Connect (/session/new?target=https://symfony.com/doc/current/http_cache/validation.html)

Symfony (/)

SymfonyLive (https://live.symfony.com/)

LILLE	PARIS	TUNIS	WARSZAWA	
February 28	March 26–27	April 18	June 11–12	Se
(https://lille2020.live.symfony.com/)	(https://paris2020.live.symfony.com/)	(https://tunis2020.live.symfony.com/)	(https://warszawa2020.live.symfony.com)	(https://berlin20

Home (/) / Documentation (/doc/current/index.html) / Http Cache (/doc/current/http_cache.html)

HTTP Cache Validation

5.0 version   edit this page
(https://github.com/symfony/symfony-
docs/edit/5.0/http_cache/validation.rst)

- Validation with the ETag Header
- Validation with the Last–Modified Header
- Optimizing your Code with Validation

When a resource needs to be updated as soon as a change is made to the underlying data, the expiration model falls short. With the expiration model (https://tools.ietf.org/html/rfc2616#section-13.2), the application won't be asked to return the updated response until the cache finally becomes stale.

The validation model addresses this issue. Under this model, the cache continues to store responses. The difference is that, for each request, the cache asks the application if the cached response is still valid or if it needs to be regenerated. If the cache *is* still valid, your application should return a 304 status code and no content. This tells the cache that it's OK to return the cached response.

Under this model, you only save CPU if you're able to determine that the cached response is still valid by doing *less* work than generating the whole page again (see below for an implementation example).

> **Tip**
>
> The 304 status code means "Not Modified". It's important because with this status code the response does *not* contain the actual content being requested. Instead, the response only consists of the response headers that tells the cache that it can use its stored version of the content.

Like with expiration, there are two different HTTP headers that can be used to implement the validation model: `ETag` and `Last-Modified` .

Expiration and Validation

You can use both validation and expiration within the same `Response`. As expiration wins over validation, you can benefit from the best of both worlds. In other words, by using both expiration and validation, you can instruct the cache to serve the cached content, while checking back at some interval (the expiration) to verify that the content is still valid.

> **Tip**
>
> You can also define HTTP caching headers for expiration and validation by using annotations. See the FrameworkExtraBundle documentation (https://symfony.com/doc/current/bundles/SensioFrameworkExtraBundle/annotations/cache.html).

Validation with the `ETag` Header ¶

The HTTP ETag (https://en.wikipedia.org/wiki/HTTP_ETag) ("entity-tag") header is an optional HTTP header whose value is an arbitrary string that uniquely identifies one representation of the target resource. It's entirely generated and set by your application so that you can tell, for example, if the `/about` resource that's stored by the cache is up-to-date with what your application would return.

An `ETag` is like a fingerprint and is used to quickly compare if two different versions of a resource are equivalent. Like fingerprints, each `ETag` must be unique across all representations of the same resource.

To see a simple implementation, generate the `ETag` as the `md5` of the content:

```
1   // src/Controller/DefaultController.php
2   namespace App\Controller;
3
4   use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
5   use Symfony\Component\HttpFoundation\Request;
6
7   class DefaultController extends AbstractController
8   {
9       public function homepage(Request $request)
10      {
11          $response = $this->render('static/homepage.html.twig');
12          $response->setEtag(md5($response->getContent()));
13          $response->setPublic(); // make sure the response is public/cacheable
14          $response->isNotModified($request);
15
16          return $response;
17      }
18  }
```

The `isNotModified()` (https://github.com/symfony/symfony/blob/5.0/src/Symfony/Component/HttpFoundation/Response.php) method compares the `If-None-Match` header with the `ETag` response header. If the two match, the method automatically sets the `Response` status code to 304.

> **Note**
>
> When using `mod_deflate` or `mod_brotli` in Apache 2.4, the original `ETag` value is modified (e.g. if `ETag` was `foo`, Apache turns it into `foo-gzip` or `foo-br`), which breaks the `ETag`–based validation.
>
> You can control this behavior with the DeflateAlterETag (https://httpd.apache.org/docs/trunk/mod/mod_deflate.html#deflatealteretag) and BrotliAlterETag (https://httpd.apache.org/docs/2.4/mod/mod_brotli.html#brotlialteretag) directives. Alternatively, you can use the following Apache configuration to keep both the original `ETag` and the modified one when compressing responses:
>
> ```
> 1   RequestHeader edit "If-None-Match" '^"((.*)-(gzip|br))"$' '"$1", "$2"'
> ```

> **Note**
>
> The cache sets the `If-None-Match` header on the request to the `ETag` of the original cached response before sending the request back to the app. This is how the cache and server communicate with each other and decide whether or not the resource has been updated since it was cached.

This algorithm is simple enough and very generic, but you need to create the whole `Response` before being able to compute the `ETag`, which is sub–optimal. In other words, it saves on bandwidth, but not CPU cycles.

In the Optimizing your Code with Validation section, you'll see how validation can be used more intelligently to determine the validity of a cache without doing so much work.

> **Tip**
>
> Symfony also supports weak `ETag``s by passing ``true` as the second argument to the `setEtag()` (https://github.com/symfony/symfony/blob/5.0/src/Symfony/Component/HttpFoundation/Response.php) method.

Validation with the Last-Modified Header ¶

The `Last-Modified` header is the second form of validation. According to the HTTP specification, "The `Last-Modified` header field indicates the date and time at which the origin server believes the representation was last modified." In other words, the application decides whether or not the cached content has been updated based on whether or not it's been updated since the response was cached.

For instance, you can use the latest update date for all the objects needed to compute the resource representation as the value for the `Last-Modified` header value:

```
1   // src/Controller/ArticleController.php
2   namespace App\Controller;
3
4   // ...
5   use App\Entity\Article;
6   use Symfony\Component\HttpFoundation\Request;
7   use Symfony\Component\HttpFoundation\Response;
8
9   class ArticleController extends AbstractController
10  {
11      public function show(Article $article, Request $request)
12      {
13          $author = $article->getAuthor();
14
15          $articleDate = new \DateTime($article->getUpdatedAt());
16          $authorDate = new \DateTime($author->getUpdatedAt());
17
18          $date = $authorDate > $articleDate ? $authorDate : $articleDate;
19
20          $response = new Response();
21          $response->setLastModified($date);
22          // Set response as public. Otherwise it will be private by default.
23          $response->setPublic();
24
25          if ($response->isNotModified($request)) {
26              return $response;
27          }
28
29          // ... do more work to populate the response with the full content
30
31          return $response;
32      }
33  }
```

The `isNotModified()` (https://github.com/symfony/symfony/blob/5.0/src/Symfony/Component/HttpFoundation/Response.php) method compares the `If-Modified-Since` header with the `Last-Modified` response header. If they are equivalent, the `Response` will be set to a 304 status code.

Note

The cache sets the `If-Modified-Since` header on the request to the `Last-Modified` of the original cached response before sending the request back to the app. This is how the cache and server communicate with each other and decide whether or not the resource has been updated since it was cached.

Optimizing your Code with Validation ¶

The main goal of any caching strategy is to lighten the load on the application. Put another way, the less you do in your application to return a 304 response, the better. The `Response::isNotModified()` method does exactly that by exposing a simple and efficient pattern:

```
1    // src/Controller/ArticleController.php
2    namespace App\Controller;
3
4    // ...
5    use Symfony\Component\HttpFoundation\Request;
6    use Symfony\Component\HttpFoundation\Response;
7
8    class ArticleController extends AbstractController
9    {
10       public function show($articleSlug, Request $request)
11       {
12           // Get the minimum information to compute
13           // the ETag or the Last-Modified value
14           // (based on the Request, data is retrieved from
15           // a database or a key-value store for instance)
16           $article = ...;
17
18           // create a Response with an ETag and/or a Last-Modified header
19           $response = new Response();
20           $response->setEtag($article->computeETag());
21           $response->setLastModified($article->getPublishedAt());
22
23           // Set response as public. Otherwise it will be private by default.
24           $response->setPublic();
25
26           // Check that the Response is not modified for the given Request
27           if ($response->isNotModified($request)) {
28               // return the 304 Response immediately
29               return $response;
30           }
31
32           // do more work here - like retrieving more data
33           $comments = ...;
34
35           // or render a template with the $response you've already started
36           return $this->render('article/show.html.twig', [
37               'article' => $article,
38               'comments' => $comments,
39           ], $response);
40       }
41   }
```

When the `Response` is not modified, the `isNotModified()` automatically sets the response status code to `304`, removes the content, and removes some headers that must not be present for `304` responses (see `setNotModified()` (https://github.com/symfony/symfony/blob/5.0/src/Symfony/Component/HttpFoundation/Response.php)).

Latest from the Symfony Blog

SymfonyLive Lille 2020: join us next week! (/blog/symfonylive-lille-2020-join-us-next-week)
February 21, 2020

They Help Us Make Symfony

Thanks Jacek Jędrzejewski (https://connect.symfony.com/profile/jacek.jedrzejewski) for being a Symfony contributor (/contributors).

2 commits · 8 lines

Get Involved in the Community

A passionate group of over 600,000 developers from more than 120 countries, all committed to helping PHP surpass the impossible.

Getting involved →

What is Symfony? (/what-is-symfony)
Symfony at a Glance (/at-a-glance)
Symfony Components (/components)
Case Studies (/blog/category/case-studies)
Symfony Releases (/releases)
Security Policy (/doc/current/contributing/code/security.html)
Logo & Screenshots (/logo)
Trademark & Licenses (/license)
symfony1 Legacy (/legacy)

Learn Symfony (/doc/current/index.html)
Getting Started (/doc/5.0/setup.html)
Components (/doc/5.0/components/index.html)
Best Practices (/doc/5.0/best_practices/index.html)
Bundles (/doc/bundles/)
Reference (/doc/5.0/reference/index.html)
Training (https://training.sensiolabs.com/en/courses?q=symfony)
Certification (https://certification.symfony.com/)

Screencasts (https://symfonycasts.com)
Learn Symfony (https://symfonycasts.com/tracks/symfony)
Learn PHP (https://symfonycasts.com/tracks/php)

Community (/community)
SymfonyConnect (https://connect.symfony.com/)
Support (/support)

Learn JavaScript (https://symfonycasts.com/tracks/javascript)

Learn Drupal (https://symfonycasts.com/tracks/drupal)

Learn RESTful APIs (https://symfonycasts.com/tracks/rest)

How to be Involved (/doc/current/contributing/index.html)

Code of Conduct (/doc/current/contributing/code_of_conduct/code_of_conduct.html)

Events & Meetups (/events/)

Projects using Symfony (/projects)

Downloads Stats (/stats/downloads)

Contributors (/contributors)

Blog (/blog/)

Events & Meetups (/events)

A week of symfony (/blog/category/a-week-of-symfony)

Case studies (/blog/category/case-studies)

Cloud (/blog/category/cloud)

Community (/blog/category/community)

Conferences (/blog/category/conferences)

Diversity (/blog/category/diversity)

Documentation (/blog/category/documentation)

Living on the edge (/blog/category/living-on-the-edge)

Releases (/blog/category/releases)

Security Advisories (/blog/category/security-advisories)

SymfonyInsight (/blog/category/symfony-insight)

Twig (/blog/category/twig)

Services (https://sensiolabs.com)

Our services (https://sensiolabs.com)

Train developers (https://training.sensiolabs.com/en)

Manage your project quality (https://insight.symfony.com/)

Improve your project performance (https://blackfire.io/?utm_source=symfony&utm_medium=symfonycom_footer&utm_campaign=profiler)

Host Symfony projects (/cloud/)

About (/about)

SensioLabs (https://sensiolabs.com/en/join_us/join_us.html)

Careers (https://sensiolabs.com/en/join_us/join_us.html)

Support (/support)

Cloud TOS (/cloud/tos)

Deployed on

sf

(/cloud/)

Follow Symfony

(https://github.com/symfony) (https://stackoverflow.com/questions/tagged/symfony) (/slack) (https://twitter.com/symfony) (https://www.facebook.com/SymfonyFramework) (https://www.youtube.com/user/SensioLabs) (https://symfonycasts.com/) (https://feeds.feedburner.com/symfony/blog)

Switch to dark theme