip

a recipient can infer that the only content-coding available for that resource is "gzip" (along with the "identity" non-encoding; see {{content-encoding}}).

Given:

Variants: content-encoding

a recipient can infer that no content-codings are supported. Note that as always with header field names, it is case-insensitive.

A more complex example:

Variants: Content-Encoding;gzip;brotli, Content-Language;en ;fr

Here, recipients can infer that two Content-Encodings are available, as well as two content languages. Note that, as with all HTTP header fields that use the "#" list rule (see [RFC7230], Section 7), they might occur in the same header field or separately, like this:

Variants: Content-Encoding;gzip;brotli
Variants: Content-Language;en ;fr

The ordering of available-values after the field-name is significant, as it might be used by the header's algorithm for selecting a response.

Senders SHOULD consistently send "Variant" header fields on all cacheable (as per [RFC7234], Section 3) responses for a resource, since its absence will trigger caches to fall back to "Vary" processing.

Likewise, servers MUST send the "Content-*" response headers nominated by "Variants" when sending that header.

2.1. Defining Content Negotiation Using Variants

To be usable with Variants, proactive content negotiation mechanisms need to be specified to take advantage of it. Specifically, they:

- MUST define a request header field that advertises the clients preferences or capabilities, whose field-name SHOULD begin with "Accept-".

- MUST define a response header field that indicates the result of selection, whose field-name SHOULD begin with "Content-" and whose field-value SHOULD be a token.

- MUST define an algorithm for selecting a result. It MUST return an ordered list of selected responses, given the incoming request, a list of selected responses, and the list of available values from "Variants". If the result is an empty list, it implies that the cache does not contain an appropriate response.

Appendix A fulfils these requirements for some existing proactive content negotiation mechanisms in HTTP.

Note that unlike Vary, Variants does not use stored request headers to help select a response; this is why defining a response header to aid identification and selection is required.

2.2. Cache Behaviour

Caches that implement the "Variants" header field and the relevant semantics of the field-name it contains can use that knowledge to either select an appropriate stored representation, or forward the request if no appropriate representation is stored.

They do so by running this algorithm (or its functional equivalent) upon receiving a request, "incoming-request":

1. Let "selected-responses" be a list of the stored responses suitable for reuse as defined in [RFC7234] Section 4, excepting the requirement to calculate a secondary cache key.

2. Order "selected-responses" by the "Date" header field, most recent to least recent.

3. If the freshest (as per [RFC7234], Section 4.2) has one or more "Variants" header field(s):

    1. Select one member of "selected-responses" and let its "Variants" header field-value(s) be "Variants". This SHOULD be the most recent response, but MAY be from an older one as long as it is still fresh.

    2. For each "variant" in "Variants":

        1. If the "field-name" corresponds to the response header field identified by a content negotiation mechanism that the implementation supports:

            1. Let "available-values" be a list containing all "available-value" for the "variant".

            2. Let "selected-responses" be the result of running the algorithm defined by the content negotiation mechanism with "incoming-request", "selected-responses" and "available-values".

            3. For the purposes of selecting a response, ignore the content negotiation's identified request header field-name in the "Vary" header field of each member of "selected-responses", if present.

4. Process any member of "selected-responses" that has a "Vary" response header field whose field-value still contains one or more "field-name"s, removing that members if it does not match (as per [RFC7234], Section 4.1).

5. Return the first member of "selected-responses". If "selected-responses" is empty, return "null".

This algorithm will either return the appropriate stored response to use, or "null" if the cache needs to forward the request towards the origin server.

2.2.1. Relationship to Vary

Caches that fully implement this specification MUST ignore request header-fields in the "Vary" header for the purposes of secondary cache key calculation ([RFC7234], Section 4.1) when their semantics are understood, implemented as per this specification, and their corresponding response header field is listed in "Variants".

Request header fields listed in "Vary" that are not implemented in terms of this specification or not present in the "Variants" field SHOULD still form part of the secondary cache key.

The algorithm in Section 2.2 implements these requirements.

2.3. Examples

2.3.1. Single Variant

Given a request/response pair:

```
GET /foo HTTP/1.1
Host: www.example.com
Accept-Language: en;q=1.0, fr;q=0.5

HTTP/1.1 200 OK
Content-Type: image/gif
Content-Language: en
Cache-Control: max-age=3600
Variants: Content-Language;en;de
Vary: Accept-Language
Transfer-Encoding: chunked
```

Upon receipt of this response, the cache knows that two representations of this resource are available, one with a "Content-Language" of "en", and another whose "Content-Language" is "de".

Subsequent requests (while this response is fresh) will cause the cache to either reuse this response or forward the request, depending on what the selection algorithm "Accept-Language" and "Content-Language" determines.

So, a request with "en" in "Accept-Language" is received and its q-value indicates that it is acceptable, the stored response is used. A request that indicates that "de" is acceptable will be forwarded to the origin, thereby populating the cache. A cache receiving a request that indicates both languages are acceptable will use the q-value to make a determination of what response to return.

A cache receiving a request that does not list either language as acceptable (or does not contain an Accept-Language at all) will return the "en" representation (possibly fetching it from the origin), since it is listed first in the "Variants" list.

Note that "Accept-Language" is listed in Vary, to assure backwards-compatibility with caches that do not support "Variants".

Also, note that is is the response header which is listed in Variants, not the request header (the opposite of Vary).

2.3.2. Multiple Variants

A more complicated request/response pair:

```
GET /bar HTTP/1.1
Host: www.example.net
Accept-Language: en;q=1.0, fr;q=0.5
Accept-Encoding: gzip, br

HTTP/1.1 200 OK
Content-Type: image/gif
Content-Language: en
Content-Encoding: br
Variants: Content-Language;en;jp;de
Variants: Content-Encoding;br;gzip
Vary: Accept-Language, Accept-Encoding
Transfer-Encoding: chunked
```

Here, the cache knows that there are two axes that the response varies upon; "Content-Language" and "Content-Encoding". Thus, there are a total of six possible representations for the resource, and the cache needs to consider the selection algorithms for both axes.

Upon a subsequent request, if both selection algorithms return a stored representation, it can be served from cache; otherwise, the request will need to be forwarded to origin.

2.3.3. Partial Coverage

Now, consider the previous example, but where only one of the varied axes is listed in "Variants":

```
GET /bar HTTP/1.1
Host: www.example.net
Accept-Language: en;q=1.0, fr;q=0.5
Accept-Encoding: gzip, br

HTTP/1.1 200 OK
Content-Type: image/gif
Content-Language: en
Content-Encoding: br
Variants: Content-Encoding;br;gzip
Vary: Accept-Language, Accept-Encoding
Transfer-Encoding: chunked
```

Here, the cache will need to calculate a secondary cache key as per [RFC7234], Section 4.1 - but considering only "Accept-Language" to be in its field-value - and then continue processing "Variants" for the set of stored responses that the algorithm described there selects.

3. IANA Considerations

This specification registers one value in the Permanent Message Header Field Names registry established by [RFC3864]:

- Header field name: Variants
- Applicable protocol: http
- Status: standard
- Author/Change Controller: IETF
- Specification document(s): [this document]
- Related information:

4. Security Considerations

If the number or advertised characteristics of the representations available for a resource are considered sensitive, the "Variants" header by its nature will leak them.

Note that the "Variants" header is not a commitment to make representations of a certain nature available; the runtime behaviour of the server always overrides hints like "Variants".

5. Acknowledgments

This protocol is conceptually similar to, but simpler than, Transparent Content Negotiation [RFC2295]. Thanks to its authors for their inspiration.

It is also a generalisation of a Fastly VCL feature designed by Rogier 'DocWilco' Mulhuijzen.

Thanks to Hooman Beheshti for his review and input.

6. References

6.1. Normative References

[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>.

[RFC4647] Phillips, A. and M. Davis, "Matching of Language Tags", BCP 47, RFC 4647, DOI 10.17487/RFC4647, September 2006, <https://www.rfc-editor.org/info/rfc4647>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <https://www.rfc-editor.org/info/rfc7234>.

[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>.

6.2. Informative References

[I-D.ietf-httpbis-client-hints] Grigorik, I., "HTTP Client Hints", draft-ietf-httpbis-client-hints-04 (work in progress), April 2017.

[RFC2295] Holtman, K. and A. Mutz, "Transparent Content Negotiation in HTTP", RFC 2295, DOI 10.17487/RFC2295, March 1998, <https://www.rfc-editor.org/info/rfc2295>.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <https://www.rfc-editor.org/info/rfc3864>.

Appendix A. Variants and Defined Content Negotiation Mechanisms

This appendix defines the required information to use existing proactive content negotiation mechanisms (as defined in [RFC7231], Section 5.3) with the "Variants" header field.

A.1. Content-Encoding

When negotiating for the "Content-Encoding" response header field's value, the applicable request header field is "Accept-Encoding", as per [RFC7231] Section 5.3.4.

To perform content negotiation for Content-Encoding given an "incoming-request", "stored-responses" and "available-values":

1. Let "preferred-codings" be a list of the "coding"s in the "Accept-Encoding" header field of "incoming-request", ordered by their "weight", highest to lowest. If "Accept-Encoding" is not present or empty, "preferred-codings" will be empty.

2. If "identity" is not a member of "preferred-codings", append "identity" to "preferred-codings" with a "weight" of 0.001.

3. Remove any member of "preferred-codings" whose "weight" is 0.

4. Append "identity" to "available-values".

5. Remove any member of "available-values" not present in "preferred-codings", comparing in a case-insensitive fashion.

6. Let "filtered-responses" be an empty list.

7. For each "available-value" of "available-values":

    1. If there is a member of "stored-responses" whose "Content-Encoding" field-value has "content-coding"s ([RFC7231], Section 3.1.2.2) that all match members of "available-value" in a case-insensitive fashion, append that stored response to "filtered-responses".

8. If there is a member of "stored-responses" that does not have a "Content-Encoding" header field, append that stored response to "filtered-responses".

9. Return "filtered-responses".

This algorithm selects the stored response(s) in order of preference by the client; if none are stored in cache, the request will be forwarded towards the origin. It defaults to the "identity" non-encoding.

Implementations MAY remove members of "filtered-responses" based upon their "weight" or other criteria before returning. For example, they might wish to return an empty list when the client's most-preferred available response is not stored, so as to populate the cache as well as honour the client's preferences.

A.2. Content-Language

When negotiating for the "Content-Language" response header field's value, the applicable request header field is "Accept-Language", as per [RFC7231] Section 5.3.5.

To perform content negotiation for Content-Language given an "incoming-request", "stored-responses" and "available-values":

1. Let "preferred-langs" be a list of the "language-range"s in the "Accept-Language" header field ([RFC7231], Section 5.3.5) of "incoming-request", ordered by their "weight", highest to lowest.

2. If "preferred-langs" is empty, append "*" with a "weight" of 0.001.

3. Remove any member of "preferred-langs" whose "weight" is 0.

4. Filter "available-values" using "preferred-langs" with either the Basic Filtering scheme defined in [RFC4647] Section 3.3.1, or the

Lookup scheme defined in Section 3.4 of that document. Use the first member of "available-values" as the default.

5. Let "filtered-responses" be an empty list.

6. For each "available-value" of "available-values":

    1. If there is a member of "stored-responses" whose "Content-Language" field-value has a "language-tag" ([RFC7231], Section 3.1.3.2) that matches "available-value" in a case-insensitive fashion, append that stored response to "filtered-responses".

7. Return "filtered-responses".

This algorithm selects the available response(s) (according to "Variants") in order of preference by the client; if none are stored in cache, the request will be forwarded towards the origin. If no preferred language can be selected, the first "available-value" will be used as the default.

Implementations MAY remove members of "filtered-responses" based upon their "weight" or other criteria before returning. For example, they might wish to return an empty list when the client's most-preferred available response is not stored, so as to populate the cache as well as honour the client's preferences.

Author's Address

Mark Nottingham
Fastly

Email: mnot@mnot.net
URI: https://www.mnot.net/