Engineer

JavaScript

språket nettlesere kjører

Hva det er

JavaScript er et dynamisk typet programmeringsspråk, standardisert som ECMAScript, som enhver nettleser kan kjøre, og som også kjører på servere og byggemaskiner gjennom kjøremiljøer som Node.js. Mange forkorter det til JS. Standardens årlige utgaver heter ES2015, ES2024 og så videre, den eldre nummereringen (ES5, ES6) høres fortsatt, og «vanilla JavaScript» betyr språket uten noe rammeverk oppå. Bortsett fra navnet og en del syntaks med krøllparenteser har det lite til felles med Java.

Brendan Eich skrev den første versjonen hos Netscape, og den ble levert med nettleseren Navigator 2.0. Ecma International vedtok den første utgaven av standarden, ECMA-262, i juni 1997, og komiteen TC39 har publisert en ny utgave hvert år siden ECMAScript 2016. Et forslag kommer inn i spesifikasjonen først på trinn 4 i komiteens prosess, og det trinnet krever implementasjoner som allerede er levert, så teksten og motorene er sjelden uenige lenge.

Språket sier svært lite om verden utenfor. ECMA-262 definerer syntaksen, typene, objekter, promises og moduler, og overlater inndata og utdata til det miljøet som er vert for språket. I en nettleser er den verten webplattformen: dokumentet, hendelser, tidtakere og nettverksforespørsler, spesifisert for det meste i HTML-standarden og søsterstandardene dens. I Node.js leverer verten i stedet filer, sockets og prosesser. Det skillet forklarer en vanlig overraskelse: kode kan være helt gyldig JavaScript og likevel slutte å virke når den flyttes fra en side til en server, fordi språket ble med og verten ble igjen.

I en nettleser deler skriptene på en side én hovedtråd med layout, tegning og inndata. En oppgave kjører ferdig før nettleseren kan svare på neste klikk. Promises og `async` -funksjoner holder ventingen utenfor den tråden, mens

beregninger blir værende på den med mindre de flyttes til en Web Worker. Tall er flyttall med dobbel presisjon etter IEEE 754, så heltall er eksakte bare opp til 2^53 – 1, og alt større krever `BigInt` .

Bruk begrepet i en beslutning

Bruk «Hva er JavaScript?» 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

Den egentlige avgjørelsen er hvor mye av det du skal sende til dem som leser nettstedet ditt. JavaScript er språket som får en side til å oppføre seg uten en rundtur til en server, og hver kilobyte av det lastes ned, parses, kompileres og kjøres på en enhet du ikke har kontroll over. Pakken som føles øyeblikkelig på en utviklers bærbare maskin, kan holde en mellomklassetelefon opptatt i flere sekunder, og leseren betaler for det med batteri og tålmodighet.

Skript er også der en side er mest utsatt. Hvis en angripers skript blir kjørt, har det sidens rettigheter: det kan lese det leseren skriver, og sende det hvor som helst siden har lov til å sende noe. Det er den risikoen Content Security Policy finnes for å begrense. Ved å la være å levere kjørbar JavaScript fjerner nettstedet sine egne nettleserskript som angrepsflate; inndata man ikke kan stole på, må fortsatt håndteres trygt, og en restriktiv policy er fortsatt nødvendig. Pakkeøkosystemet gir en annen angrepsflate. Et typisk bygg av en frontend trekker inn hundrevis av transitive avhengigheter, og hver av dem er kode som kjører med de samme rettighetene som din egen.

Den andre siden av regnskapet er reell. Ett språk på begge sider av et system lar nettleseren og serveren dele en valideringsregel eller en dataform, og noen produkter trenger mye skript: tilstand uten nett, dra og slipp, kart, samarbeid i sanntid. Det nyttige spørsmålet stilles per funksjon: må dette skje i leserens nettleser uten å spørre en server? Hvis ikke, gjengi det på serveren eller ved byggetid og send HTML.

Det beslektede valget er JavaScript mot TypeScript. TypeScript legger til en typekontroll og et byggesteg, og kompileres tilbake til ren JavaScript. For en kodebase som flere personer endrer hver uke, betaler typekontrollen som regel for seg selv; for en forbedring på hundre linjer på en statisk side holder det med JavaScript og en streng linter.

Slik ser det ut i praksis

Dette nettstedet leverer statiske sider uten kjørbar JavaScript til leserne. Søkekoden er bevart, men slått av; JavaScript brukes fortsatt i verktøyene for bygging og kontroll. Tre sider viser det tidligere nettleserunntaket og arbeidet utenfor nettleseren.

En tidligere versjon på tre språk hadde ingen backend ved kjøretid og leverte ett førstepartsskript. Beskrivelsen av Hugo-bygget dokumenterer det avgrensede unntaket: et porteføljefilter med kontroller som var skjult til skriptet kjørte. De 108 malfilene, inkludert 53 partialer og 18 kortkoder, gjorde arbeidet under bygging. Beskrivelsen nevner også kostnaden den gangen: interaktive funksjoner måtte løses i bygget, og en søkeindeks ble vurdert og lagt til side fremfor levert. Tallene og unntaket beskriver den tidligere løsningen, ikke koden leseren mottar nå.

Det samme innholdet publiseres over Gemini og Gopher, to protokoller uten plass til skript i det hele tatt. Ett bygg skrev 777 dokumenter i hvert format på det dokumenterte implementeringstidspunktet fra den samme kilden som sidene. En separat sammenligning av bygg bekreftet null endrede HTML-byte da disse utdataformatene ble lagt til; resultatet omfatter ikke senere redaksjonelle eller visuelle endringer. Nettstedet argumenterer for at et dokument ikke trenger en megabyte skript for å kunne leses, og et format som ikke kan romme skript, er den tydeligste måten å vise det på.

Det meste av JavaScript-koden her ligger i verktøyene. På implementeringstidspunktet som det lenkede arbeidet dokumenterer, besto kvalitetsporten på nettstedet av trettito skript og 10 242 linjer JavaScript, da den største kodemengden i repoet. En tid var dette den eneste koden som verken ble formatert, lintet eller typekontrollert. Å få den under en formaterer og en linter begynte med å tilpasse formatereren til koden: repoets standard på fire mellomrom ville ha skrevet om 2 173 linjer i det største kontrollskriptet, og en overstyring til to mellomrom for de filene etterlot 38 linjer med reelle avvik.

Linteren ga deretter 13 funn, og alle 13 ble rettet, ingen undertrykt. En feil i et kontrollskript viser seg som en kontroll som består når den burde feile, så portens egen kode får den samme behandlingen som den gir alt annet.

Hvor det går galt

Feilene nedenfor følger av hvordan språket er bygd, og hver av dem har en kjent løsning.

Siden tegnes, men ignorerer trykk i et sekund eller mer. En lang oppgave, som å hydrere en stor kodebunt, parse et stort JSON-svar eller sortere tusenvis av rader, holder den ene tråden som håndterer inndata. Send mindre kode for den første interaksjonen ved å dele kodebuntene per rute og utsette resten, del opp langt arbeid i biter som gir kontrollen tilbake til hendelsesløkken, og flytt tunge beregninger til en Web Worker. Mål på en treg telefon og ikke på den bærbare maskinen koden ble skrevet på.

Priser og identifikatorer kommer ut litt feil. Hvert tall er et binært flyttall, så desimalbrøker rundes av, og svært store heltall kollapser inn i naboene sine. En 64-biters databasenøkkel sendt som et JSON-tall er det vanlige offeret:

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

Hold pengebeløp i heltall av den minste enheten, som øre eller cent, send store identifikatorer som strenger, og bruk `BigInt` der regnestykket må være eksakt.

En sammenligning slipper gjennom feil verdi. Operatoren `==` konverterer operandene før den sammenligner dem, så `'' == 0` og `'0' == false` er begge sanne. Bruk `===` overalt og la regelen `eqeqeq` i linteren håndheve det, slik at konverteringen blir noe koden ber uttrykkelig om.

Tekst fra en bruker ender opp med å kjøre som kode. Å tilordne ubetrodd tekst til `innerHTML`, bygge markup ved å skjøte sammen strenger eller sende inndata til `eval` gjør data om til skript. Skriv tekst med `textContent`, send en

Content Security Policy som bare slipper inn skript med nonce eller hash, og hold avhengighetslisten kort og låst i en låsefil, og les hva hver oppdatering trekker inn.

Kode som virker i nettleseren, krasjer på serveren. Språket flyttet seg, men verten gjorde det ikke: `window` og `document` finnes bare i en nettleser, og Node.js leser en fil som CommonJS eller som en ES-modul avhengig av innstillinger nettleseren aldri ser. Hold vertsspesifikke kall i kantene, sett feltet `type` til `module` eksplisitt i `package.json`, og kjør hver kodebit i det miljøet den faktisk skal kjøre i.

Les mer

- ECMAScript Language Specification — det levende utkastet til ECMA-262, teksten hver motor implementerer, oppdatert etter hvert som forslag når trinn 4.
- The TC39 Process — trinnene et forslag går gjennom før det blir en del av språket, og hva hvert av dem krever.
- ECMA-262 hos Ecma International — de publiserte årlige utgavene, for når et prosjekt må navngi versjonen det sikter mot.
- HTML Standard: event loops — hvordan en nettleser planlegger oppgaver, mikrooppgaver og tegning rundt skriptene på en side.
- HTML Standard: the script element — klassiske skript og modulskript, og hvordan `async` og `defer` endrer når de kjører.

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 JavaScript?» 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/javascript/