Engineer

JavaScript

sproget, browsere kører

Hvad det er

JavaScript er et dynamisk typet programmeringssprog, standardiseret under navnet ECMAScript, som enhver webbrowser kan afvikle, og som også kører på servere og byggemaskiner gennem afviklingsmiljøer som Node.js. I daglig tale hedder det JS. Standardens årlige udgaver hedder ES2015, ES2024 og så videre, de gamle numre (ES5, ES6) høres stadig, og »vanilla JavaScript« betyder sproget uden et framework ovenpå. Bortset fra navnet og de krøllede parenteser har det meget lidt med Java at gøre.

Brendan Eich skrev den første version hos Netscape, og den kom med i browseren Navigator 2.0. Ecma International vedtog første udgave af standarden, ECMA-262, i juni 1997, og organisationens komité TC39 har udsendt en ny udgave hvert år siden ECMAScript 2016. Et forslag kommer først med i specifikationen på trin 4 i komitéens proces, og det trin kræver implementeringer, der allerede er sendt ud. Derfor er teksten og motorerne sjældent uenige ret længe.

Sproget selv siger meget lidt om verden udenfor. ECMA-262 definerer syntaksen, typerne, objekter, promises og moduler og overlader input og output til det miljø, der afvikler sproget, og som standarden kalder værten. I en browser er værten webplatformen: dokumentet, hændelser, timere og netværkskald, som for det meste er beskrevet i HTML Standard og dens søsterstandarder. I Node.js stiller værten i stedet filer, sockets og processer til rådighed. Den deling forklarer en klassisk overraskelse: kode kan være fuldt gyldig JavaScript og alligevel gå i stykker, når den flytter fra en side til en server, for sproget flyttede med, mens værten blev hjemme.

I browseren deler en sides scripts én hovedtråd med layout, tegning og input. En opgave kører færdig, før browseren kan reagere på det næste klik. Promises og `async` -funktioner holder ventetiden væk fra tråden, men beregninger bliver på den, medmindre de flyttes over i en Web Worker. Tal er IEEE 754-værdier med dobbelt præcision, så hele tal er kun eksakte op til 2^53 − 1, og alt derover kræver `BigInt` .

Brug begrebet i en beslutning

Brug »Hvad er JavaScript?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Brugerflader, der indlæses hurtigt, virker på en telefon og opfører sig ens i alle de browsere, jeres kunder nu bruger.

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. Har styret full-stack-udviklingen af GIS-kort — full stack-kortløsning, fra database til browser

Har styret full-stack-udviklingen af GIS-kort Kontakt os →

Hvorfor det betyder noget

Den egentlige beslutning handler om, hvor meget af det man sender til dem, der læser sitet. JavaScript er det sprog, der får en side til at reagere uden en tur forbi serveren, og hver kilobyte af det skal hentes, parses, kompileres og afvikles på en enhed, man ikke selv råder over. Et bundt, der føles øjeblikkeligt på en udviklers bærbare, kan holde en mellemklassetelefon beskæftiget i flere sekunder, og regningen betaler læseren med batteri og tålmodighed.

Script er også der, hvor en side står mest åben. Hvis en angribers script bliver kørt, sker det med sidens rettigheder: det kan læse, hvad læseren taster, og sende det hvor som helst, siden selv må sende hen. Det er den risiko, Content Security Policy findes for at inddæmme. Ved ikke at levere JavaScript til afvikling i browseren fjerner sitet sine egne browserscripts som angrebsflade; input, man ikke kan stole på, skal stadig håndteres sikkert, og en restriktiv politik er fortsat nødvendig. Pakkeøkosystemet åbner en anden flanke. Et typisk frontendbyg trækker hundredvis af transitive afhængigheder med sig, og hver eneste af dem er kode, der kører med de samme rettigheder som ens egen.

Den anden side af regnestykket er lige så virkelig. Når det samme sprog kører på begge sider af et system, kan browseren og serveren dele en valideringsregel eller en dataform, og nogle produkter har brug for meget script: offlinetilstand, træk og slip, kort, samarbejde i realtid. Det nyttige spørgsmål stilles derfor

funktion for funktion: skal det her ske i læserens browser, uden at en server bliver spurgt? Lyder svaret nej, så lad serveren eller byggeprocessen lave HTML'en, og send den.

Det beslægtede valg står mellem JavaScript og TypeScript. TypeScript tilføjer en typetjekker og et byggetrin og kompilerer tilbage til almindelig JavaScript. I en kodebase, som flere mennesker ændrer hver uge, tjener tjekkeren som regel sig selv hjem. Til en forbedring på hundrede linjer på en statisk side rækker JavaScript og en streng linter.

Sådan ser det ud i praksis

Dette site leverer statiske sider uden JavaScript til afvikling i læserens browser. Søgekoden er bevaret, men slået fra; JavaScript bruges stadig i værktøjerne til byg og kontrol. Tre sider viser den tidligere browserundtagelse og arbejdet uden for browseren.

En tidligere version på tre sprog havde ingen backend ved kørsel og leverede ét førstepartsscript. Beskrivelsen af Hugo-bygget dokumenterer den afgrænsede undtagelse: et porteføljefilter, hvis betjening var skjult, indtil scriptet kørte. De 108 skabelonfiler, herunder 53 partials og 18 shortcodes, gjorde arbejdet, når sitet blev bygget. Beskrivelsen nævner også prisen dengang: interaktive funktioner skulle løses i bygget, og et søgeindeks blev vurderet og lagt til side frem for leveret. Tallene og undtagelsen beskriver den tidligere løsning, ikke den kode, læseren modtager nu.

Det samme indhold udgives over Gemini og Gopher, to protokoller, hvor et script slet ikke har nogen plads. Ét build skrev 777 dokumenter i hvert format i den dokumenterede implementeringsfase fra den samme kilde som siderne. En separat sammenligning af builds bekræftede nul ændrede HTML-bytes, da de outputformater blev tilføjet; resultatet omfatter ikke senere redaktionelle eller visuelle ændringer. Sitets argument er, at et dokument klarer sig uden en megabyte script for at blive læst, og et format, der slet ikke kan rumme et script, er den mest ligefremme måde at vise det på.

Det meste JavaScript her bor i værktøjerne. I den implementeringsfase, som det linkede arbejde dokumenterer, bestod sitets kvalitetsgate af toogtredive scripts og 10.242 linjer JavaScript, dengang den største kodemængde i repositoryet. I en periode var det den eneste kode, som hverken blev formateret, lintet eller typetjekket. Arbejdet med at få den under en formatter og en linter begyndte

med at tilpasse formatteren til koden: repositoryets standard på fire mellemrum ville have omskrevet 2.173 linjer af den største tjekker, og en overstyring til to mellemrum for de filer efterlod 38 linjer med reelle afvigelser. Derefter rapporterede linteren 13 fund, og alle 13 blev rettet uden en eneste undertrykkelse. En fejl i en tjekker viser sig som et tjek, der består, når det burde slå fejl, og derfor får gatens egen kode den behandling, den selv giver alt andet.

Hvor det går galt

Fejlene herunder følger af, hvordan sproget er skruet sammen, og hver af dem har en kendt løsning.

Siden er tegnet, men reagerer først på tryk et sekund senere. En lang opgave, som at hydrere et stort bundt, parse et stort JSON-svar eller sortere tusindvis af rækker, optager den ene tråd, der håndterer input. Send mindre kode til den første interaktion ved at dele bundtet op pr. rute og udskyde resten, del langt arbejde op i bidder, der giver kontrollen tilbage til event-loopet, og flyt tunge beregninger over i en Web Worker. Mål på en langsom telefon frem for på den bærbare, koden blev skrevet på.

Priser og id'er kommer ud en anelse forkerte. Hvert tal er et binært flydende kommatal, så decimalbrøker afrundes, og meget store heltal falder sammen med deres naboer. En 64-bit databasenøgle sendt som et JSON-tal er det typiske offer:

```
0.1 + 0.2 === 0.3; // false
Number.MAX_SAFE_INTEGER; // 9007199254740991
9007199254740993 === 9007199254740992; // true
```

Hold beløb i hele mindsteenheder som øre, send store id'er som strenge, og brug `BigInt`, hvor regnestykket skal være eksakt.

En sammenligning lukker den forkerte værdi igennem. Operatoren `==` konverterer sine operander, før den sammenligner, så `"" == 0` og `"0" == false` er begge sande. Brug `===` overalt, og lad linterens regel `eqeqeq` håndhæve det, så en konvertering kun sker, når koden udtrykkeligt beder om den.

Tekst fra en bruger ender med at køre som kode. At tildele utroværdig tekst til `innerHTML`, at bygge markup ved at sætte strenge sammen eller at give input videre til `eval` gør data til script. Skriv tekst med `textContent`, send en Content Security Policy, der kun lukker scripts ind via nonce eller hash, og hold listen over afhængigheder kort og fastlåst i en lockfile, så hver opdatering bliver læst, før den kommer ind.

Kode, der virker i browseren, går ned på serveren. Sproget flyttede, værten blev: `window` og `document` findes kun i en browser, og Node.js læser en fil som CommonJS eller som ES-modul afhængigt af indstillinger, browseren aldrig ser. Hold værtsspecifikke kald ude i kanterne, sæt `"type": "module"` eksplicit i `package.json`, og kør hver del af koden i det miljø, den faktisk skal køre i.

Læs mere

- ECMAScript Language Specification — det levende udkast til ECMA-262, den tekst alle motorer implementerer, opdateret efterhånden som forslag når trin 4.
- The TC39 Process — de trin, et forslag går igennem, før det bliver en del af sproget, og hvad hvert trin kræver.
- ECMA-262 hos Ecma International — de udgivne årlige udgaver, til når et projekt skal navngive den version, det sigter efter.
- HTML Standard: event loops — hvordan en browser planlægger opgaver, mikroopgaver og rendering omkring en sides scripts.
- HTML Standard: the script element — klassiske scripts og modulscripts, og hvordan `async` og `defer` ændrer, hvornår de kører.

Ordliste

Vælg ud fra jeres faktiske vilkår

Beskriv arbejdsgangen, de eksisterende grænseflader og de brugere, der har brug for ændringen. Vi kan definere en testbar leverance og en overdragelse, der gør teamet i stand til at vedligeholde løsningen.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er JavaScript?« 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/javascript/