Engineer

Lazy Loading

Laden von etwas erst beim ersten Bedarf

Was es ist

Lazy Loading bedeutet, den Abruf von etwas bis zu dem Moment aufzuschieben, in dem es benötigt wird, statt alles vorab zu laden. Auf einer Webseite betrifft das meist Bilder und eingebettete Frames weiter unten, die erst abgerufen werden, wenn der Leser auf sie zuscrollt. Dieselbe Idee gilt für Code, der erst nach einem Klick ausgeführt wird, und für verknüpfte Datensätze, die eine Anwendung aus einer Datenbank liest.

Der HTML-Standard verankert das im Markup. Bilder und Iframes unterstützen ein `loading` -Attribut mit zwei Werten. `eager` , der Standardwert, ruft die Ressource sofort ab. `lazy` wird laut Standard „verwendet, um den Abruf einer Ressource aufzuschieben, bis bestimmte Bedingungen erfüllt sind“. Der Standard definiert dasselbe Attribut auch für `audio` und `video` . Ein fehlender oder falsch geschriebener Wert gilt als `eager` ; bei einem Tippfehler lädt die Seite also genauso wie zuvor.

Die Bedingung ist, dass das Element in die Nähe des sichtbaren Bereichs gelangt. Jedes Dokument besitzt für diese Aufgabe einen eigenen Intersection Observer. Meldet dieser, dass ein verzögert geladenes Element gleich sichtbar wird, setzt der Browser dessen Laden fort. Der Observer blickt etwas über den sichtbaren Bereich hinaus, damit ein Bild eintreffen kann, während der Leser noch darauf zuscrollt. Wie groß dieser Rand ist, überlässt der Standard dem Browser.

Seiten können denselben Mechanismus unmittelbar verwenden. Die W3C-Spezifikation Intersection Observer ermöglicht es einem Skript, sich asynchron benachrichtigen zu lassen, wenn ein Element beginnt oder aufhört, sich mit dem sichtbaren Bereich oder einem anderen Element zu überschneiden. Sie nennt das individuell verzögerte Laden von Inhalten und Daten als eines der Verfahren, die

sie unterstützen soll. Damit deckt sie ab, was das Attribut nicht erfasst: ein per CSS gesetztes Hintergrundbild, eine Karte, die ihr eigenes Skript braucht, oder eine Liste, die ihre nächste Seite abruft, wenn der Leser ihr Ende erreicht.

Bei Code ist die Einheit ein Modul. Die Next.js-Dokumentation stellt Lazy Loading als Möglichkeit vor, die zum Rendern einer Route benötigte JavaScript-Menge zu verringern: Client-Komponenten und Bibliotheken bleiben aus dem ersten Bundle heraus und werden bei Bedarf abgerufen. Ihr Beispiel ist ein Dialog, der erst lädt, wenn jemand ihn per Klick öffnet. `next/dynamic` verbindet Reacts `lazy()` und `Suspense`; ein einfacher `import()`-Aufruf erledigt dasselbe für eine Bibliothek.

Der Begriff ist älter als all diese Verfahren. In Martin Fowlers Katalog von Mustern für Unternehmensanwendungen hinterlässt Lazy Load einen Platzhalter dort, wo verknüpfte Daten stehen würden, und ruft diese erst ab, wenn etwas sie verwendet. So zieht das Lesen eines Objekts aus einer Datenbank kein großes Netz weiterer Objekte nach sich, die niemand betrachtet. Ein ORM arbeitet so, wenn es einen verknüpften Datensatz beim ersten Zugriff des Codes abruft. Django tut dies standardmäßig: Ein Feld, das nicht zur ursprünglichen Abfrage gehörte, wird beim ersten Zugriff aus der Datenbank gelesen. Die Dokumentation beschreibt dies, ohne den Begriff zu verwenden.

Lazy Loading ist kein Caching, das eine Kopie von etwas bereits Abgerufenem zur erneuten Verwendung aufbewahrt. Es ist das Gegenteil von Preloading: Ein `<link rel="preload">` fordert den Browser auf, eine Ressource vorab abzurufen, weil die Seite sie mit hoher Wahrscheinlichkeit benötigen wird.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Lazy Loading?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Oberflächen, die schnell laden, auf dem Smartphone funktionieren und sich in allen Browsern Ihrer Kunden gleich verhalten.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. Full-Stack-GIS-Kartenentwicklung geleitet — Full-Stack-Kartenentwicklung von der Datenbank bis zum Browser

Full-Stack-GIS-Kartenentwicklung geleitet Kontakt aufnehmen →

Warum es wichtig ist

Eine Seite bezahlt für jedes angeforderte Byte, unabhängig davon, ob jemand es sieht. Ein langer Artikel mit vierzig Fotos, ein Seitenfuß mit eingebetteter Karte und eine Produktseite mit einem Video weit unten verbrauchen Bandbreite, Akkuleistung und Verbindungszeit für Inhalte, zu denen manche Leser niemals scrollen. Mit Lazy Loading entstehen diese Anfragen nur für Leser, die so weit kommen. Was der erste Bildschirm braucht, muss die Verbindung nicht mit ihnen teilen.

Wie früh früh genug ist, entscheidet der Browser. Der Standard schlägt vor, beim Rand die Scrollgeschwindigkeit des Lesers und die Qualität der Verbindung zu berücksichtigen. Bei langsamen Verbindungen oder solchen mit hoher Latenz soll der Rand größer ausfallen, damit Bilder bereitstehen, wenn sie ins Blickfeld scrollen. Er verlangt außerdem, dass der Rand keine Informationen preisgibt, mit denen sich ein Fingerabdruck des Lesers erstellen ließe.

Auch eine Datenschutzregel ist eingebaut. Browser laden nur dann verzögert, wenn die Skriptausführung aktiviert ist. Andernfalls könnte eine Website laut Standard Bilder über ihr Markup verteilen und aus den Zeitpunkten und der Auswahl der beim Server angeforderten Bilder ungefähr ableiten, wie weit jeder Leser gescrollt ist. Bei ausgeschalteter Skriptausführung wird `loading="lazy"` stillschweigend zu einem gewöhnlichen Ladevorgang.

Bei Code beschreibt Next.js den Gewinn für das erste Laden: weniger JavaScript, das vor dem Rendern einer Route benötigt wird. Eine Diagrammbibliothek, die nur auf einem Tab verwendet wird, oder ein Rich-Text-Editor hinter einer Schaltfläche „Bearbeiten“ muss nicht jede andere Seite verlangsamen.

Bei einem ORM liegt der Gewinn in dem, was nie gelesen wird. Fowler weist darauf hin, dass es bequem ist, ein Objekt samt allem damit Verknüpften zu laden, bis dadurch riesige Mengen von Objekten geladen werden, obwohl nur wenige benötigt werden. Lazy Load durchbricht diese Kette.

Wie es in der Praxis aussieht

Bei Bildern und Frames erledigt das Attribut die Arbeit. Das folgende Bild und die Karte liegen weit unten auf einer Seite; beide können deshalb warten:

```
<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="Fishing boats in the harbour at dawn"
  loading="lazy"
/>

<iframe
  src="/embeds/route-map.html"
  title="Map of the walking route"
  width="640"
  height="360"
  loading="lazy"
></iframe>
```

`width` und `height` geben dem Bild seine Proportionen, bevor auch nur ein Teil davon eingetroffen ist: Die HTML-Renderingregeln bilden die beiden Zahlen auf ein Seitenverhältnis ab, sodass der Browser einen Bereich der richtigen Form freihalten kann. `sizes="auto"` ist nur bei einem verzögert geladenen Bild erlaubt. Es weist den Browser an, anhand der tatsächlich im Layout verwendeten Bildbreite aus `srcset` auszuwählen. Bei einem solchen Bild steht diese Breite zum Abrufzeitpunkt bereits fest. Das folgende `100vw` ist der Ersatzwert, falls sich die Breite nicht bestimmen lässt.

Manches verzögerte Laden braucht ein Skript. Hier wird eine Kartenbibliothek erst abgerufen, wenn ihr Container gleich ins Blickfeld scrollt:

```
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` erweitert den sichtbaren Bereich für diese Prüfung oben und unten um 200 Pixel. Das Modul beginnt deshalb kurz vor dem Sichtbarwerden des Containers zu laden. Der Callback läuft asynchron und kann mehrere Einträge zugleich melden. Er prüft `isIntersecting`, weil der Observer auch Elemente außerhalb des Bereichs meldet. Durch das vorherige Trennen des Observers erledigt der Callback seine Arbeit genau einmal, unabhängig davon, wie oft der Leser vorbeiscrollt. Bis dahin gehört `route-map.js` überhaupt nicht zur Seite.

Schriften lassen sich so aufteilen, dass ein Browser ohne jedes Skript nur die Teile abruft, die eine Seite verwendet. Eine Notiz auf dieser Website über einen Safari-Fehler bei Unterstreichungen hält fest, wie die Schrift der Website, Ubuntu, damals in sechs Dateien ausgeliefert wurde: drei Schriftschnitte für lateinischen Text und drei für kyrillischen. Jede Datei nennt den Zeichenbereich, den sie abdeckt. Der Browser verwendet für lateinischen Text die lateinischen Schriftschnitte und für kyrillischen die kyrillischen. Eine in kyrillischer Schrift geschriebene Sprachbezeichnung kann deshalb auch auf einer ansonsten englischen oder dänischen Seite einen kyrillischen Schriftschnitt benötigen. Die Notiz bezeichnet das nicht als Lazy Loading, doch dieselbe bedarfsabhängige Auswahl verhindert, dass die Seite einen ungenutzten Schriftschnitt anfordert, sofern er nicht ausdrücklich vorab geladen wird.

Dateien, die jede Seite sofort braucht, verlangen die entgegengesetzte Entscheidung. Ein Portfolioeintrag über das Verkleinern des Stylesheet-Bundles erklärt, warum eine als base64 in das Stylesheet eingebettete Schrift entfernt wurde. Sie war dort eingebaut worden, um die erste Darstellung zu beschleunigen. Eine separate, vorab geladene Datei bringt jedoch denselben Vorteil, während die eingebettete Kopie jede Seite mit ihrem Gewicht belastete, ob sie dort benötigt wurde oder nicht. In diesem Stadium wurden stattdessen drei Schriftschnitte vorab geladen. Laut Eintrag haben Felddaten und nicht der Wunsch nach Vollständigkeit den dritten auf diese Liste gebracht. Lazy Loading und Preloading sind zwei Seiten derselben Abwägung: Was der erste Bildschirm braucht, kommt früh; der Rest wartet, bis er an der Reihe ist.

Wo es schiefgeht

Das erste Bild trifft zuletzt ein. Ein verzögert geladenes Bild wartet auf mehr als nur das Netzwerk. Sein Abruf wird erst fortgesetzt, wenn der Observer des Browsers meldet, dass es nahe am sichtbaren Bereich ist. Observer werden im Rahmen des Renderns eines Frames aktualisiert, nach Stilberechnung und Layout. Bei einem Foto weit unten kostet dieses Warten nichts. Beim großen Bild ganz oben verzögert es das eine Bild, das jeder Besucher sieht. Lassen Sie

`loading` weg oder setzen Sie es auf `eager`, wenn etwas auf dem ersten Bildschirm eines Telefons oder Desktop-Rechners erscheinen kann. Setzen Sie `lazy` nur für den darunterliegenden Inhalt ein.

Die Seite springt, wenn Bilder eintreffen. Ein Bild ohne `width` und `height` hat bis zum Eintreffen seiner Datei keine bekannte Form und schiebt danach alles darunter nach unten. Beim verzögerten Laden geschieht das während des Scrollens, oft mitten in einem Satz. Geben Sie jedem Bild seine Abmessungen im Markup oder per CSS. Für `sizes="auto"` macht der HTML-Standard eine ausdrückliche Angabe: Ohne angegebene Abmessungen wird ein solches Bild wahrscheinlich mit 300 mal 150 Pixeln gezeichnet. Deshalb empfiehlt er nachdrücklich, sie anzugeben.

Bilder, die ein Skript brauchen, um überhaupt zu erscheinen. Eine selbst gebaute Variante legt die eigentliche Adresse in ein `data-src`-Attribut und lässt ein Skript sie nach `src` kopieren, wenn das Bild näher kommt. Lädt dieses Skript nicht, löst es einen Fehler aus oder wird es blockiert, erscheint das Bild nie. Alles, was das Markup ohne Skriptausführung liest, sieht ein Bild ohne Quelle. Das eingebaute Attribut verhält sich bei diesem Ausfall umgekehrt: Bei deaktivierter Skriptausführung lässt der Standard den Browser das Bild sofort laden. Verwenden Sie `loading="lazy"` für Bilder und Frames und beschränken Sie Skripte auf das, was das Attribut nicht abdeckt.

Aufschieben, was jeder Besuch braucht. Lazy Loading lohnt sich, wenn viel vom Aufgeschobenen nie verwendet wird. Wird alles gebraucht, zerlegt es lediglich einen Abruf in viele. Eine Schleife, die für jeden Listeneintrag einen verknüpften Datensatz liest, sendet für jeden eine eigene Abfrage. Die Django-Dokumentation zählt im Standardmodus eine Abfrage für die Liste plus eine je Eintrag und nennt das Ergebnis das N+1-Abfragenproblem. Eine Seite, deren Komponenten beim Erscheinen jeweils ihre eigenen Daten anfordern, tut über das Netzwerk dasselbe. Laden Sie das, was sicher gebraucht wird, gemeinsam mit der Liste in einer Abfrage oder Anfrage und setzen Sie Lazy Loading nur für das ein, was wahrscheinlich übersprungen wird.

Weiterführende Lektüre

- HTML-Standard: Attribute für verzögertes Laden — die Schlüsselwörter `lazy` und `eager`, der eigene Intersection Observer des Browsers und die Regel, die verzögertes Laden ohne Skriptausführung ausschaltet.

- Intersection Observer — der W3C-Arbeitsentwurf hinter dem Observer: Bezugsbereiche, Ränder, Schwellenwerte und das Einreihen von Callbacks.
- Client-Komponenten und Bibliotheken verzögert laden — `next/dynamic`, `React.lazy()` mit `Suspense`, das Überspringen serverseitigen Renderns und das Laden einer Bibliothek mit `import()`.
- Lazy Load — Martin Fowlers Muster aus Patterns of Enterprise Application Architecture mit seinen vier Varianten.
- Abrufmodi — wie Django Felder abruft, die nicht mit der ursprünglichen Abfrage geladen wurden, und welche Modi diese Abrufe bündeln oder blockieren.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist Lazy Loading?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/lazy-loading/