Network Working Group	M. Nottingham
Internet-Draft	M. Kelly
Intended status: Informational	November 25, 2011
Expires: May 28, 2012	

Linked Cache Invalidation
draft-nottingham-linked-cache-inv-01

Abstract

This memo defines Web link types that invalidate HTTP caches, along with an HTTP cache-control extension that allows caches that understand those link types to use responses containing them. Together, these mechanisms offer a way to avoid use of a response that has become stale due to another request that changes server-side state. Collectively, this is referred to as Linked Cache Invalidation (LCI).

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

In normal operation, a HTTP [RFC2616] cache will invalidate a stored response if a state-changing request (e.g., POST, PUT or DELETE) is made for that URI. HTTP also provides for such a state-changing request to invalidate related resources (using the Location and Content-Location headers in the response), but this is of limited utility, because those headers have defined semantics, and can only occur once each.

Because of this, it is not practical to make a response that depends on the state of another resource cacheable. For example, an update to a blog entry might change several different resources, such as the user's summary page, the blog's "front" page, the blog's Atom feed, and of course the blog entry itself. If any of these resources is made cacheable, it will not reflect those changes, causing confusion if the user tries to verify that their changes have been correctly applied.

This memo introduces new Web link relation types [RFC5988] that allow more fine-grained relationships between resources to be defined, so that caches can invalidate all related resources when the state of one changes. It also introduces a cache-control response extension, so that responses using the relations can be cached by implementations that understand these relations.

1.1. Example

Taking the blog use case described above, imagine that we have the following related resources:

*http://example.com/blog/2011/05/04/hi {the blog entry}

*http://example.com/blog/2011/05/04/hi/comments {full comments for the entry}

*http://example.com/blog/ {the blog "home"}

*http://example.com/users/bob/ {the user page, listing his entries}

When someone comments on Bob's blog entry, they might send a request like this:

```
POST /cgi-bin/blog.cgi HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 7890

[...]
```

This request (if successful) should have the effect of invalidating the related resources listed above.
If the comment is successful, it's typical to redirect the client back to the original blog page, with a response like this:

```
HTTP/1.1 302 Moved Temporarily
Location: http://example.com/blog/2011/05/04/hi
Content-Length: 0
```

Which would invalidate the blog entry URI, as per HTTP's normal operation.
To invalidate the full comments page for the entry, the relationship can be described in that page's response headers:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 5555
Link: </blog/2011/05/04/hi>; rel="inv-by"
Cache-Control: no-cache, inv-maxage=600

[...]
```

This declares that whenever the entry page (the target of the link header) changes, this response (the full comments page) changes as well; it's invalidated by the link target.
Note that the full comments page also carries a Cache-Control header that instructs "normal" caches not to reuse this response, but allows those caches that are aware of LCI to consider it fresh for ten minutes.
To invalidate the blog home page and user page, it's impractical to list all of the resources that might change if a new entry is posted; not only are there many of them, but their URLs might not be known when the pages are cached. To address this, the POST response itself can

nominate resources to invalidate, using the 'invalidates' relation, making that response:

```
HTTP/1.1 302 Moved Temporarily
Location: http://example.com/blog/2011/05/04/hi
Link: <http://example.com/blog/>; rel="invalidates",
      <http://example.com/users/bob/>; rel="invalidates"
Content-Length: 0
```

Depending on how important it is to see updates on the home page and user page, those responses can either allow caching regardless of support for LCI, like this:

```
Cache-Control: max-age=300
```

... or they can only allow caching by LCI-aware caches, like this:

```
Cache-Control: no-cache, inv-maxage=300
```

Together, these techniques can be used to invalidate a variety of related responses.

It is important to note that the invalidations are only effective in the caches that the client's request stream travels through. Typically, this means that the client making the changes (e.g., the blog update above) will see the effects immediately, while other users whose requests travel through different caches will only see the changes once the content becomes stale (if it is cached).

This makes Linked Cache Invalidation useful in a number of cases, but not all; when it's important that changes be propagated quickly, the freshness lifetime of cached responses can be reduced, but there will still be lag.

When multiple caches are close together, the HyperText Caching Protocol (HTCP) [RFC2756] can be used to propagate invalidation events between caches, reducing (but not eliminating) these effects.

2. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This document uses the Augmented Backus-Naur Form (ABNF) notation of [RFC2616], and explicitly includes the following rules from it: delta-seconds.

3. The 'invalidates' link relation type

The 'invalidates' link relation type allows a response that is an signifies a state change on the server to indicate one or more associated URIs whose states have also changed.

*Relation name: invalidates

*Description: Indicates that when the link context changes, the link target also has changed.

*Reference: [this document]

*Notes:

4. The 'inv-by' link relation type

The 'inv-by' link relation type allows a response to nominate one or more other resources that affect the state of the resource it's associated with. That is, when one of the nominated resources changes, it also changes the state of this response's resource.

*Relation name: inv-by

*Description: Indicates that when the link target changes, the link's context has also changed.

*Reference: [this document]

*Notes:

5. The 'inv-maxage' response cache-control extension

When present, the 'inv-maxage' cache-control extension indicates the number of seconds that caches who implement Linked Cache invalidation can consider responses fresh for.

"inv-maxage" "=" delta-seconds

HTTP caches MAY, if they fully implement this specification, disregard the HTTP response cache-control directives 'no-cache', 'max-age' and 's-maxage' and use the value of inv-maxage as a replacement for max-age.
HTTP caches using inv-maxage to calculate freshness MUST invalidate all stored responses whose request-URIs (after normalisation) are indicated by the 'invalidates' link relation type contained in a successful response to a state-changing request, provided that they are allowed.
HTTP caches using inv-maxage to calculate freshness MUST invalidate all stored responses containing the 'inv-by' relation that indicates the current request-URI (after normalisation) upon receipt of a successful response to a state-changing request.
Here, a response is considered to "contain" a link relation if it is carried in the Link HTTP header [RFC5988]. I.e., it is not necessary to look at the response body.

"Invalidate" means that the cache will either remove all stored responses related to the effective request URI, or will mark these as "invalid" and in need of a mandatory validation before they can be returned in response to a subsequent request.
A "successful" response is one with a 2xx or redirecting 3xx (e.g., 301, 302, 303, 307) status code.
A "state-changing" request is one with an unsafe method (e.g., POST, PUT, DELETE, PATCH), or one that is not known to be safe.
In this context, "normalisation" means, in the case of a relative request-URI, that it is absolutised using the value of the Host request header and the appropriate protocol scheme.
Finally, an invalidation based upon "invalidates" is "allowed" if the host part of the request-URI (if absolute) or Host request header (if the request-URI is relative) matches the host part of the target URI. This prevents some types of denial-of-service attacks.
Implementations SHOULD effect invalidations when they become aware of changes through other means; e.g., HTCP [RFC2756] CLR messages, upon invalidations caused by other links (i.e., chained "cascades" of linked invalidations), or when a changed response is seen (such as when HTTP validation is unsuccessful).

6. Security Considerations

Linked Cache Invalidation does not guarantee that invalidations will be effected; e.g., they can be lost due to network issues or cache downtime. Furthermore, it does not guarantee that all caches that understand LCI will be made aware of invalidations that happen, because of how they originate.
Therefore, care should be taken that LCI invalidations are not relied upon (e.g., to purge sensitive content).
Furthermore, while some care is taken to avoid denial-of-service attacks through invalidation, cache efficiency may still be impaired under certain circumstances (e.g., arranging for one request to invalidate a large number of responses), leading to a reduction in service quality.

7. IANA Considerations

This document registers two entries in the Link Relation Type Registry; see Section 3 and Section 4.

8. References

8.1. Normative References

[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[RFC2616]	

	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
[RFC5988]	Nottingham, M., "Web Linking", RFC 5988, October 2010.

8.2. Informative References

[RFC2756]	Vixie, P. and D. Wessels, "Hyper Text Caching Protocol (HTCP/0.0)", RFC 2756, January 2000.

Appendix A. Acknowledgements

Thanks to Michael Hausenblas for his input.
The authors take all responsibility for errors and omissions.

Authors' Addresses

Mark Nottingham Nottingham EMail: mnot@mnot.net URI: http://www.mnot.net/

Mike Kelly Kelly EMail: mike@stateless.co URI: http://stateless.co/