Engineer

Lazy loading

att läsa in något först när det behövs

Vad det är

Lazy loading betyder att man väntar med att hämta något tills det faktiskt behövs, i stället för att ladda allt från början. På en webbsida gäller det oftast bilder och inbäddade ramar längre ned, som hämtas först när läsaren skrollar mot dem. Samma idé gäller kod som bara körs efter ett klick, och de relaterade poster som ett program läser från en databas.

HTML-standarden bygger in det i själva HTML-koden. Bilder och iframes har ett `loading`-attribut med två värden. `eager`, som är standardvärdet, hämtar resursen direkt. Om `lazy` säger standarden att nyckelordet ”används för att skjuta upp hämtningen av en resurs tills vissa villkor är uppfyllda”. Standarden definierar samma attribut även för `audio` och `video`. Ett värde som saknas eller är felstavat räknas som `eager`, så ett stavfel gör bara att sidan laddas som den alltid har gjort.

Villkoret är att elementet närmar sig viewporten. Varje dokument har en egen intersection observer för just det här, och när den rapporterar att ett element med `lazy` är på väg in i bild, återupptar webbläsaren laddningen. Observatören tittar en bit bortom det synliga området, så att en bild kan hinna fram medan läsaren fortfarande skrollar mot den, och standarden låter webbläsaren själv avgöra hur bred den marginalen ska vara.

Sidor kan också använda mekanismen direkt. W3C:s specifikation Intersection Observer låter ett skript be om att få veta, asynkront, när ett element börjar eller slutar överlappa viewporten eller ett annat element, och den nämner egen uppskjuten laddning av innehåll och data bland de användningsområden den skrevs för. Den täcker det som attributet inte täcker: en bakgrundsbild satt i CSS, en karta som behöver ett eget skript, en lista som hämtar nästa sida när läsaren når slutet.

För kod är enheten en modul. Next.js-dokumentationen beskriver lazy loading som ett sätt att minska den JavaScript som behövs för att rendera en route: klientkomponenter och bibliotek hålls utanför det första paketet och hämtas när de behövs, och exemplet är en dialogruta som laddas först när någon klickar för att öppna den. `next/dynamic` kombinerar Reacts `lazy()` och `Suspense`, och ett vanligt `import()`-anrop gör samma sak för ett bibliotek.

Begreppet är äldre än allt detta. I Martin Fowlers katalog över mönster för affärsapplikationer lämnar en Lazy Load en markör där de relaterade data skulle ha legat, och hämtar dem bara om något använder dem. Då drar inte läsningen av ett objekt från en databas med sig ett stort nät av andra objekt som ingen tittar på. En ORM fungerar så när den hämtar en relaterad post första gången koden rör vid den. Django gör det som standard: ett fält som inte fanns med i den ursprungliga databasfrågan läses från databasen första gången det används, och dokumentationen beskriver detta utan att använda namnet.

Lazy loading är inte cachning, som behåller en kopia av något som redan har hämtats så att det kan användas igen. Det är motsatsen till förladdning: en `<link rel="preload">` ber webbläsaren hämta en resurs i förväg, eftersom sidan med stor sannolikhet kommer att behöva den.

Använd begreppet i ett beslut

Använd ”Vad är lazy loading?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Gränssnitt som laddar snabbt, fungerar i mobilen och beter sig likadant i alla webbläsare dina kunder råkar använda.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Styrde full stack-utvecklingen av GIS-kartor — kartutveckling i full stack, från databas till webbläsare

Styrde full stack-utvecklingen av GIS-kartor Hör av er →

Varför det spelar roll

En sida betalar för varje byte den ber om, vare sig någon ser den eller inte. En lång artikel med fyrtio fotografier, en sidfot med en inbäddad karta och en produktsida med en video längst ned lägger alla bandbredd, batteri och uppkopplingstid på innehåll som en del läsare aldrig skrollar fram till. Med lazy loading görs de förfrågningarna bara för de läsare som kommer så långt, och det som den första skärmen behöver slipper dela uppkopplingen med dem.

Hur tidigt som är tillräckligt tidigt överlåts åt webbläsaren. Standarden föreslår att marginalen tar hänsyn till hur snabbt läsaren skrollar och hur bra nätverket är, med en bredare marginal på långsamma uppkopplingar och uppkopplingar med hög latens, så att bilderna är på plats när de kommer in i bild. Den kräver också att marginalen inte läcker information som kan användas för att känna igen läsaren genom ett fingeravtryck av webbläsaren.

Det finns också en integritetsregel inbyggd. Webbläsare använder bara lazy loading när skript är påslagna. Annars, förklarar standarden, skulle en webbplats kunna strö ut bilder i sin HTML-kod och räkna ut ungefär hur långt varje läsare har skrollat utifrån vilka bilder servern blir ombedd att skicka, och när. Med skript avstängda blir `loading="lazy"` i det tysta en vanlig laddning.

För kod beskriver Next.js vinsten som en lättare första laddning: mindre JavaScript som behövs innan en route kan renderas. Ett diagrambibliotek som bara används på en flik, eller en redigerare för formaterad text bakom en Redigera-knapp, behöver inte göra alla andra sidor långsammare.

I en ORM ligger vinsten i det som aldrig läses. Fowlers poäng är att det är bekvämt att ladda ett objekt tillsammans med allt som hör till det, ända tills det innebär att ett enormt antal objekt laddas när bara några få behövs, och Lazy Load bryter den kedjan.

Så ser det ut i praktiken

För bilder och ramar gör attributet jobbet. Bilden och kartan nedan ligger långt ned på en sida, så båda kan vänta:

```
<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="Fiskebåtar i hamnen i gryningen"
  loading="lazy"
/>

<iframe
  src="/embeds/route-map.html"
  title="Karta över vandringsleden"
  width="640"
```

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

`width` och `height` ger bilden dess proportioner innan något av den har kommit fram: HTML:s regler för rendering gör om de två talen till ett bildförhållande, så att webbläsaren kan hålla en ruta med rätt form ledig. `sizes="auto"` är bara tillåtet på en bild med `lazy`. Värdet ber webbläsaren välja ur `srcset` efter den bredd bilden faktiskt får i layouten, en bredd som webbläsaren känner till när en lazy-bild ska hämtas. `100vw` efteråt är reserven när den bredden inte går att avgöra.

Viss lazy loading kräver ett skript. Här hämtas ett kartbibliotek först när behållaren är på väg in på skärmen:

```
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` utökar viewporten med 200 pixlar upptill och nedtill för den här kontrollen, så att modulen börjar laddas en kort stund innan rutan syns. Callbacken körs asynkront och kan rapportera flera element på en gång, och den kontrollerar `isIntersecting` eftersom observatören också rapporterar element som ligger utanför området. Eftersom `disconnect()` anropas först gör callbacken sitt jobb en gång, hur ofta läsaren än skrollar förbi. Fram till dess är `route-map.js` inte en del av sidan alls.

Typsnitt kan delas upp så att en webbläsare bara hämtar de delar en sida använder, helt utan skript. En anteckning här på webbplatsen om en understrykningsbugg i Safari berättar hur webbplatsens typsnitt, Ubuntu, då levererades som sex filer, tre vikter för latinsk text och tre för kyrillisk, där varje

fil anger vilket teckenområde den omfattar. Webbläsaren använder de latinska varianterna för latinsk text och de kyrilliska för kyrillisk text. Ett språknamn med kyrilliska tecken kan därför kräva en kyrillisk variant även på en sida som annars är på engelska eller danska. Anteckningen kallar inte detta lazy loading, men samma behovsstyrda val håller en oanvänd variant borta från sidans typsnittsförfrågningar, såvida den inte uttryckligen förladdas.

Filerna som varje sida behöver direkt kräver det motsatta beslutet. En portföljpost om att banta stilmallspaketet förklarar varför ett typsnitt som låg inbäddat i stilmallen som base64 plockades ut. Det hade lagts där för att få fram den första renderingen snabbare, men en separat fil som laddas med preload ger samma vinst, medan den inbäddade kopian lade sin vikt på varje sida, vare sig sidan hade nytta av den eller inte. I det skedet hämtades tre vikter av typsnittet i förväg i stället, och enligt posten var det mätningar från riktiga besök, inte en känsla av fullständighet, som gav den tredje dess plats på listan. Lazy loading och preload är två sidor av samma bedömning: det den första skärmen behöver kommer tidigt, och resten får vänta på sin tur.

Var det går fel

Den första bilden kommer sist. En bild med `lazy` väntar på mer än nätverket. Hämtningen återupptas först när webbläsarens observatör rapporterar att den är nära viewporten, och observatörer uppdateras sent i varje renderingsrunda, efter att stil och layout har beräknats. För ett fotografi långt ned på sidan kostar väntan ingenting. För den stora bilden högst upp försenar den just den bild som alla besökare ser. Utelämna `loading` , eller sätt det till `eager` , för allt som kan synas på den första skärmen på en telefon eller en dator, och spara `lazy` till det som kommer längre ned.

Sidan hoppar när bilderna landar. En bild utan `width` och `height` har ingen känd form förrän filen har kommit fram, och knuffar sedan ned allt under sig. Med lazy loading händer det medan läsaren skrollar, ofta mitt i en mening. Ge varje bild dess mått i HTML-koden eller i CSS. Om `sizes="auto"` är HTML-standarden tydlig: utan angivna mått ritas en sådan bild troligen upp i 300 gånger 150 pixlar, och standarden uppmuntrar därför starkt till att ange dem.

Bilder som behöver ett skript för att finnas. En hemsnickrad lösning lägger den riktiga adressen i ett `data-src` -attribut och låter ett skript kopiera den till `src` när bilden närmar sig. Om skriptet inte laddas, kastar ett fel eller blockeras, dyker bilden aldrig upp, och allt som läser HTML-koden utan att köra skript ser en bild

utan källa. Det inbyggda attributet slår fel åt andra hållet: med skript avstängda låter standarden webbläsaren hämta bilden direkt. Använd `loading="lazy"` för bilder och ramar, och spara skript till det som attributet inte täcker.

Att skjuta upp det som varje besök behöver. Lazy loading lönar sig när mycket av det som skjuts upp aldrig används. När allt kommer att användas delar det bara upp en hämtning i många. En loop som läser en relaterad post för varje objekt i en lista skickar en separat databasfråga för vart och ett: Djangos dokumentation räknar en databasfråga för listan plus en per objekt i standardläget och kallar det N+1-problemet. En sida där varje komponent ber om sina egna data när den dyker upp gör samma sak över nätverket. Hämta det som säkert behövs tillsammans med listan, i en enda databasfråga eller ett enda anrop över nätverket, och spara lazy loading till det som troligen hoppas över.

Läs mer

- HTML Standard: Lazy loading attributes — nyckelorden `lazy` och `eager`, webbläsarens egen intersection observer och regeln som stänger av lazy loading utan skript.
- Intersection Observer — W3C:s arbetsutkast bakom observatören: rötter, marginaler, tröskelvärden och hur callbacks köas.
- How to lazy load Client Components and libraries — `next/dynamic`, `React.lazy()` med `Suspense`, att hoppa över rendering på servern och att hämta ett bibliotek med `import()`.
- Lazy Load — Martin Fowlers mönster från Patterns of Enterprise Application Architecture med de fyra varianterna.
- Fetch modes — hur Django hämtar fält som inte laddades med den ursprungliga databasfrågan, och lägena som samlar ihop eller blockerar de hämtningarna.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är lazy loading?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/lazy-loading/