

 This page was translated from English by the community. [Learn more and join the MDN Web Docs community.](#)

# HTTP caching

## Visão geral


O cache HTTP armazena uma resposta associada a uma solicitação e reutiliza a resposta armazenada para solicitações subsequentes.

Existem várias vantagens na reutilização. Primeiro, como não há necessidade de entregar a solicitação ao servidor de origem, quanto mais próximos o cliente e o cache estiverem, mais rápida será a resposta. O exemplo mais típico é quando o próprio navegador armazena um cache para solicitações do navegador.

Além disso, quando uma resposta é reutilizável, o servidor de origem não precisa processar a solicitação — portanto, não precisa analisar e rotear a solicitação, restaurar a sessão com base no cookie, consultar o banco de dados para obter resultados ou renderizar o mecanismo de modelo . Isso reduz a carga no servidor.

A operação adequada do cache é fundamental para a integridade do sistema.

## Tipos de caches

Na especificação [HTTP Caching](#) , existem dois tipos principais de caches: **caches privados** e **caches compartilhados**.

### Caches privados

Um cache privado é um cache vinculado a um cliente específico — normalmente um cache de navegador. Como a resposta armazenada não é compartilhada com outros clientes, um cache privado pode armazenar uma resposta personalizada para esse usuário.

Por outro lado, se o conteúdo personalizado for armazenado em um cache que não seja um cache privado, outros usuários poderão recuperar esse conteúdo - o que pode causar vazamento não intencional de informações.

Se uma resposta contém conteúdo personalizado e você deseja armazenar a resposta apenas no cache privado, você deve especificar uma diretiva `private`.

```
Cache-Control: private
```



Os conteúdos personalizados são geralmente controlados por cookies, mas a presença de um cookie nem sempre indica que é privado e, portanto, um cookie por si só não torna a resposta privada.

Observe que se a resposta tiver um cabeçalho `Authorization`, ela não poderá ser armazenada no cache privado (ou em um cache compartilhado, a menos que `public` seja especificado).

## Cache compartilhado

O cache compartilhado está localizado entre o cliente e o servidor e pode armazenar respostas que podem ser compartilhadas entre os usuários. E os caches compartilhados podem ser subdivididos em **caches de proxy** e **caches gerenciados**.

## Caches de proxy

Além da função de controle de acesso, alguns proxies implementam cache para reduzir o tráfego fora da rede. Isso geralmente não é gerenciado pelo desenvolvedor do serviço, portanto, deve ser controlado por cabeçalhos HTTP apropriados e assim por diante. No entanto, no passado, implementações de cache de proxy desatualizadas — como implementações que não entendem adequadamente o padrão HTTP Caching — geralmente causavam problemas para os desenvolvedores.

**Kitchen-sink headers** como os seguintes são usados para tentar contornar implementações de "cache de proxy antigo e não atualizado" que não entendem as diretivas atuais de especificação de cache HTTP, como `no-store`.

**Cache-Control:** no-store, no-cache, max-age=0, must-revalidate, proxy-revalidate



No entanto, nos últimos anos, à medida que o HTTPS se tornou mais comum e a comunicação cliente/servidor tornou-se criptografada, os caches de proxy no caminho só podem encapsular uma resposta e não podem se comportar como um cache, em muitos casos. Portanto, nesse cenário, não há necessidade de se preocupar com implementações de cache de proxy desatualizadas que nem conseguem ver a resposta.

Por outro lado, se um proxy de ponte [TLS](#) descriptografa todas as comunicações de maneira intermediária instalando um certificado de uma [CA \(entidade certificadora\) \(en-US\)](#) gerenciada pela organização no PC e executa o controle de acesso, etc. — é possível ver o conteúdo da resposta e armazená-la em cache. No entanto, como [CT \(transparência do certificado\) \(en-US\)](#) se tornou comum nos últimos anos e alguns navegadores permitem apenas certificados emitidos com um SCT (carimbo de data e hora do certificado assinado), esse método exige a aplicação de uma política empresarial. Em um ambiente tão controlado, não há necessidade de se preocupar com o cache do proxy estar "desatualizado e não atualizado".

## Caches gerenciados

Os caches gerenciados são implantados explicitamente por desenvolvedores de serviços para descarregar o servidor de origem e fornecer conteúdo com eficiência. Os exemplos incluem proxies reversos, CDNs e service workers em combinação com a API de cache.

As características dos caches gerenciados variam dependendo do produto implementado. Na maioria dos casos, você pode controlar o comportamento do cache através do cabeçalho `Cache-Control` e seus próprios arquivos de configuração ou painéis.

Por exemplo, a especificação HTTP Caching essencialmente não define uma maneira de excluir explicitamente um cache — mas com um cache gerenciado, a resposta armazenada pode ser excluída a qualquer momento por meio de operações de painel,

chamadas de API, reinicializações e assim por diante. Isso permite uma estratégia de cache mais proativa.

Também é possível ignorar os protocolos de especificação de cache HTTP padrão em favor da manipulação explícita. Por exemplo, o seguinte pode ser especificado para desativar um cache privado ou cache proxy, ao usar sua própria estratégia para armazenar em cache apenas em um cache gerenciado.

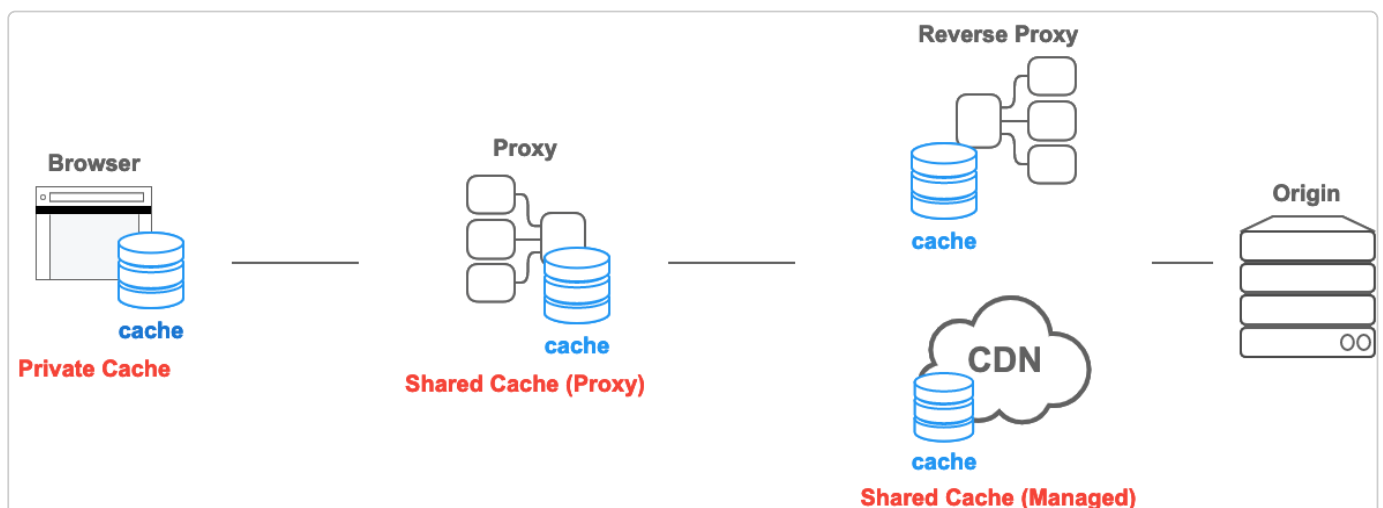
```
Cache-Control: no-store
```



Por exemplo, o Varnish Cache usa VCL (Varnish Configuration Language, um tipo de lógica [DSL \(en-US\)](#)) para lidar com o armazenamento em cache, enquanto os service workers em combinação com a Cache API permitem que você crie essa lógica em JavaScript.

Isso significa que se um cache gerenciado ignorar intencionalmente uma diretiva `no-store`, não há necessidade de percebê-lo como "não compatível" com o padrão. O que você deve fazer é evitar o uso de cabeçalhos de pia de cozinha, mas leia atentamente a documentação de qualquer mecanismo de cache gerenciado que estiver usando e verifique se está controlando o cache adequadamente da maneira fornecida pelo mecanismo escolhido para usar.

Observe que alguns CDNs fornecem seus próprios cabeçalhos que são eficazes apenas para esse CDN (por exemplo, `Surrogate-Control`). Atualmente, o trabalho está em andamento para definir um cabeçalho [CDN-Cache-Control](#) [✉](#) para padronizá-los.



## Cache heurístico

O HTTP foi projetado para armazenar em cache o máximo possível, portanto, mesmo que nenhum `Cache-Control` seja fornecido, as respostas serão armazenadas e reutilizadas se determinadas condições forem atendidas. Isso é chamado de **caching heurístico**.

Por exemplo, tome a seguinte resposta. Esta resposta foi atualizada pela última vez há 1 ano.

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
```



 mdn web docs

...

Sabe-se heurísticamente que o conteúdo que não foi atualizado por um ano inteiro não será atualizado por algum tempo depois disso. Portanto, o cliente armazena essa resposta (apesar da falta de `max-age`) e a reutiliza por um tempo. Quanto tempo para reutilizar depende da implementação, mas a especificação recomenda cerca de 10% (neste caso 0,1 ano) do tempo após o armazenamento.

O cache heurístico é uma solução alternativa que veio antes que o suporte `Cache-Control` fosse amplamente adotado, e basicamente todas as respostas deveriam especificar explicitamente um cabeçalho `Cache-Control`.

## Fresh e stale com base na idade

As respostas HTTP armazenadas têm dois estados: **fresh** e **stale**. O estado *fresh* geralmente indica que a resposta ainda é válida e pode ser reutilizada, enquanto o estado *stale* significa que a resposta em cache já expirou.

O critério para determinar quando uma resposta é recente e quando está desatualizada é **age**. Em HTTP, a idade é o tempo decorrido desde que a resposta foi gerada. Isso é semelhante ao [TTL \(en-US\)](#) em outros mecanismos de cache.

Veja a seguinte resposta de exemplo (604800 segundos é uma semana):

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
Cache-Control: max-age=604800
```

```
<!doctype html>
...
```

O cache que armazenou a resposta de exemplo calcula o tempo decorrido desde que a resposta foi gerada e usa o resultado como *age* da resposta.

Para a resposta de exemplo, o significado de `max-age` é o seguinte:

- Se a idade da resposta for *menos* de uma semana, a resposta será *fresh*.
- Se a idade da resposta for *mais* de uma semana, a resposta será *stale*.

Enquanto a resposta armazenada permanecer atualizada, ela será usada para atender às solicitações do cliente.

Quando uma resposta é armazenada em um cache compartilhado, é necessário informar ao cliente a idade da resposta. Continuando com o exemplo, se o cache compartilhado armazenasse a resposta por um dia, o cache compartilhado enviaria a seguinte resposta para solicitações de clientes subsequentes.

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
Cache-Control: max-age=604800
Age: 86400
```

```
<!doctype html>
...
```

O cliente que receber essa resposta a achará atualizada pelos 518.400 segundos restantes, a diferença entre a `max-age` e a `Age` da resposta.

# Expires ou max-age

No HTTP/1.0, o frescor costumava ser especificado pelo cabeçalho `Expires`.

O cabeçalho `Expires` especifica o tempo de vida do cache usando um tempo explícito em vez de especificar um tempo decorrido.

`Expires`: Tue, 28 Feb 2022 22:22:22 GMT



No entanto, o formato de hora é difícil de analisar, muitos bugs de implementação foram encontrados e é possível induzir problemas alterando intencionalmente o relógio do sistema; portanto, `max-age` — para especificar um tempo decorrido — foi adotado para `Cache-Control` em HTTP/1.1.

Se `Expires` e `Cache-Control: max-age` estiverem disponíveis, `max-age` será definido como preferencial. Portanto, não é necessário fornecer `Expires` agora que o HTTP/1.1 é amplamente utilizado.

## Vary

A maneira como as respostas são diferenciadas umas das outras é essencialmente baseada em seus URLs:



URL	Response
<code>https://example.com/index.html</code>	<code>&lt;!doctype html&gt;...</code>
<code>https://example.com/style.css</code>	<code>body { ...</code>
<code>https://example.com/script.js</code>	<code>function main () { ...</code>

Mas o conteúdo das respostas nem sempre é o mesmo, mesmo que tenham a mesma URL. Especialmente quando a negociação de conteúdo é realizada, a resposta do servidor pode depender dos valores dos cabeçalhos de solicitação `Accept`, `Accept-Language` e `Accept-Encoding`.

Por exemplo, para conteúdo em inglês retornado com um cabeçalho `Accept-Language: en` e armazenado em cache, é indesejável reutilizar essa resposta em cache para solicitações

que tenham um cabeçalho de solicitação `Accept-Language: ja`. Nesse caso, você pode fazer com que as respostas sejam armazenadas em cache separadamente — com base no idioma — adicionando "`Accept-Language`" ao valor do cabeçalho `Vary`.

`Vary: Accept-Language`



Isso faz com que o cache seja codificado com base em uma composição do URL de resposta e no cabeçalho de solicitação `Accept-Language` — em vez de ser baseado apenas no URL de resposta.



URL	Accept-Language	Response
<code>https://example.com/index.html</code>	<code>ja-JP</code>	<code>&lt;!doctype html&gt;...</code>
<code>https://example.com/index.html</code>	<code>en-US</code>	<code>&lt;!doctype html&gt;...</code>
<code>https://example.com/style.css</code>	<code>ja-JP</code>	<code>body { ...</code>
<code>https://example.com/script.js</code>	<code>ja-JP</code>	<code>function main () { ...</code>

Além disso, se você estiver fornecendo otimização de conteúdo (por exemplo, para design responsivo) com base no agente do usuário, pode ser tentado a incluir "`User-Agent`" no valor do cabeçalho `vary`. No entanto, o cabeçalho de solicitação `User-Agent` geralmente tem um número muito grande de variações, o que reduz drasticamente a chance de o cache ser reutilizado. Portanto, se possível, considere uma maneira de variar o comportamento com base na detecção de recursos, em vez de com base no cabeçalho da solicitação `User-Agent`.

Para aplicativos que empregam cookies para impedir que outros reutilizem conteúdo personalizado em cache, você deve especificar `Cache-Control: private` em vez de especificar um cookie para `vary`.

## Validação

As respostas obsoletas não são descartadas imediatamente. O HTTP tem um mecanismo para transformar uma resposta obsoleta em uma nova, perguntando ao servidor de origem. Isso é chamado de **validação** ou, às vezes, **revalidação**.



A validação é feita usando uma **solicitação condicional** que inclui um cabeçalho de solicitação `If-Modified-Since` ou `If-None-Match`.

## If-Modified-Since

A resposta a seguir foi gerada às 22:22:22 e tem um `max-age` de 1 hora, então você sabe que ela é atualizada até 23:22:22.

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
Last-Modified: Tue, 22 Feb 2022 22:00:00 GMT
Cache-Control: max-age=3600

<!doctype html>
...
```



Às 23:22:22, a resposta se torna obsoleta e o cache não pode ser reutilizado. Portanto, a solicitação abaixo mostra um cliente enviando uma solicitação com um cabeçalho de solicitação `If-Modified-Since`, para perguntar ao servidor se houve alguma alteração feita desde o horário especificado.

```
GET /index.html HTTP/1.1
Host: example.com
Accept: text/html
If-Modified-Since: Tue, 22 Feb 2022 22:00:00 GMT
```



O servidor responderá com `304 Not Modified` se o conteúdo não mudou desde o horário especificado.

Como essa resposta indica apenas "sem alteração", não há corpo de resposta - há apenas um código de status - portanto, o tamanho da transferência é extremamente pequeno.

```
HTTP/1.1 304 Not Modified
Content-Type: text/html
Date: Tue, 22 Feb 2022 23:22:22 GMT
Last-Modified: Tue, 22 Feb 2022 22:00:00 GMT
Cache-Control: max-age=3600
```



Ao receber essa resposta, o cliente reverte a resposta antiga armazenada e pode reutilizá-la durante a 1 hora restante.

O servidor pode obter o tempo de modificação do sistema de arquivos do sistema operacional, o que é relativamente fácil de fazer no caso de servir arquivos estáticos. No entanto, existem alguns problemas; por exemplo, o formato de hora é complexo e difícil de analisar, e os servidores distribuídos têm dificuldade em sincronizar os horários de atualização de arquivo.

Para resolver tais problemas, o cabeçalho de resposta `ETag` foi padronizado como alternativa.

## ETag/If-None-Match

O valor do cabeçalho de resposta `ETag` é um valor arbitrário gerado pelo servidor. Não há restrições sobre como o servidor deve gerar o valor, portanto, os servidores são livres para definir o valor com base em qualquer meio que escolherem - como um hash do conteúdo do corpo ou um número de versão.

Como exemplo, se um valor de hash for usado para o cabeçalho `ETag` e o valor de hash do recurso `index.html` for `33a64df5`, a resposta será a seguinte:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
ETag: "33a64df5"
Cache-Control: max-age=3600

<!doctype html>
...
```



Se essa resposta estiver obsoleta, o cliente pega o valor do cabeçalho de resposta `ETag` para a resposta em cache e o coloca no cabeçalho de solicitação `If-None-Match`, para perguntar ao servidor se o recurso foi modificado:

```
GET /index.html HTTP/1.1
Host: example.com
```



```
Accept: text/html
If-None-Match: "33a64df5"
```

O servidor retornará `304 Not Modified` se o valor do cabeçalho `ETag` que ele determinar para o recurso solicitado for o mesmo que o valor `If-None-Match` na solicitação.

Mas se o servidor determinar que o recurso solicitado deve ter agora um valor 'ETag' diferente, o servidor responderá com um '200 OK' e a versão mais recente do recurso.

**Nota:** ao avaliar como usar `ETag` e `Last-Modified`, considere o seguinte: Durante a revalidação de cache, se ambos `ETag` e `Last-Modified` estiverem presentes, `ETag` terá precedência. Portanto, se você está considerando apenas o armazenamento em cache, você pode pensar que 'Last-Modified' é desnecessário. No entanto, `Last-Modified` não é apenas útil para cache; em vez disso, é um cabeçalho HTTP padrão que também é usado por sistemas de gerenciamento de conteúdo (CMS) para exibir a hora da última modificação, por rastreadores para ajustar a frequência de rastreamento e para outras finalidades. Portanto, considerando o ecossistema HTTP geral, é preferível fornecer `ETag` e `Last-Modified`.

## Forçar revalidação

Se você não quer que uma resposta seja reutilizada, mas quer sempre buscar o conteúdo mais recente do servidor, você pode usar a diretiva `no-cache` para forçar a validação.

Ao adicionar `Cache-Control: no-cache` à resposta junto com `Last-Modified` e `ETag` — como mostrado abaixo — o cliente receberá uma resposta `200 OK` se o recurso solicitado foi atualizado ou caso contrário, receba uma resposta `304 Not Modified` se o recurso solicitado não tiver sido atualizado.

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
Last-Modified: Tue, 22 Feb 2022 22:00:00 GMT
ETag: deadbeef
```



```
Cache-Control: no-cache
```

```
<!doctype html>
```

```
...
```

Costuma-se dizer que a combinação de `max-age=0` e `must-revalidate` tem o mesmo significado de `no-cache`.

```
Cache-Control: max-age=0, must-revalidate
```



`max-age=0` significa que a resposta é imediatamente obsoleta, e `must-revalidate` significa que ela não deve ser reutilizada sem revalidação uma vez que está obsoleta - então, em combinação, a semântica parece ser a mesma que `no-cache`.

No entanto, esse uso de `max-age=0` é um resquício do fato de que muitas implementações anteriores ao HTTP/1.1 eram incapazes de lidar com a diretiva `no-cache` — e, portanto, para lidar com essa limitação, `max-age =0` foi usado como solução alternativa.

Mas agora que os servidores compatíveis com HTTP/1.1 são amplamente implantados, não há razão para usar essa combinação `max-age=0` e `must-revalidate` - você deve usar apenas `no-cache`.

## Não armazenar em cache

A diretiva `no-cache` não impede o armazenamento de respostas, mas impede a reutilização de respostas sem revalidação.

Se você não quiser uma resposta armazenada em nenhum cache, use `no-store`.

```
Cache-Control: no-store
```



No entanto, em geral, um requisito de "não armazenar em cache" na prática equivale ao seguinte conjunto de circunstâncias:

- Não deseja que a resposta seja armazenada por outra pessoa que não seja o cliente específico, por questões de privacidade.

- Quer fornecer informações atualizadas sempre.
- Não sei o que pode acontecer em implementações desatualizadas.

Sob esse conjunto de circunstâncias, `no-store` nem sempre é a diretiva mais apropriada.

As seções a seguir examinam as circunstâncias com mais detalhes.

## Não compartilhe com outras pessoas

Seria problemático se uma resposta com conteúdo personalizado fosse inesperadamente visível para outros usuários de um cache.

Nesse caso, usar a diretiva `private` fará com que a resposta personalizada seja armazenada apenas com o cliente específico e não seja vazada para nenhum outro usuário do cache.

```
Cache-Control: private
```



Nesse caso, mesmo que `no-store` seja fornecido, `private` também deve ser fornecido.

## Forneça conteúdo atualizado sempre

A diretiva `no-store` impede que uma resposta seja armazenada, mas não exclui nenhuma resposta já armazenada para o mesmo URL.

Em outras palavras, se já houver uma resposta antiga armazenada para uma URL específica, retornar `no-store` não impedirá que a resposta antiga seja reutilizada.

No entanto, uma diretiva `no-cache` forçará o cliente a enviar uma solicitação de validação antes de reutilizar qualquer resposta armazenada.

```
Cache-Control: no-cache
```




Se o servidor não suportar solicitações condicionais, você pode forçar o cliente a acessar o servidor todas as vezes e sempre obter a resposta mais recente com `200 OK`.

## Lidando com implementações desatualizadas

Como solução para implementações desatualizadas que ignoram `no-store`, você pode ver cabeçalhos de pia de cozinha, como o seguinte, sendo usados.

```
Cache-Control: no-store, no-cache, max-age=0, must-revalidate, proxy-revalidate
```



É [recomendado](#)  usar `no-cache` como uma alternativa para lidar com essas implementações desatualizadas, e não há problema se `no-cache` for fornecido desde o início, pois o servidor sempre receberá a solicitação.

Se você está preocupado com o cache compartilhado, certifique-se de evitar o cache não intencional adicionando também `private`:

```
Cache-Control: no-cache, private
```



## O que é perdido por `no-store`

Você pode pensar que adicionar `no-store` seria a maneira correta de desativar o armazenamento em cache.

No entanto, não é recomendado conceder `no-store` liberalmente, porque você perde muitas vantagens que o HTTP e os navegadores têm, incluindo o cache de retorno/avanço do navegador.

Portanto, para obter as vantagens do conjunto completo de recursos da plataforma web, prefira o uso de `no-cache` em combinação com `private`.

## Recarregue e force o recarregamento

A validação pode ser realizada tanto para solicitações quanto para respostas.

As ações **reload** e **force reload** são exemplos comuns de validação realizada do lado do navegador.

## Recarregar

Para recuperar a corrupção da janela ou atualizar para a versão mais recente do recurso, os navegadores fornecem uma função de recarregamento para os usuários.

Uma visualização simplificada da solicitação HTTP enviada durante um recarregamento do navegador é a seguinte:


```
GET / HTTP/1.1
Host: example.com
Cache-Control: max-age=0
If-None-Match: "deadbeef"
If-Modified-Since: Tue, 22 Feb 2022 20:20:20 GMT
```



(As solicitações do Chrome, Edge e Firefox são muito parecidas com as acima; as solicitações do Safari serão um pouco diferentes.)

A diretiva `max-age=0` na solicitação especifica "reutilização de respostas com idade igual ou inferior a 0" — portanto, na verdade, as respostas armazenadas intermediárias não são reutilizadas.

Como resultado, uma solicitação é validada por `If-None-Match` e `If-Modified-Since`.

Esse comportamento também é definido no padrão [Fetch](#)  e pode ser reproduzido em JavaScript chamando `fetch()` com o modo de cache definido como `no-cache` (observe que `reload` não é o modo correto para este caso):

```
// Nota: "reload" não é o modo correto para um recarregamento normal; "no-cache" é
fetch("/", { cache: "no-cache" });
```



## Forçar recarga

Os navegadores usam `max-age=0` durante os recarregamentos por motivos de compatibilidade com versões anteriores — porque muitas implementações desatualizadas anteriores ao HTTP/1.1 não entendiam `no-cache`. Mas `no-cache` está bem agora neste caso de uso, e **forçar recarregamento** é uma forma adicional de contornar as respostas em cache.

A solicitação HTTP durante um **recarregamento forçado** do navegador tem a seguinte aparência:

```
GET / HTTP/1.1
Host: example.com
Pragma: no-cache
Cache-Control: no-cache
```



(As solicitações do Chrome, Edge e Firefox são muito parecidas com as acima; as solicitações do Safari serão um pouco diferentes.)

Como essa não é uma solicitação condicional com `no-cache`, você pode ter certeza de que receberá um `200 OK` do servidor de origem.

Esse comportamento também é definido no padrão [Fetch](#) e pode ser reproduzido em JavaScript chamando `fetch()` com o modo de cache definido como `reload` (observe que não é `force-reload`):

```
// Nota: "reload" — em vez de "no-cache" — é o modo correto para um "force
reload"fetch("/", { cache: "reload" });
```



## Evitando a revalidação

O conteúdo que nunca muda deve receber uma longa `max-age` usando cache busting — ou seja, incluindo um número de versão, valor de hash, etc., no URL da solicitação.

No entanto, quando o usuário recarrega, uma solicitação de revalidação é enviada mesmo que o servidor saiba que o conteúdo é imutável.

Para evitar isso, a diretiva `immutable` pode ser usada para indicar explicitamente que a revalidação não é necessária porque o conteúdo nunca muda.

```
Cache-Control: max-age=31536000, immutable
```



Isso evita revalidações desnecessárias durante as recargas.



Observe que, em vez de implementar essa diretiva, [o Chrome mudou sua implementação](#) para que a revalidação não seja executado durante recargas para sub-recursos.

## Excluindo respostas armazenadas

Basicamente, não há como excluir respostas que já foram armazenadas com um `max-age` longo.

Imagine que a seguinte resposta de `https://example.com/` foi armazenada.

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Cache-Control: max-age=31536000
```



```
<!doctype html>
```

```
...
```

Você pode querer sobrescrever essa resposta depois que ela expirar no servidor, mas não há nada que o servidor possa fazer depois que a resposta for armazenada - já que nenhuma solicitação chega ao servidor devido ao armazenamento em cache.

Um dos métodos mencionados na especificação é enviar uma solicitação para a mesma URL com um método não seguro como `POST`, mas isso geralmente é difícil de fazer intencionalmente para muitos clientes.

Há também uma especificação para um cabeçalho e valor `Clear-Site-Data: cache`, mas [nem todos os navegadores o suportam](#) — e mesmo quando usado, afeta apenas os caches do navegador e não afeta os caches intermediários.

Portanto, deve-se presumir que qualquer resposta armazenada permanecerá por seu período de `max-age`, a menos que o usuário execute manualmente uma ação de recarregar, forçar o recarregamento ou limpar o histórico.

O armazenamento em cache reduz o acesso ao servidor, o que significa que o servidor perde o controle dessa URL. Se o servidor não quiser perder o controle de uma URL — por exemplo, no caso de um recurso ser atualizado com frequência — você deve adicionar

`no-cache` para que o servidor sempre receba solicitações e envie as respostas pretendidas.

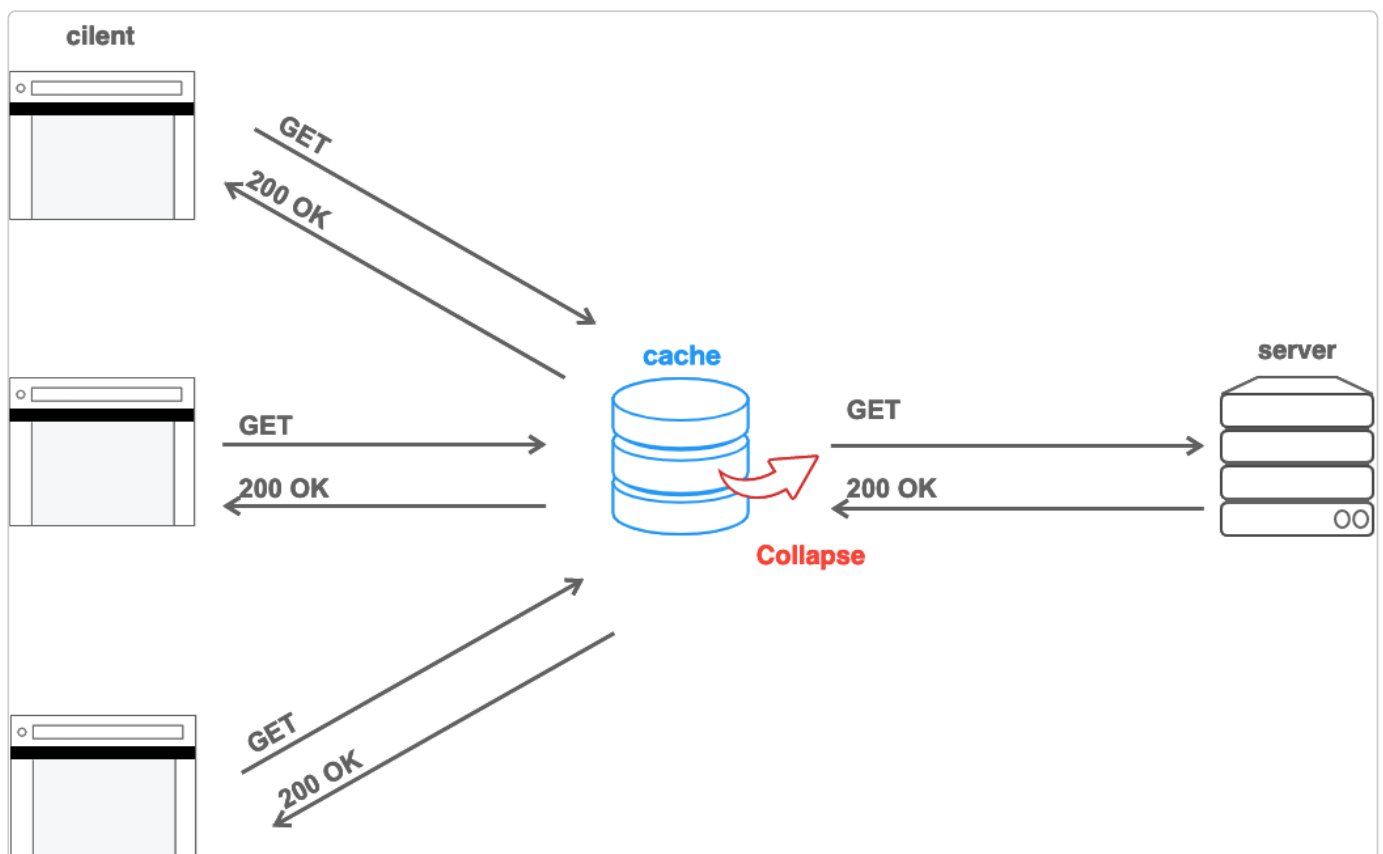
## Recolhimento da solicitação

O cache compartilhado está localizado principalmente antes do servidor de origem e destina-se a reduzir o tráfego para o servidor de origem.

Assim, se várias solicitações idênticas chegarem a um cache compartilhado ao mesmo tempo, o cache intermediário encaminhará uma única solicitação em seu nome para a origem, que poderá reutilizar o resultado para todos os clientes. Isso é chamado de **recolhimento da solicitação**.

O colapso da solicitação ocorre quando as solicitações chegam ao mesmo tempo, portanto, mesmo que `max-age=0` ou `no-cache` seja fornecido na resposta, ele será reutilizado.

Se a resposta for personalizada para um usuário específico e você não quiser que ela seja compartilhada em recolhimento, você deve adicionar a diretiva `private`:



# Padrões de cache comuns

Existem muitas diretivas na especificação `Cache-Control`, e pode ser difícil entender todas elas. Mas a maioria dos sites pode ser coberta por uma combinação de alguns padrões.

Esta seção descreve os padrões comuns no projeto de caches.

## Configurações padrão

Como mencionado acima, o comportamento padrão para armazenamento em cache (ou seja, para uma resposta sem `Cache-Control`) não é simplesmente "não armazenar em cache", mas cache implícito de acordo com o chamado "caching heurístico".

Para evitar esse cache heurístico, é preferível fornecer explicitamente a todas as respostas um cabeçalho padrão `Cache-Control`.

Para garantir que, por padrão, as versões mais recentes dos recursos sempre serão transferidas, é uma prática comum fazer com que o valor padrão de `Cache-Control` inclua `no-cache`:

```
Cache-Control: no-cache
```



Além disso, se o serviço implementa cookies ou outros métodos de login, e o conteúdo é personalizado para cada usuário, também deve ser fornecido `private`, para evitar o compartilhamento com outros usuários:

```
Cache-Control: no-cache, private
```



## Bloqueio de cache

Os recursos que funcionam melhor com cache são arquivos estáticos imutáveis cujo conteúdo nunca muda. E para recursos que *fazem* alterações, é uma prática recomendada comum alterar a URL sempre que o conteúdo for alterado, para que a unidade de URL possa ser armazenada em cache por um período mais longo.

Como exemplo, considere o seguinte HTML:

```
<script src="bundle.js"></script>
<link rel="stylesheet" href="build.css" />
<body>
  hello
</body>
```



No desenvolvimento web moderno, os recursos JavaScript e CSS são atualizados com frequência à medida que o desenvolvimento avança. Além disso, se as versões dos recursos JavaScript e CSS que um cliente usa estiverem fora de sincronia, a exibição será interrompida.

Portanto, o HTML acima dificulta o cache de `bundle.js` e `build.css` com `max-age`.

Portanto, você pode fornecer JavaScript e CSS com URLs que incluem uma parte de alteração com base em um número de versão ou valor de hash. Algumas das maneiras de fazer isso são mostradas abaixo.

```
# versão no nome do arquivo
bundle.v123.js

# versão na consulta
bundle.js?v=123

# hash no nome do arquivo
bundle.YsAIAAAA-QG4G6kCMAMBAAAAAAAOk.js


# hash na consulta
bundle.js?v=YsAIAAAA-QG4G6kCMAMBAAAAAAAOk
```

Como o cache distingue recursos uns dos outros com base em seus URLs, o cache não será reutilizado novamente se o URL for alterado quando um recurso for atualizado.

```
<script src="bundle.v123.js"></script>
<link rel="stylesheet" href="build.v123.css" />
<body>
  hello
</body>
```



Com esse design, os recursos JavaScript e CSS podem ser armazenados em cache por um longo tempo. Então, quanto tempo deve ser definido como `max-age`? A especificação QPACK fornece uma resposta a essa pergunta.

[QPACK](#)  é um padrão para compactar campos de cabeçalho HTTP, com tabelas de valores de campo comumente usados definidas.

Alguns valores de cabeçalho de cache comumente usados são mostrados abaixo.

```
36 cache-control max-age=0
37 cache-control max-age=604800
38 cache-control max-age=2592000
39 cache-control no-cache
40 cache-control no-store
41 cache-control public, max-age=31536000
```


Se você selecionar uma dessas opções numeradas, poderá compactar valores em 1 byte quando transferidos via HTTP3.

Os números `37`, `38` e `41` são para períodos de uma semana, um mês e um ano.

Como o cache remove entradas antigas quando novas entradas são salvas, a probabilidade de que uma resposta armazenada ainda exista após uma semana não é tão alta — mesmo se `max-age` for definido como 1 semana. Portanto, na prática, não faz muita diferença qual você escolhe.

Observe que o número `41` tem a `max-age` mais longa (1 ano), mas com `public`.

O valor `public` tem o efeito de tornar a resposta armazenável mesmo se o cabeçalho `Authorization` estiver presente.

 **Nota:** A diretiva `public` só deve ser usada se houver necessidade de armazenar a resposta quando o cabeçalho `Authorization` for definido. Não é necessário de outra forma, porque uma resposta será armazenada no cache compartilhado enquanto `max-age` for fornecido.

Portanto, se a resposta for personalizada com autenticação básica, a presença de `público` pode causar problemas. Se estiver preocupado com isso, você pode escolher o segundo valor mais longo, `38` (1 mês).

```
# resposta para bundle.v123.js

# Se você nunca personalizar as respostas por meio de autorização
Cache-Control: public, max-age=31536000

# Se você não pode ter certeza
Cache-Control: max-age=2592000
```



## Validação

Não se esqueça de definir os cabeçalhos `Last-Modified` e `ETag`, para que você não precise retransmitir um recurso ao recarregar. É fácil gerar esses cabeçalhos para arquivos estáticos pré-criados.

O valor `ETag` aqui pode ser um hash do arquivo.

```
# resposta para bundle.v123.js
Last-Modified: Tue, 22 Feb 2022 20:20:20 GMT
ETag: YsAIAAAA-QG4G6kCMAMBAAAAAAoK
```



Além disso, `immutable` pode ser adicionado para evitar a validação no recarregamento.

O resultado combinado é mostrado abaixo.

```
# bundle.v123.js
HTTP/1.1 200 OK
Content-Type: application/javascript
Content-Length: 1024
Cache-Control: public, max-age=31536000, immutable
Last-Modified: Tue, 22 Feb 2022 20:20:20 GMT
ETag: YsAIAAAA-QG4G6kCMAMBAAAAAAoK
```



O **bloqueio de cache** é uma técnica para tornar uma resposta armazenável em cache por um longo período, alterando o URL quando o conteúdo é alterado. A técnica pode ser

aplicada a todos os sub-recursos, como imagens.

**i Nota:** Ao avaliar o uso de `immutable` e QPACK: Se você está preocupado que `immutable` altera o valor predefinido fornecido pelo QPACK, considere que neste caso, a parte `immutable` pode ser codificada separadamente dividindo o valor `Cache-Control` em duas linhas — embora isso dependa do algoritmo de codificação que uma implementação QPACK específica usa.

```
Cache-Control: public, max-age=31536000
```

```
Cache-Control: immutable
```



## Principais recursos

Ao contrário dos sub-recursos, os recursos principais não podem ser bloqueados no cache porque suas URLs não podem ser decoradas da mesma forma que as URLs de sub-recursos.

Se o próprio HTML a seguir for armazenado, a versão mais recente não poderá ser exibida, mesmo que o conteúdo seja atualizado no lado do servidor.

```
<script src="bundle.v123.js"></script>
<link rel="stylesheet" href="build.v123.css" />
<body>
  hello
</body>
```



Para esse caso, `no-cache` seria apropriado — em vez de `no-store` — já que não queremos armazenar HTML, mas apenas queremos que ele esteja sempre atualizado.

Além disso, adicionar `Last-Modified` e `ETag` permitirá que os clientes enviem solicitações condicionais, e um `304 Not Modified` pode ser retornado se não houver atualizações no HTML:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Cache-Control: no-cache
```



```
Last-Modified: Tue, 22 Feb 2022 20:20:20 GMT
```

```
ETag: AAPuIbA0dvAGEETbgAAAAAABAAE
```

Essa configuração é apropriada para HTML não personalizado, mas para uma resposta personalizada usando cookies — por exemplo, após um login — não se esqueça de especificar também `private`:

```
HTTP/1.1 200 OK
```

```
Content-Type: text/html
```

```
Content-Length: 1024
```

```
Cache-Control: no-cache, private
```

```
Last-Modified: Tue, 22 Feb 2022 20:20:20 GMT
```

```
ETag: AAPuIbA0dvAGEETbgAAAAAABAAE
```

```
Set-Cookie: __Host-SID=AHNtAyt3fvJrUL5g5tnGwER; Secure; Path=/; HttpOnly
```



O mesmo pode ser usado para `favicon.ico`, `manifest.json`, `.well-known` e endpoints de API cujos URLs não podem ser alterados usando cache busting.

A maior parte do conteúdo da web pode ser coberta por uma combinação dos dois padrões descritos acima.

## Mais sobre caches gerenciados

Com o método descrito nas seções anteriores, os sub-recursos podem ser armazenados em cache por um longo tempo usando o cache busting, mas os recursos principais (que geralmente são documentos HTML) não podem.

O armazenamento em cache dos recursos principais é difícil porque, usando apenas diretivas padrão da especificação HTTP Caching, não há como excluir ativamente o conteúdo do cache quando o conteúdo é atualizado no servidor.

No entanto, é possível implementar um cache gerenciado, como um CDN ou um service worker.

Por exemplo, uma CDN que permite a limpeza de cache por meio de uma API ou operação de painel permitiria uma estratégia de armazenamento em cache mais agressiva





armazenando o recurso principal e limpando explicitamente o cache relevante somente quando ocorrer uma atualização no servidor.

Um service worker poderia fazer o mesmo se pudesse excluir o conteúdo da API de Cache quando ocorrer uma atualização no servidor.

Para obter mais informações, consulte a documentação do seu CDN e consulte a [documentação do service worker](#).

## Veja também

- [RFC 9111: Protocolo de transferência de hipertexto \(HTTP/1.1\): armazenamento em cache](#) 
- [Tutorial de armazenamento em cache - Mark Nottingham](#) 

**Last modified:** 22 de nov. de 2022, [by MDN contributors](#)