Engineer

Lazy loading

å laste noe først når det faktisk trengs

Hva det er

Lazy loading betyr å vente med å hente noe til det faktisk skal brukes, i stedet for å laste alt med en gang. På en nettside handler det som regel om bilder og innebygde rammer lenger ned, som først hentes når leseren blar seg ned mot dem. Den samme tanken gjelder for kode som bare skal kjøre etter et klikk, og for de relaterte postene et program leser fra en database.

HTML-standarden bygger det inn i selve HTML-koden. Bilder og iframes har et `loading`-attributt med to verdier. `eager`, som er standardverdien, henter ressursen med en gang. `lazy` brukes ifølge standarden «til å utsette hentingen av en ressurs til visse betingelser er oppfylt». Standarden definerer det samme attributtet også på `audio` og `video`. En manglende eller feilstavet verdi regnes som `eager`, så en skrivefeil gjør bare at siden lastes slik den alltid har gjort.

Betingelsen er at elementet nærmer seg visningsområdet. Hvert dokument har sin egen intersection observer til akkurat dette, og når den melder at et element med `lazy` er i ferd med å komme til syne, gjenopptar nettleseren lastingen. Observatøren ser litt forbi det synlige området, slik at et bilde kan komme fram mens leseren fortsatt er på vei ned mot det, og standarden overlater til nettleseren å avgjøre hvor bred den marginen skal være.

Sider kan også bruke mekanismen direkte. W3Cs spesifikasjon Intersection Observer lar et skript få beskjed, asynkront, når et element begynner eller slutter å overlappe visningsområdet eller et annet element, og den nevner egenbygd utsatt lasting av innhold og data blant formålene den er skrevet for. Den dekker det attributtet ikke dekker: et bakgrunnsbilde satt i CSS, et kart som trenger sitt eget skript, en liste som henter neste side når leseren kommer til bunnen.

For kode er enheten en modul. Next.js-dokumentasjonen beskriver lazy loading som en måte å kutte ned på JavaScript-en som trengs for å vise en rute: klientkomponenter og biblioteker holdes utenfor den første bunten og hentes når det blir behov for dem, og eksemplet er en dialogboks som først lastes når noen klikker for å åpne den. `next/dynamic` kombinerer Reacts `lazy()` og `Suspense`, og et vanlig `import()`-kall gjør det samme for et bibliotek.

Begrepet er eldre enn alt dette. I Martin Fowlers katalog over mønstre for forretningsapplikasjoner legger en Lazy Load igjen en markør der de relaterte dataene skulle ha stått, og henter dataene først hvis noe bruker dem. Da drar ikke lesingen av ett objekt fra en database med seg et stort nett av andre objekter som ingen ser på. En ORM arbeider slik når den henter en relatert post første gang koden rører den. Django gjør det som standard: et felt som ikke var med i den opprinnelige spørringen, leses fra databasen første gang det brukes, og dokumentasjonen beskriver dette uten å bruke navnet.

Lazy loading er ikke det samme som caching, som tar vare på en kopi av noe som allerede er hentet, slik at det kan brukes igjen. Det er det motsatte av preload: en `<link rel="preload">` ber nettleseren hente en ressurs på forhånd, fordi siden med stor sannsynlighet kommer til å trenge den.

Bruk begrepet i en beslutning

Bruk «Hva er lazy loading?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Grensesnitt som laster raskt, fungerer på mobilen og oppfører seg likt i alle nettlesere kundene dine måtte bruke.

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. Styrte fullstack-utviklingen av GIS-kart — fullstack-kartutvikling, fra database til nettleser

Styrte fullstack-utviklingen av GIS-kart Ta kontakt →

Hvorfor det er viktig

En side betaler for hver byte den ber om, enten noen ser den eller ikke. En lang artikkel med førti fotografier, en bunntekst med et innebygd kart og en produktside med en video helt nederst bruker alle båndbredde, batteri og tilkoblingstid på innhold som noen lesere aldri blar ned til. Med lazy loading sendes disse forespørslene bare for leserne som kommer så langt, og det den første skjermen trenger, slipper å dele forbindelsen med dem.

Hvor tidlig det skal hentes, er overlatt til nettleseren. Standarden foreslår at marginen tar hensyn til hvor fort leseren blar, og hvor godt nettet er, med en bredere margin på trege forbindelser og forbindelser med høy forsinkelse, slik at bildene er på plass når de kommer til syne. Den krever også at marginen ikke lekker opplysninger som kan brukes til å kjenne igjen leseren gjennom et fingeravtrykk av nettleseren.

Det er også en personvernregel bygd inn. Nettlesere bruker bare lazy loading når skripting er slått på. Ellers, forklarer standarden, kunne et nettsted strødd bilder utover HTML-koden sin og regnet seg fram til omtrent hvor langt hver leser har bladd, ut fra hvilke bilder serveren blir bedt om, og når. Med skripting slått av blir `loading="lazy"` i det stille en vanlig lasting.

For kode beskriver Next.js gevinsten som en lettere første innlasting: mindre JavaScript som trengs før en rute kan vises. Et diagrambibliotek som bare brukes på én fane, eller en redigerer for formatert tekst bak en Rediger-knapp, trenger ikke gjøre alle de andre sidene tregere.

I en ORM ligger gevinsten i det som aldri blir lest. Fowlers poeng er at det er praktisk å laste et objekt sammen med alt som hører til det, helt til det betyr at et enormt antall objekter lastes når det bare var bruk for noen få, og Lazy Load bryter den kjeden.

Slik ser det ut i praksis

For bilder og rammer gjør attributtet jobben. Bildet og kartet nedenfor ligger langt nede på en side, så begge kan vente:

```
<img
  src="/photos/harbour-800.jpg"
  srcset="/photos/harbour-800.jpg 800w, /photos/harbour-1600.jpg 1600w"
  sizes="auto, 100vw"
  width="1600"
  height="900"
  alt="Fiskebåter i havna ved daggry"
  loading="lazy"
/>

<iframe
  src="/embeds/route-map.html"
  title="Kart over turruten"
  width="640"
```

```
  height="360"
  loading="lazy"
></iframe>
```

`width` og `height` gir bildet proporsjonene sine før noe av det har kommet fram: HTMLs regler for gjengivelse gjør om de to tallene til et sideforhold, slik at nettleseren kan holde av en boks med riktig form. `sizes="auto"` er bare tillatt på et bilde med `lazy`. Verdien ber nettleseren velge fra `srcset` etter bredden bildet faktisk får i oppsettet, og den bredden kjenner nettleseren når et lazy-bilde skal hentes. `100vw` etterpå er reserven når bredden ikke lar seg avgjøre.

Noe lazy loading krever et skript. Her hentes et kartbibliotek først når beholderen er i ferd med å komme inn på skjermen:

```
const box = document.querySelector("#route-map");

const observer = new IntersectionObserver(
  async (entries) => {
    if (!entries.some((entry) => entry.isIntersecting)) return;
    observer.disconnect();
    const { drawMap } = await import("./route-map.js");
    drawMap(box);
  },
  { rootMargin: "200px 0px" },
);

observer.observe(box);
```

`rootMargin` utvider visningsområdet med 200 piksler oppe og nede for denne kontrollen, så modulen begynner å lastes litt før boksen blir synlig. Callbacken kjører asynkront og kan melde om flere elementer på én gang, og den sjekker `isIntersecting` fordi observatøren også melder om elementer som ligger utenfor området. Fordi `disconnect()` kalles først, gjør callbacken jobben sin én gang, uansett hvor ofte leseren blar forbi. Fram til da er `route-map.js` ikke en del av siden i det hele tatt.

Fonter kan deles opp slik at en nettleser bare henter delene en side bruker, helt uten skript. Et notat her på nettstedet om en understrekingsfeil i Safari forteller hvordan nettstedets skrifttype, Ubuntu, den gang ble levert som seks filer, tre vekter for latinsk tekst og tre for kyrillisk, hver med sitt eget tegnområde. Nettleseren bruker de latinske variantene til latinsk tekst og de kyrilliske til

kyrillisk tekst. Et språknavn med kyrilliske tegn kan derfor kreve en kyrillisk variant på en side som ellers er på engelsk eller dansk. Notatet kaller ikke dette lazy loading, men det samme behovsstyrte valget holder en ubrukt fontvariant utenfor sidens fontforespørsler, med mindre den uttrykkelig hentes med preload.

Filene som hver side trenger med en gang, krever den motsatte avgjørelsen. En porteføljeoppføring om å kutte stilarkpakken forklarer hvorfor en font som lå innebygd i stilarket som base64, ble tatt ut. Den var lagt der for å få fram den første visningen raskere, men en separat fil med preload gir den samme fordelen, mens den innebygde kopien la vekten sin på hver eneste side, uansett om siden hadde bruk for den. På det tidspunktet ble tre vekter av fonten hentet med preload i stedet, og oppføringen forteller at det var målinger fra virkelige besøk, ikke ønsket om et komplett sett, som satte den tredje på listen. Lazy loading og preload er to sider av samme vurdering: det den første skjermen trenger, kommer tidlig, og resten venter på tur.

Hvor det går galt

Det første bildet kommer sist. Et bilde med `lazy` venter på mer enn nettet. Hentingen gjenopptas først når nettleserens observatør melder at det er nær visningsområdet, og observatører oppdateres sent i hver gjengivelsesrunde, etter at stiler og oppsett er beregnet. For et fotografi langt nede på siden koster ventetiden ingenting. For det store bildet øverst forsinker den nettopp det bildet alle besøkende ser. Utelat `loading`, eller sett det til `eager`, for alt som kan dukke opp på den første skjermen på en telefon eller en datamaskin, og spar `lazy` til det som kommer lenger ned.

Siden hopper når bildene lander. Et bilde uten `width` og `height` har ingen kjent form før filen har kommet, og skyver så alt under seg nedover. Med lazy loading skjer det mens leseren blar, ofte midt i en setning. Gi alle bilder målene sine i HTML-koden eller i CSS. Om `sizes="auto"` er HTML-standarden helt tydelig: uten oppgitte mål blir et slikt bilde sannsynligvis tegnet i 300 ganger 150 piksler, og standarden oppfordrer derfor sterkt til å oppgi dem.

Bilder som trenger et skript for å finnes. En hjemmelaget løsning legger den ekte adressen i et `data-src`-attributt og lar et skript kopiere den over i `src` når bildet nærmer seg. Hvis skriptet ikke lastes, kaster en feil eller blir blokkert, dukker bildet aldri opp, og alt som leser HTML-koden uten å kjøre skript, ser et bilde uten kilde. Det innebygde attributtet svikter motsatt vei: med skripting

slått av lar standarden nettleseren hente bildet med en gang. Bruk `loading="lazy"` for bilder og rammer, og spar skript til det attributtet ikke dekker.

Å utsette det hvert besøk trenger. Lazy loading lønner seg når mye av det som utsettes, aldri blir brukt. Når alt blir brukt, deler det bare én henting opp i mange. En løkke som leser en relatert post for hvert element i en liste, sender en egen spørring for hvert element: Djangos dokumentasjon teller én spørring for listen pluss én per element i standardmodusen og kaller det N+1-problemet. En side der hver komponent ber om sine egne data etter hvert som den dukker opp, gjør det samme over nettet. Hent det som helt sikkert trengs, sammen med listen, i én spørring mot databasen eller ett kall over nettet, og spar lazy loading til det som trolig blir hoppet over.

Les mer

- HTML Standard: Lazy loading attributes — nøkkelordene `lazy` og `eager`, nettleserens egen intersection observer og regelen som slår av lazy loading uten skripting.
- Intersection Observer — W3Cs arbeidsutkast bak observatøren: røtter, marginer, terskler og hvordan callbacker settes i kø.
- How to lazy load Client Components and libraries — `next/dynamic`, `React.lazy()` med `Suspense`, å hoppe over gjengivelse på serveren og å hente et bibliotek med `import()`.
- Lazy Load — Martin Fowlers mønster fra Patterns of Enterprise Application Architecture med de fire variantene.
- Fetch modes — hvordan Django henter felt som ikke ble lastet med den opprinnelige spørringen, og modusene som samler eller blokkerer disse hentingene.

Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er lazy loading?» 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/lazy-loading/