Engineer

Caching

⏭ å ta vare på et svar så arbeidet ikke gjøres på nytt

Hva det er

Caching er praksisen med å ta vare på resultatet av et stykke arbeid, som et HTTP-svar, svaret på en spørring eller en ferdig gjengitt side, og gi det lagrede resultatet til senere forespørsler etter regler som sier hvor lenge det kan stoles på, og hva som pensjonerer det. Verbet er å cache, og et resultat som tas vare på slik, er cachet. Når et program cacher returverdiene til én funksjon inne i seg selv, kalles det som regel memoisering.

Lageret som holder kopiene, er cachen. Caching er politikken rundt den, og den politikken har tre spørsmål. Hvor lenge er en kopi fersk, altså god nok til å serveres uten at noen blir spurt? Når den er blitt foreldet, hvordan sjekkes den? Og når originalen endres, hva fjerner eller erstatter kopien?

HTTP svarer på alle tre i headere. RFC 9111 kaller et lagret svar ferskt så lenge alderen er lavere enn ferskhetslevetiden, som opphavsserveren setter med direktivet `max-age` i `Cache-Control` eller med en dato i `Expires`. Etter det er svaret foreldet, og en cache må normalt validere det før det brukes igjen. Den sender en betinget forespørsel med en validator den har tatt vare på, som regel svarets `ETag` i en `If-None-Match` -header. Har ingenting endret seg, svarer opphavsserveren 304 Not Modified uten innhold. Valideringen koster fortsatt en tur fram og tilbake, men den sparer overføringen og arbeidet med å bygge innholdet på nytt.

```
HTTP/1.1 200 OK
Cache-Control: max-age=3600, stale-while-revalidate=60
ETag: "v42"
```

Inne i en applikasjon får de samme spørsmålene andre former. Med *cache-aside* leser koden cachen først, og ved bom leser den databasen og fyller cachen selv. Med *write-through* oppdaterer hver skriving cachen sammen med databasen, så en lesing finner den gjeldende verdien. En utløpstid på hver oppføring, levetiden eller TTL (*time to live*), spiller rollen til `max-age`. Selv en database kan cache: en materialisert visning i PostgreSQL lagrer resultatet av en spørring som en tabell, og det resultatet endres bare når noen kjører `REFRESH MATERIALIZED VIEW`.

Bruk begrepet i en beslutning

Bruk «Hva er caching?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Finne spørringene og de manglende indeksene bak et system som har blitt tyngre for hvert år, og rette dem som betyr noe.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. 30 dagers strømdata på 180 ms — 180 ms for å aggregere en måned med måleravlesninger

30 dagers strømdata på 180 ms Ta kontakt →

Hvorfor det er viktig

Hver regel for caching bytter ferskhet mot fart, og det riktige byttet varierer med hva slags data det gjelder. En logo kan være et år gammel. En produktpris kan være et minutt gammel. En saldo eller et antall ledige seter bør bekreftes mot kilden før noen handler på det. Caching går oftest galt når én regel brukes på alle tre.

Statiske filer får det reneste svaret. Når en bygging skriver en hash av innholdet i hver fil inn i filnavnet, får en endret fil en ny URL, og den gamle URL-en kan caches i et år. RFC 8246 legger til direktivet `immutable`, som forteller nettleseren at svaret forblir likt så lenge det er ferskt, slik at selv en ny innlasting kan hoppe over revalideringen. HTML-siden som navngir de filene, må derimot ha kort levetid eller valideres ved hver bruk. Ellers fortsetter besøkende å laste de gamle navnene.

To direktiver er lette å forveksle. `no-cache` lar en cache lagre svaret, men forbyr å bruke det igjen uten først å validere det hos opphavsserveren. `no-store` forbyr å lagre det i det hele tatt. For en side som endres ofte, gir `no-cache` sammen med en `ETag` oppdatert innhold og sparer likevel overføringen når ingenting er endret.

Invalidering er den vanskelige halvdelen. RFC 9111 krever at en cache invaliderer en URL når en usikker forespørsel mot den, som `POST`, `PUT` eller `DELETE`, lykkes, og samme avsnitt påpeker at dette bare når cachene forespørselen går gjennom. En nettleser, et CDN og en applikasjonscache er tre adskilte steder, og en kopi som fjernes fra ett av dem, blir liggende i de to andre. Derfor lener de fleste systemer seg på utløpstid som sikkerhetsnett og ser på eksplisitt invalidering som en måte å bli oppdatert raskere på.

Foreldede data kan også være nyttige. RFC 5861 definerer `stale-while-revalidate`, som lar en cache servere et foreldet svar i et oppgitt antall sekunder mens den henter et ferskt i bakgrunnen, og `stale-if-error`, som lar den fortsette å servere en foreldet kopi når opphavsserveren svarer med en feil. Begge gjør en hard utløpstid myk: den besøkende får et svar med en gang, og oppdateringen kommer etter.

Slik ser det ut i praksis

I to prosjekter ble reglene for caching satt i hver sin ende av et system.

Det ene er en plattform der mange av brukerne er til sjøs: sjøfolk og feltpersonell, på mobilen og ofte frakoblet. Der gjorde caching webappen installerbar og brukbar til sjøs. Service workeren settes opp gjennom next-pwa, og cachingen under kjøring i Workbox serverer fonter, bilder, lyd, video og CDN-ressurser etter prinsippet cache først: fra enheten når en kopi finnes der, fra nettet bare ved bom. Det er filene som er like fra én forespørsel til den neste, og siden kobler dem med HTTP-headere for cache-control. Fordi en service worker krever HTTPS, ble installasjon og frakoblet bruk testet over HTTPS på ekte enheter. Resultatet siden rapporterer, er en app som kan installeres, fortsetter å virke uten nett og laster gjentatte besøk fra cachen i stedet for over nettet. En cache først-regel hopper helt over ferskhetssjekken så lenge oppføringen finnes, og derfor passer den filer der innholdet er knyttet til adressen.

Det andre er en PostgreSQL-database som ble ti ganger raskere, med brukere, leietakere, geografiske data og daglige målinger av strømforbruk. Caching var ett tiltak blant flere. Ved siden av målrettede indekser, GiST for romlige spørringer og sammensatte indekser for filtre på leietaker og dato, og tidsbasert partisjonering av tabellen med strømdata, ble noen spørringsresultater holdt som materialiserte visninger, og caching av spørringer og tilkoblingspooling ble satt opp for å kutte overhead. Siden rapporterer svartider som ble 10 ganger

bedre, kritiske operasjoner på millisekunder og ressursbruk ned 40 %. Den gir arbeidet som helhet æren for dette, så hvor stor del cachingen sto for, er ikke målt for seg.

Hvor det går galt

Etter en produksjonssetting får noen besøkende fortsatt den gamle versjonen. Svaret hadde ikke noen eksplisitt ferskhet, så cachene anslo en selv. RFC 9111 lar dem bruke en brøkdel av tiden siden `Last-Modified` og nevner 10 % som en typisk innstilling, så en fil som sist ble endret for en måned siden, kan brukes igjen i rundt tre dager uten at noen blir spurt. Send en eksplisitt `Cache-Control` med hvert svar, legg en hash av innholdet i navnet på statiske filer, og gi HTML-en som peker på dem, kort levetid.

En tømming rydder CDN-et, og nettleserne beholder den gamle kopien. En tømming (*purge*) fjerner CDN-ets kopi og ingenting annet. Hver nettleser som lagret svaret, beholder det til dens egen levetid er ute. Gi offentlige svar en kort `max-age` for nettlesere og en lengre `s-maxage`, som bare delte cacher bruker, så kopien som kan tømmes, er den som lever lengst. Når kopien i nettleseren må endres straks, endrer du URL-en.

En side som var borte en kort stund, blir værende borte. RFC 9110 gjør en 404 heuristisk cachebar, så en cache kan beholde en som kom uten informasjon om ferskhet, i en levetid den anslår selv. En lenke som ble fulgt et øyeblikk før siden ble publisert, kan fortsette å gi 404 etter at siden finnes. Gi feilsvar en eksplisitt kort levetid, og tøm URL-en når siden går live.

En gammel verdi skrives tilbake etter en oppdatering. Med cache-aside bommer en leser og leser raden. Før den lagrer verdien, oppdaterer en skriver raden og sletter oppføringen. Så lagrer leseren det den leste, og den gamle verdien blir liggende til noe fjerner den. Slett oppføringen etter at transaksjonen er committet, og gi hver oppføring en TTL, så en invalidering som tapte et kappløp eller ble glemt, ender av seg selv. I Redis forkaster en vanlig `SET` en tidligere utløpstid, så sett TTL-en i samme kommando:

```
SET price:42 "19.90" EX 300
```

Les mer

- RFC 9111, Section 3: Storing Responses in Caches — når et svar kan lagres, fulgt av reglene for ferskhet, revalidering og invalidering.
- RFC 9110, Section 13: Conditional Requests — `ETag`, `If-None-Match` og 304-svaret som gjør revalidering billig.
- RFC 5861: HTTP Cache-Control Extensions for Stale Content — de to direktivene som lar en cache servere foreldede svar med vilje.
- RFC 8246: HTTP Immutable Responses — direktivet `immutable` for filer der URL-en endres med innholdet.
- Workbox: caching strategies — cache først, nett først og stale-while-revalidate i en service worker, med kode.
- PostgreSQL: Materialized Views — hvordan du holder et spørringsresultat inne i databasen og oppdaterer det ved behov.

Ordliste

Velg ut fra deres faktiske rammer

Ta med et representativt datasett, spørsmålene det skal besvare og dagens flaskehals. Vi kan avtale målinger av kvalitet, gjennomstrømning og gjenoppretting.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er caching?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

welcome@engineer.company +45 50 10 61 84 København, Danmark

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/caching/