Engineer

Caching

⏭ at beholde et svar, så arbejdet ikke gentages

Hvad det er

Caching er praksissen med at gemme resultatet af et stykke arbejde, for eksempel et HTTP-svar, et forespørgselsresultat eller en færdig side, og give det gemte resultat til senere forespørgsler efter regler, der siger, hvor længe man kan stole på det, og hvad der får det til at udgå. Verbet er at cache, og et resultat, der gemmes på den måde, er cachelagret; caching af en enkelt funktions returværdier inde i et program kaldes som regel memoisering.

Selve lageret, der holder kopierne, er cachen. Caching er politikken omkring det, og politikken har tre spørgsmål. Hvor længe er en kopi frisk, altså god nok til at serveres uden at spørge nogen? Når den er forældet, hvordan bliver den så tjekket? Og når originalen ændrer sig, hvad fjerner eller erstatter så kopien?

HTTP besvarer alle tre i headere. RFC 9111 kalder et gemt svar friskt, så længe dets alder er under dets friskhedslevetid, som kilden sætter med `max-age`-direktivet i `Cache-Control` eller med en `Expires`-dato. Derefter er svaret forældet, og en cache skal normalt validere det, før det genbruges. Den sender en betinget forespørgsel med en validator, den har gemt, som regel svarets `ETag` i en `If-None-Match`-header, og er intet ændret, svarer kilden 304 Not Modified uden krop. Validering koster stadig en tur frem og tilbage, men sparer overførslen og arbejdet med at bygge kroppen igen.

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

Inde i en applikation tager de samme spørgsmål andre former. Ved cache-aside læser koden først cachen, og ved en miss læser den databasen og fylder selv cachen. Ved write-through opdaterer hver skrivning cachen sammen med databasen, så en læsning finder den aktuelle værdi. En udløbstid på hver post, dens time to live eller TTL, spiller rollen som `max-age`. Selv en database kan cache: et materialiseret view i PostgreSQL gemmer en forespørgsels resultat som en tabel, og resultatet ændrer sig kun, når nogen kører `REFRESH MATERIALIZED VIEW`.

Brug begrebet i en beslutning

Brug »Hvad er caching?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. At finde de forespørgsler og manglende indeks bag et system, der er blevet tungere hvert år, og rette dem, der betyder noget.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. 30 dages eldata på 180 ms — 180 ms til at aggregere en måneds måleraflæsninger

30 dages eldata på 180 ms Kontakt os →

Hvorfor det betyder noget

Hver regel for caching bytter friskhed for fart, og den rigtige byttehandel afhænger af, hvilken slags data det gælder. Et logo må gerne være et år gammelt. En varepris kan være et minut gammel. En saldo eller et antal ledige pladser bør bekræftes hos kilden, før nogen handler på det. Caching går oftest galt, når én regel bruges på alle tre.

Statiske filer får det reneste svar. Når et build skriver en hash af hver fils indhold ind i navnet, får en ændret fil en ny URL, så den gamle URL kan caches i et år. RFC 8246 tilføjer direktivet `immutable`, som fortæller browseren, at svaret forbliver det samme, mens det er friskt, så selv en genindlæsning kan springe revalideringen over. Den HTML-side, der nævner filerne, skal have kort levetid eller revalideres ved hver brug, ellers bliver de besøgende ved med at hente de gamle navne.

To direktiver er lette at forveksle. `no-cache` lader en cache gemme svaret, men forbyder genbrug uden først at validere det hos kilden; `no-store` forbyder at gemme det overhovedet. For en side, der ændrer sig ofte, giver `no-cache` med en `ETag` aktuelt indhold og sparer stadig overførslen, når intet er ændret.

Invalidering er den svære halvdel. RFC 9111 kræver, at en cache invaliderer en URL, når en usikker forespørgsel til den, som `POST`, `PUT` eller `DELETE`, lykkes, og samme afsnit gør opmærksom på, at det kun når de caches, forespørgslen passerer igennem. En browser, et CDN og en applikationscache er tre forskellige steder, og fjerner man en kopi fra ét af dem, ligger den stadig i de to andre. Derfor læner de fleste systemer sig op ad udløb som sikkerhedsnet og bruger eksplicit invalidering til at blive aktuelle hurtigere.

Forældede data kan også være nyttige. RFC 5861 definerer `stale-while-revalidate`, som lader en cache servere et forældet svar i et angivet antal sekunder, mens den henter et friskt i baggrunden, og `stale-if-error`, som lader den blive ved med at servere en forældet kopi, når kilden svarer med en fejl. Begge gør et hårdt udløb blødt: den besøgende får et svar med det samme, og opdateringen følger efter.

Sådan ser det ud i praksis

En stor del af brugerne på én platform er til søs: søfarende og medarbejdere i marken, på telefonen og ofte offline. Dér gjorde caching webappen installerbar og brugbar til søs. Service workeren er konfigureret gennem next-pwa, og Workbox' runtime-caching serverer skrifttyper, billeder, lyd, video og CDN-ressourcer efter cache first: fra enheden, når der ligger en kopi, og kun fra netværket ved en miss. Det er de filer, der er ens fra den ene forespørgsel til den næste, og siden supplerer dem med cache-control-headere i HTTP. Fordi service workers kræver HTTPS, blev installation og offlinebrug testet over HTTPS på rigtige enheder. Resultatet, siden beskriver, er en app, der kan installeres, virker offline og henter gentagne besøg fra cachen frem for netværket. En cache first-regel springer friskhedstjekket helt over, så længe en post findes, og derfor passer den til filer, hvis indhold er bundet til deres adresse.

Caching hører også hjemme i den anden ende af et system, ved databasen. En PostgreSQL-database, der blev 10 gange hurtigere, rummede brugere, tenants, geografiske data og daglige målinger af elforbrug, og dér var caching ét tiltag blandt flere. Ved siden af målrettede indeks, GiST til spatiale forespørgsler og sammensatte til filtre på tenant og dato, og tidsbaseret partitionering af eltabellen blev nogle forespørgselsresultater holdt som materialiserede views, og caching af forespørgsler og connection pooling blev sat op for at skære overhead ned. Siden melder om 10 gange kortere svartider, kritiske operationer på millisekunder og et ressourceforbrug, der faldt med 40 %. Den tilskriver det arbejdet som helhed, så cachingens egen andel er ikke målt for sig.

Hvor det går galt

Efter en deploy får nogle besøgende stadig den gamle version. Svaret havde ingen eksplicit friskhed, så cachene skønnede selv. RFC 9111 lader dem bruge en brøkdel af tiden siden `Last-Modified` og nævner 10 % som en typisk indstilling, så en fil, der sidst blev ændret for en måned siden, kan genbruges i omkring tre dage uden at spørge. Send en eksplicit `Cache-Control` med hvert svar, læg en indholdshash i navnene på statiske filer, og hold den HTML, der peger på dem, kortlivet.

En purge rydder CDN'et, og browserne beholder den gamle kopi. En purge fjerner CDN'ets kopi og intet andet; hver browser, der har gemt svaret, beholder det, til dens egen levetid udløber. Giv offentlige svar en kort `max-age` til browsere og en længere `s-maxage`, som kun delte caches bruger, så den kopi, der kan purges, er den, der lever længst. Når en browserkopi skal skifte med det samme, så skift URL'en.

En side, der kortvarigt manglede, bliver ved med at mangle. RFC 9110 gør et 404-svar heuristisk cachebart, så en cache må beholde et, der kom uden friskhedsoplysninger, i en levetid, den selv skønner. Et link, der blev fulgt et øjeblik før siden blev udgivet, kan blive ved med at give 404, efter at siden findes. Giv fejlsvar en eksplicit kort levetid, og purge URL'en, når siden går i luften.

En gammel værdi skrives tilbage efter en opdatering. Ved cache-aside misser en læser og læser rækken; før den gemmer værdien, opdaterer en skriver rækken og sletter posten; så gemmer læseren det, den læste, og den gamle værdi bliver liggende, til noget fjerner den. Slet posten, efter at transaktionen er committet, og giv hver post en TTL, så en invalidering, der kom for sent eller udeblev, ender af sig selv. I Redis kasserer en almindelig `SET` enhver tidligere udløbstid, så sæt TTL'en i samme kommando:

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

Læs mere

- RFC 9111, Section 3: Storing Responses in Caches — hvornår et svar må gemmes, efterfulgt af reglerne for friskhed, validering og invalidering.
- RFC 9110, Section 13: Conditional Requests — `ETag`, `If-None-Match` og 304-svaret, der gør revalidering billig.

- RFC 5861: HTTP Cache-Control Extensions for Stale Content — de to direktiver, der lader en cache servere forældede svar med vilje.
- RFC 8246: HTTP Immutable Responses — direktivet `immutable` til filer, hvis URL skifter med indholdet.
- Workbox: caching strategies — cache first, network first og stale-while-revalidate i en service worker, med kode.
- PostgreSQL: Materialized Views — at holde et forespørgselsresultat inde i databasen og genopfriske det efter behov.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring et repræsentativt datasæt, de spørgsmål det skal besvare, og den aktuelle flaskehals. Vi kan aftale målinger af kvalitet, gennemløb og gendannelse.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er caching?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

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

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/caching/