Engineer

Web components

🖼 browserfunktioner til nye, genbrugelige HTML-elementer

Hvad det er

Web components, også skrevet Web Components med store bogstaver, er det gængse navn på en gruppe browserfunktioner, der lader udviklere definere deres egne HTML-elementer. En side bruger derefter sådan et element som ethvert andet tag. Standarderne kalder delene ved deres egne navne: custom elements og elementerne `template` og `slot` i HTML-standarden og shadow trees i DOM-standarden. Fællesnavnet er titlen på *Introduction to Web Components*, en Working Group Note, som W3C udgav den 24. juli 2014 og siden har indstillet og markeret som et dokument, der ikke bør henvises til.

Et custom element defineres af en JavaScript-klasse, der udvider `HTMLElement` og registreres under et navn med `customElements.define()`. Definitionen fortæller HTML-parseren, hvordan elementet skal konstrueres, og hvordan det skal reagere på ændringer, og fra da af kan elementet oprettes fra markup, med `document.createElement()` eller med `new`. Navnet begynder med et lille ASCII-bogstav, indeholder ingen store ASCII-bogstaver og skal indeholde en bindestreg. Bindestregen holder egne navne adskilt fra selve sproget: HTML-standarden lover, at der fremover ikke tilføjes noget element med bindestreg i navnet til HTML, SVG eller MathML.

Der findes to slags. Et autonomt custom element, som standardens eget `<flag-icon country="nl">`, tager udgangspunkt i et almindeligt element uden indbygget betydning. Et customized built-in element udvider et eksisterende element: en klasse, der udvider `HTMLButtonElement`, registreres med `{ extends: "button" }` og bruges som `<button is="plastic-button">`, beholder alt det, der gør en knap til en knap, også fokusadfærd, formularindsendelse og attributten `disabled`.

Et shadow tree er et separat træ af noder, der er knyttet til et element, som kaldes dets vært; DOM-standarden kalder værtens almindelige træ for light tree. `attachShadow()` opretter en shadow root i tilstanden open eller closed, og kun i tilstanden open returnerer værtens egenskab `shadowRoot` den. Et `slot` -element i værtens shadow tree markerer, hvor værtens egne børn vises. Det viser de noder, der er tildelt det, eller sit eget indhold, når ingen er tildelt, og en navngivet slot tager de børn, hvis `slot` -attribut bærer dens navn. Elementet `template` rummer et HTML-fragment, som ikke vises, og som et script kan klone. Med attributten `shadowrootmode` repræsenterer en `template` en deklarativ shadow root, så et shadow tree også kan skrives direkte i markup.

Brug begrebet i en beslutning

Brug »Hvad er web components?« 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

Et opfundet tag var tidligere ikke til megen nytte: HTML-standarden bemærker, at den slags elementer historisk har været i strid med standarden og ikke særlig funktionelle. Et custom element er et fuldgyldigt medlem af dokumentet. Parseren konstruerer det, DOM-metoderne opretter det, og dets klasse bestemmer, hvad der sker, når dets attributter ændres, eller når det kommer ind på eller forlader en side.

Definitionen behøver ikke komme først. Et element, som parseren allerede har oprettet, bliver et custom element, så snart en definition for dets navn registreres, og HTML-standarden kalder det en opgradering. Indtil da behandles elementet som et udefineret element, stort set som en `span` , så teksten og linkene i det vises, som de er. Standarden siger ligeud, at opgraderinger gør det muligt at forbedre indholdet i et custom element progressivt.

Et custom element kan også indgå i formularer og i tilgængelighedstræet. En klasse, der sætter `static formAssociated = true` og kalder `attachInternals()` , får et `ElementInternals` -objekt, og `setFormValue()` på det objekt giver elementet en værdi, som formularen sender med ligesom enhver anden kontrols. Det samme

objekt sætter en standardrolle og standardtilstande i ARIA, som gælder, medmindre sidens forfatter tilsidesætter dem med attributten `role` eller de tilsvarende `aria-*` -attributter, og det kan rumme egne tilstande, som stylesheets rammer med pseudoklassen `:state()` .

For et team er gevinsten ved et shadow tree, at komponenten bliver selvstændig: almindelige selektorer på siden krydser ikke ind i komponentens interne shadow tree, og komponenten bestemmer, hvilke dele den vil gøre tilgængelige for styling udefra. Nedarvede værdier og custom properties kan stadig nå ind, og indhold fra light tree, der vises gennem slots, er stadig omfattet af sidens selektorer. Hvor flere biblioteker deler en side, tilbyder HTML-standarden scoped registries, oprettet med `new CustomElementRegistry()` , så deres elementer kan sameksistere, uden at bibliotekerne skal koordinere deres navne.

Sådan ser det ud i praksis

Et produktkort viser delene i samspil. Siden rummer indholdet som almindelig markup, med en `slot` -attribut på det element, der skal ind i en navngivet slot:

```
<product-card>
  <span slot="name">Egeskammel</span>
  <a href="/butik/egeskammel/">Se detaljerne</a>
</product-card>
<script type="module" src="product-card.mjs"></script>
```

Modulet definerer elementet. Skabelonen bygges én gang, og hvert kort får sin egen kopi af den i en åben shadow root:

```
const template = document.createElement("template");
template.innerHTML = `
  <h2 part="title"><slot name="name">Produkt uden navn</slot></h2>
  <slot></slot>
`;

class ProductCard extends HTMLElement {
  constructor() {
    super();
    const root = this.attachShadow({ mode: "open" });
    root.append(template.content.cloneNode(true));
  }
}
```

```
if (!customElements.get("product-card")) {
  customElements.define("product-card", ProductCard);
}
```

Siden styler titlen gennem den ene del, komponenten stiller til rådighed:

```
product-card::part(title) {
  font-size: 1.25rem;
}
```

Før `product-card.mjs` er kørt, er kortet et udefineret element, og læseren ser produktnavnet og linket som almindeligt indhold i teksten. Når modulet kører, registrerer `define()` klassen, og det kort, der allerede står på siden, bliver opgraderet. Konstruktøren knytter en shadow root til kortet og fylder den med en klon af skabelonens indhold. `span` -elementet tildeles den slot, der hedder `name` , og vises inde i overskriften. Linket har ingen `slot` -attribut, så det havner i den unavngivne standardslot. Et kort uden navn viser i stedet slottens eget indhold, »Produkt uden navn«. Tjekket med `customElements.get()` forhindrer, at en ekstra kopi af modulet, der ankommer i et andet bundle, får `define()` til at kaste en fejl.

Fælles komponenter hjælper kun, hvis folk bruger dem, og dette sites portefølje har en post om netop det. Liquid Glass-designsystemet til iOS 26 tager udgangspunkt i en brugerflade, hvor hver ny skærm genopfinder sine egne knapper, badges og kort, alle lidt forskellige, indtil hver ændring betyder, at fem specialbyggede versioner skal rettes. Svaret, som siden beskriver, er et kanonisk sæt komponenter, som alle skærme sammensættes af, blandt andet Card, Label, Button, GlassIconButton og DirectoryGrid, med lint-regler, der afviser et hjemmelavet badge eller en hjemmelavet chip og henviser til den kanoniske komponent. Siden siger ikke, om komponenterne er custom elements, men disciplinen kan overføres: én definition pr. komponent, holdt i kraft af værktøjer og ikke af en reviewers hukommelse. Registret for custom elements håndhæver den første halvdel ved at afvise en ny definition for et navn, der allerede er i brug.

Hvor det går galt

Et navn med bindestreg skal gøre det ud for en rolle. »An autonomous custom element does not have any special meaning: it represents its children,« skriver HTML-standarden. Standardens eksempel `taco-button` viser, hvad det koster: for at opføre sig som en knap har elementet brug for en `tabindex`, ARIA-rollen `button`, et tilgængeligt navn, `aria-disabled`, tastaturhåndtering, der gør Enter og mellemrumstasten til klik, og styles til den deaktiverede tilstand. Selv da kan et custom element ikke erklære en standardadfærd for fokus, så `tabindex="0"` passer måske ikke til den måde, knapper opfører sig på på platformen. Hvor elementet bygger på et eksisterende element, siger standarden, at et customized built-in element vil være lettere at udvikle, vedligeholde og bruge. Sæt for et autonomt element standardrollen og standardtilstandene via `ElementInternals` i stedet for at overlade dem til hver side, der bruger det.

Konstruktøren gør arbejdet. HTML-standarden stiller krav til konstruktører i custom elements: `super()` uden argumenter kommer først, elementets attributter og børn må ikke undersøges, for et element, der er oprettet og ikke opgraderet, har ingen, og elementet må ikke selv få attributter eller børn. En konstruktør, der bryder reglerne, kan give et element, som parseren eller DOM-metoderne ikke kan oprette. Brug konstruktøren til starttilstand, standardværdier, event listeners og shadow root, og flyt hentning og rendering til `connectedCallback`.

`connectedCallback` **forventes at køre én gang.** Den kan køre mere end én gang. Som standard kører en flytning af elementet først `disconnectedCallback` og så `connectedCallback`, så elementet nulstiller sin tilstand, som om det var blevet fjernet og indsat igen. Beskyt opsætning, der kun må ske én gang, med et flag, og ryd i `disconnectedCallback` op efter det, som `connectedCallback` satte i gang. Skal et element beholde sin tilstand, når det flyttes med `moveBefore()`, så definér `connectedMoveCallback`: alene det, at den findes, erstatter standardparret, også med en tom krop.

Et navn defineres to gange. `customElements.define()` kaster en `NotSupportedError`, når navnet eller klassen allerede er registreret, og en `SyntaxError`, når navnet ikke er gyldigt, for eksempel fordi det mangler en bindestreg eller indeholder et stort bogstav. Tjek med `customElements.get()`, før du definerer, og giv et bibliotek, der skal sameksistere med andre, sit eget scoped registry.

Tilstanden closed opfattes som beskyttelse. Tilstanden closed skjuler roden for værtens egenskab `shadowRoot` og skjuler interne noder i en hændelses composed path, sådan som den ses af lyttere uden for det lukkede shadow tree. CSS Shadow Parts siger ligeud, at shadow trees med vilje ikke er en sikkerhedsgrænse, kun en bekvemmelighed for sidens forfattere. Hold hemmeligheder ude af komponenter, og vælg tilstanden closed for dens virkning på kode udefra, ikke som en lås.

`attachShadow()` **kaldes, hvor det ikke kan virke.** DOM-standarden tillader kun en shadow root på elementer med et gyldigt navn til custom elements og på en kort liste af indbyggede elementer, blandt dem `div`, `section`, `p` og `span`, men ikke `button`. På en customized built-in knap kaster kaldet en `NotSupportedError`. Det opfører sig også anderledes på et element, der allerede har en rod: er den rod deklarativ og i samme tilstand, fjerner kaldet alt i den og returnerer den tom, og ellers kaster det en fejl. Tjek først, om der allerede er en shadow root, så markup, som serveren har sendt i en deklarativ rod, bevares og ikke slettes.

Sidens styles forventes at nå ind. CSS Shadow Parts kalder grænsen en stylingbarriere og bemærker, at den også gør en komponent sværere at style, når siden gerne vil. Udstil de elementer, siden må style, med `part`, og style dem udefra med `::part()`. En komponent inde i en anden har brug for `exportparts` på den indre vært for at give sine dele videre til siden. Brug custom properties sat på værten til farver og afstande, der skal følge siden; de arves ind i komponentens shadow tree.

Indholdet findes først efter scriptet. En komponent, der skriver al sin tekst fra JavaScript, viser ikke læserne noget af den, før definitionen er indlæst, og slet intet, hvis scriptet fejler. Hold indholdet i light tree, og tildel det til slots, som produktkortet gør, så opgraderingen forbedrer det, der allerede står der. Hvor indhold må vente, skjuler HTML-standardens eget eksempel det, indtil `customElements.whenDefined()` er opfyldt for hvert element, der matcher `:not(:defined)`.

Læs mere

- HTML Standard: Custom elements — navneregler, krav til konstruktører, opgraderinger, formulartilknyttede elementer og `ElementInternals`, med gennemgåede eksempler.
- DOM Standard: Shadow tree — shadow roots og deres værter, slots og de noder, der tildeles dem, og trinene bag `attachShadow()`.

- HTML Standard: Scripting — elementet `template` med attributten `shadowrootmode` og elementet `slot`.
- CSS Shadow Parts — W3C's Working Draft fra den 16. december 2025, som definerer `part`, `exportparts` og pseudoelementet `::part()`.

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 web components?« 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/web-components/