Engineer

Web components

🖼 nettleserfunksjoner for nye, gjenbrukbare HTML-elementer

Hva det er

Web components, også skrevet Web Components med store bokstaver, er det vanlige navnet på en gruppe nettleserfunksjoner som lar utviklere definere sine egne HTML-elementer. En side bruker deretter et slikt element som en hvilken som helst annen tagg. Standardene kaller delene ved deres egne navn: custom elements og elementene `template` og `slot` i HTML-standarden, og shadow trees i DOM-standarden. Fellesnavnet er tittelen på *Introduction to Web Components*, en Working Group Note som W3C ga ut 24. juli 2014 og senere har avviklet og merket som et dokument det ikke bør henvises til.

Et custom element defineres av en JavaScript-klasse som utvider `HTMLElement` og registreres under et navn med `customElements.define()`. Definisjonen forteller HTML-parseren hvordan elementet skal konstrueres og hvordan det skal reagere på endringer, og fra da av kan elementet opprettes fra markup, med `document.createElement()` eller med `new`. Navnet begynner med en liten ASCII-bokstav, inneholder ingen store ASCII-bokstaver og må inneholde en bindestrek. Bindestreken holder egne navn atskilt fra selve språket: HTML-standarden lover at det heretter ikke legges til noe element med bindestrek i navnet i HTML, SVG eller MathML.

Det finnes to slags. Et autonomt custom element, som standardens eget `<flag-icon country="nl">`, tar utgangspunkt i et vanlig element uten innebygd betydning. Et customized built-in element utvider et eksisterende element: en klasse som utvider `HTMLButtonElement`, registreres med `{ extends: "button" }` og brukes som `<button is="plastic-button">`, beholder alt som gjør en knapp til en knapp, blant annet fokusoppførsel, innsending av skjemaer og attributtet `disabled`.

Et shadow tree er et eget tre av noder som er knyttet til et element, kalt verten; DOM-standarden kaller vertens vanlige tre for light tree. `attachShadow()` oppretter en shadow root i modusen open eller closed, og bare i modusen open returnerer vertens egenskap `shadowRoot` den. Et `slot` -element i vertens shadow tree markerer hvor vertens egne barn vises. Det viser nodene som er tilordnet det, eller sitt eget innhold når ingen er tilordnet, og en navngitt slot tar de barna som har dens navn i `slot` -attributtet. Elementet `template` rommer et HTML-fragment som ikke vises, og som et skript kan klone. Med attributtet `shadowrootmode` representerer en `template` en deklarativ shadow root, slik at et shadow tree også kan skrives rett inn i markupen.

Bruk begrepet i en beslutning

Bruk «Hva er web components?» 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

En oppdiktet tagg var tidligere til liten nytte: HTML-standarden påpeker at slike elementer historisk har vært i strid med standarden og ikke særlig funksjonelle. Et custom element er et fullverdig medlem av dokumentet. Parseren konstruerer det, DOM-metodene oppretter det, og klassen bestemmer hva som skjer når attributtene endres, eller når elementet kommer inn på eller forlater en side.

Definisjonen trenger ikke komme først. Et element som parseren allerede har opprettet, blir et custom element så snart en definisjon for navnet registreres, og HTML-standarden kaller dette en oppgradering. Fram til da behandles elementet som et udefinert element, omtrent som en `span` , slik at teksten og lenkene i det vises som de er. Standarden sier rett ut at oppgraderinger gjør det mulig å forbedre innholdet i et custom element progressivt.

Et custom element kan også delta i skjemaer og i tilgjengelighetstreet. En klasse som setter `static formAssociated = true` og kaller `attachInternals()` , får et `ElementInternals` -objekt, og `setFormValue()` på dette objektet gir elementet en verdi som skjemaet sender inn på samme måte som for enhver annen kontroll. Det samme objektet setter en standardrolle og standardtilstander i ARIA, som

gjelder med mindre sidens forfatter overstyrer dem med attributtet `role` eller de tilsvarende `aria-*` -attributtene, og det kan holde egne tilstander som stilark treffer med pseudoklassen `:state()` .

For et team er gevinsten ved et shadow tree at komponenten blir selvstendig: vanlige selektorer på siden krysser ikke inn i komponentens interne shadow tree, og komponenten bestemmer hvilke deler den vil eksponere for styling utenfra. Arvede verdier og custom properties kan fortsatt nå inn, og innhold fra light tree som vises i slots, er fortsatt underlagt sidens selektorer. Der flere biblioteker deler en side, tilbyr HTML-standarden scoped registries, opprettet med `new CustomElementRegistry()` , slik at elementene deres kan leve side om side uten at bibliotekene må samordne navnene sine.

Slik ser det ut i praksis

Et produktkort viser delene i samspill. Siden har innholdet som vanlig markup, med et `slot` -attributt på elementet som skal inn i en navngitt slot:

```
<product-card>
  <span slot="name">Eikekrakk</span>
  <a href="/butikk/eikekrakk/">Se detaljene</a>
</product-card>
<script type="module" src="product-card.mjs"></script>
```

Modulen definerer elementet. Malen bygges én gang, og hvert kort får sin egen kopi av den i en åpen shadow root:

```
const template = document.createElement("template");
template.innerHTML = `
  <h2 part="title"><slot name="name">Produkt uten 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 tittelen gjennom den ene delen komponenten eksponerer:

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

Før `product-card.mjs` har kjørt, er kortet et udefinert element, og leseren ser produktnavnet og lenken som vanlig innhold i teksten. Når modulen kjører, registrerer `define()` klassen, og kortet som allerede står på siden, blir oppgradert. Konstruktøren knytter en shadow root til kortet og fyller den med en klone av malens innhold. `span` -elementet tilordnes sloten som heter `name` , og vises inne i overskriften. Lenken har ikke noe `slot` -attributt, så den havner i den navnløse standardsloten. Et kort uten navn viser i stedet slotens eget innhold, «Produkt uten navn». Sjekken med `customElements.get()` hindrer at en ekstra kopi av modulen, som kommer med en annen bundle, får `define()` til å kaste en feil.

Felles komponenter hjelper bare hvis folk bruker dem, og nettstedets portefølje har en oppføring om akkurat det. Liquid Glass-designsystemet for iOS 26 tar utgangspunkt i et brukergrensesnitt der hver ny skjerm finner opp sine egne knapper, merker og kort, alle litt ulike, til hver endring betyr å rette fem skreddersydde versjoner. Svaret siden beskriver, er et kanonisk sett med komponenter som alle skjermer settes sammen av, blant dem Card, Label, Button, GlassIconButton og DirectoryGrid, med lint-regler som avviser et hjemmelaget merke eller en hjemmelaget brikke og peker mot den kanoniske komponenten. Siden sier ikke om komponentene er custom elements, men disiplinen lar seg overføre: én definisjon per komponent, holdt ved like av verktøy og ikke av hukommelsen til den som gjør gjennomgangen. Registeret for custom elements håndhever den første halvdelen ved å avvise en ny definisjon for et navn som allerede er i bruk.

Hvor det går galt

Et navn med bindestrek skal gjøre jobben til en rolle. «An autonomous custom element does not have any special meaning: it represents its children,» skriver HTML-standarden. Standardens eksempel `taco-button` viser hva det koster: for å oppføre seg som en knapp trenger elementet en `tabindex`, ARIA-rollen `button`, et tilgjengelig navn, `aria-disabled`, tastaturhåndtering som gjør Enter og mellomromstasten til klikk, og stiler for deaktivert tilstand. Selv da kan et custom element ikke angi en standard fokusoppførsel, så `tabindex="0"` stemmer kanskje ikke med hvordan knapper oppfører seg på plattformen. Der elementet bygger på et eksisterende element, sier standarden at et customized built-in element vil være enklere å utvikle, vedlikeholde og bruke. For et autonomt element bør du sette standardrollen og standardtilstandene via `ElementInternals` i stedet for å overlate dem til hver side som bruker det.

Konstruktøren gjør arbeidet. HTML-standarden stiller krav til konstruktører i custom elements: `super()` uten argumenter kommer først, elementets attributter og barn skal ikke undersøkes, fordi et element som er opprettet og ikke oppgradert, ikke har noen, og elementet skal ikke selv få attributter eller barn. En konstruktør som bryter reglene, kan gi et element som parseren eller DOM-metodene ikke klarer å opprette. Bruk konstruktøren til starttilstand, standardverdier, hendelseslyttere og shadow root, og flytt henting og rendering til `connectedCallback`.

`connectedCallback` **forventes å kjøre én gang.** Den kan kjøre mer enn én gang. Som standard fører en flytting av elementet til at `disconnectedCallback` og deretter `connectedCallback` kjører, så elementet nullstiller tilstanden sin som om det var fjernet og satt inn på nytt. Beskytt oppsett som bare skal skje én gang, med et flagg, og rydd i `disconnectedCallback` opp etter det `connectedCallback` satte i gang. Skal et element beholde tilstanden sin når det flyttes med `moveBefore()`, definerer du `connectedMoveCallback`: at den finnes, er nok til å erstatte standardparet, selv med en tom kropp.

Et navn defineres to ganger. `customElements.define()` kaster en `NotSupportedError` når navnet eller klassen allerede er registrert, og en `SyntaxError` når navnet ikke er gyldig, for eksempel fordi det mangler bindestrek eller inneholder en stor bokstav. Sjekk med `customElements.get()` før du definerer, og gi et bibliotek som må leve side om side med andre, sitt eget scoped registry.

Modusen closed blir tatt for beskyttelse. Modusen closed skjuler roten for vertens egenskap `shadowRoot` og skjuler interne noder i en hendelses composed path slik den ses av lyttere utenfor det lukkede shadow tree-et. CSS Shadow Parts sier rett ut at shadow trees med vilje ikke er en sikkerhetsgrense, bare en bekvemmelighet for sideforfattere. Hold hemmeligheter utenfor komponentene, og velg modusen closed for virkningen på kode utenfra, ikke som en lås.

`attachShadow()` **kalles der det ikke kan virke.** DOM-standarden tillater en shadow root bare på elementer med et gyldig navn for custom elements og på en kort liste med innebygde elementer, blant dem `div`, `section`, `p` og `span`, men ikke `button`. På en customized built-in knapp kaster kallet en `NotSupportedError`. Det oppfører seg også annerledes på et element som allerede har en rot: er roten deklarativ og i samme modus, fjerner kallet alt i den og returnerer den tom, ellers kaster det en feil. Sjekk først om det allerede finnes en shadow root, slik at markup som serveren har sendt i en deklarativ rot, blir beholdt og ikke slettet.

Sidens stiler forventes å nå inn. CSS Shadow Parts kaller grensen en stylingbarriere og påpeker at den også gjør en komponent vanskeligere å style når siden faktisk ønsker det. Eksponer elementene siden får style, med `part`, og style dem utenfra med `::part()`. En komponent inne i en annen trenger `exportparts` på den indre verten for å sende delene sine videre til siden. Bruk custom properties satt på verten til farger og avstander som skal følge siden; de arves inn i komponentens shadow tree.

Innholdet finnes først etter skriptet. En komponent som skriver all teksten sin fra JavaScript, viser ikke leserne noe av den før definisjonen er lastet, og ingenting i det hele tatt hvis skriptet feiler. Hold innholdet i light tree og tilordne det til slots, slik produktkortet gjør, så oppgraderingen forbedrer det som allerede står der. Der innhold må vente, skjuler HTML-standardens eget eksempel det til `customElements.whenDefined()` er innfridd for hvert element som matcher `:not(:defined)`.

Les mer

- HTML Standard: Custom elements — navneregler, krav til konstruktører, oppgraderinger, skjematilknyttede elementer og `ElementInternals`, med gjennomgåtte eksempler.

- DOM Standard: Shadow tree — shadow roots og vertene deres, slots og nodene som tilordnes dem, og trinnene bak `attachShadow()`.
- HTML Standard: Scripting — elementet `template` med attributtet `shadowrootmode` og elementet `slot`.
- CSS Shadow Parts — W3Cs Working Draft fra 16. desember 2025, som definerer `part`, `exportparts` og pseudoelementet `::part()`.

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 web components?» 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/web-components/