Engineer

JavaScript

språket som webbläsare kör

Vad det är

JavaScript är ett dynamiskt typat programspråk, standardiserat som ECMAScript, som varje webbläsare kan köra och som också körs på servrar och byggmaskiner genom körmiljöer som Node.js. Det kortas ofta till JS. Standardens årliga utgåvor heter ES2015, ES2024 och så vidare, den äldre numreringen (ES5, ES6) hörs fortfarande, och ”vanilla JavaScript” betyder språket utan något ramverk ovanpå. Utöver namnet och en del syntax med klammerparenteser har det lite gemensamt med Java.

Brendan Eich skrev den första versionen på Netscape, och den levererades i webbläsaren Navigator 2.0. Ecma International antog den första utgåvan av standarden, ECMA-262, i juni 1997, och dess kommitté TC39 har gett ut en ny utgåva varje år sedan ECMAScript 2016. Ett förslag kommer in i specifikationen först i steg 4 av kommitténs process, som kräver implementationer som redan har levererats, så texten och motorerna är sällan oense länge.

Språket säger väldigt lite om omvärlden. ECMA-262 definierar syntaxen, typerna, objekten, löftena och modulerna, och lämnar in- och utdata åt den miljö som kör språket. I en webbläsare är den miljön webbplattformen: dokumentet, händelserna, timrarna och nätverksbegärandena, specificerade till största delen av HTML-standarden och dess systerstandarder. I Node.js står miljön i stället för filer, socketar och processer. Den uppdelningen förklarar en vanlig överraskning: kod kan vara fullt giltig JavaScript och ändå gå sönder när den flyttas från en sida till en server, eftersom språket följde med och miljön stannade kvar.

I en webbläsare delar en sidas skript en enda huvudtråd med layout, ritning och inmatning. En uppgift körs till slut innan webbläsaren kan svara på nästa klick. Löften och `async` -funktioner håller väntandet borta från den tråden, medan

beräkningar stannar på den om de inte flyttas till en Web Worker. Tal är flyttal med dubbel precision enligt IEEE 754, så heltal är exakta bara upp till 2^53 − 1, och allt större än så kräver `BigInt`.

Använd begreppet i ett beslut

Använd ”Vad är JavaScript?” 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

Det verkliga beslutet är hur mycket av det du skickar till dem som läser webbplatsen. JavaScript är språket som får en sida att bete sig utan en tur till en server, och varje kilobyte av det laddas ned, tolkas, kompileras och körs på en enhet som du inte styr över. Paketet som känns omedelbart på utvecklarens bärbara dator kan hålla en mellanklasstelefon upptagen i flera sekunder, och läsaren betalar med batteri och tålamod.

Skript är också där en sida är mest utsatt. Om en angripares skript körs har det sidans befogenheter: det kan läsa vad läsaren skriver och skicka det vart som helst som sidan får skicka. Det är den risk som Content Security Policy finns till för att begränsa. När inget körbart JavaScript levereras försvinner webbplatsens egna webbläsarskript som angreppsyta; indata som inte går att lita på måste fortfarande hanteras säkert, och en restriktiv policy behövs fortfarande. Paketekosystemet ger ytterligare en angreppsyta. Ett typiskt bygge av en frontend drar in hundratals transitiva beroenden, och vart och ett av dem är kod som körs med samma rättigheter som din.

Den andra sidan av vågskålen är verklig. Med ett språk på båda sidor av ett system kan webbläsaren och servern dela en valideringsregel eller en dataform, och vissa produkter behöver mycket skript: tillstånd offline, dra och släpp, kartor, samarbete i realtid. Den användbara frågan ställs per funktion: måste det här ske i läsarens webbläsare utan att fråga en server? Om inte, rendera det på servern eller vid byggtid och skicka HTML.

Det närliggande valet är JavaScript eller TypeScript. TypeScript lägger till en typkontroll och ett byggsteg och kompileras tillbaka till vanlig JavaScript. För en kodbas som flera personer ändrar varje vecka brukar typkontrollen betala sig; för en förbättring på hundra rader på en statisk sida räcker JavaScript och en strikt linter.

Så ser det ut i praktiken

Den här webbplatsen levererar statiska sidor utan körbar JavaScript till läsarna. Sökkoden finns kvar men är avstängd; JavaScript används fortfarande i verktygen för bygge och kontroll. Tre sidor visar det tidigare webbläsarundantaget och arbetet utanför webbläsaren.

En tidigare version på tre språk hade ingen backend vid körning och levererade ett enda förstapartsskript. Beskrivningen av Hugo-bygget dokumenterar det avgränsade undantaget: ett portföljfilter vars kontroller var dolda tills skriptet kördes. De 108 mallfilerna, däribland 53 partialer och 18 kortkoder, gjorde arbetet vid bygget. Beskrivningen anger också kostnaden då: interaktiva funktioner måste lösas i bygget, och ett sökindex bedömdes och lades åt sidan i stället för att levereras. Siffrorna och undantaget beskriver den tidigare lösningen, inte koden som läsaren tar emot nu.

Samma innehåll publiceras över Gemini och Gopher, två protokoll som inte har någon plats alls för ett skript. Ett bygge skrev 777 dokument i varje format i det dokumenterade implementeringsskedet ur samma källa som sidorna. En separat jämförelse av byggen bekräftade noll ändrade HTML-byte när utdataformaten lades till; resultatet omfattar inte senare redaktionella eller visuella ändringar. Webbplatsens argument är att ett dokument inte behöver en megabyte skript för att kunna läsas, och ett format som inte kan bära ett enda är det tydligaste sättet att visa det.

Det mesta av JavaScript-koden här bor i verktygen. I det implementeringsskede som det länkade arbetet dokumenterar bestod webbplatsens kvalitetsgrind av trettiotvå skript och 10 242 rader JavaScript, då den största mängden kod i repot. Under en tid var det den enda kod som varken formaterades, lintades eller typkontrollerades. Arbetet med att få den under en formaterare och en linter började med att anpassa formateraren efter koden: repots standard på fyra mellanslag skulle ha skrivit om 2 173 rader i det största kontrollprogrammet, och en särskild inställning med två mellanslag för de filerna lämnade 38 rader med reella formateringsavvikelser. Lintern rapporterade sedan tretton anmärkningar,

och alla tretton åtgärdades utan att något undertrycktes. Ett fel i ett kontrollprogram visar sig som en kontroll som går igenom när den borde fallera, så grindens egen kod får samma behandling som den ger allt annat.

Var det går fel

Felen nedan följer av hur språket är byggt, och vart och ett har en känd åtgärd.

Sidan ritas upp och ignorerar sedan tryck i en sekund eller mer. En lång uppgift, som att hydrera ett stort paket, tolka ett stort JSON-svar eller sortera tusentals rader, håller den enda tråd som hanterar inmatning. Skicka mindre kod för den första interaktionen genom att dela upp paketen per rutt och skjuta upp resten, bryt upp långt arbete i bitar som lämnar tillbaka kontrollen till händelseloopen, och flytta tunga beräkningar till en Web Worker. Mät på en långsam telefon i stället för på den bärbara dator där koden skrevs.

Priser och identifierare blir en aning fel. Varje tal är ett binärt flyttal, så decimala bråk avrundas och mycket stora heltal faller ihop med sina grannar. En 64-bitars databasnyckel som skickas som ett tal i JSON är det vanliga offret:

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

Håll pengar i heltal av den minsta enheten, t.ex. ören eller cent, skicka stora identifierare som strängar och använd `BigInt` där aritmetiken måste vara exakt.

En jämförelse släpper igenom fel värde. Operatorn `==` konverterar sina operander innan den jämför dem, så både `"" == 0` och `"0" == false` är sanna. Använd `===` överallt och låt linterregeln `eqeqeq` upprätthålla det, så att konverteringen blir något som koden ber om öppet.

Text från en användare körs till slut som kod. Att tilldela opålitlig text till `innerHTML`, bygga kod genom att foga ihop strängar eller lämna indata till `eval` gör data till skript. Skriv text med `textContent`, skicka en Content Security Policy som bara släpper in skript via nonce eller hash, och håll listan över beroenden kort och låst i en låsfil, och läs vad varje uppdatering drar in.

Kod som fungerar i webbläsaren kraschar på servern. Språket flyttade och miljön gjorde det inte: `window` och `document` finns bara i en webbläsare, och Node.js läser en fil som CommonJS eller som en ES-modul beroende på inställningar som webbläsaren aldrig ser. Håll miljöspecifika anrop i kanterna, sätt `"type": "module"` uttryckligen i `package.json` och kör varje del av koden i den miljö där den faktiskt ska köras.

Läs mer

- ECMAScript Language Specification — det levande utkastet till ECMA-262, texten som varje motor implementerar, uppdaterad när förslag når steg 4.
- The TC39 Process — stegen som ett förslag passerar innan det blir en del av språket, och vad varje steg kräver.
- ECMA-262 at Ecma International — de publicerade årliga utgåvorna, för när ett projekt måste ange vilken version det siktar på.
- HTML Standard: event loops — hur en webbläsare schemalägger uppgifter, mikrouppgifter och rendering runt en sidas skript.
- HTML Standard: the script element — klassiska skript och modulskript, och hur `async` och `defer` ändrar när de körs.

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 JavaScript?” 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/javascript/