Engineer

Web Components

🖼 Browserfunktionen für neue, wiederverwendbare HTML-Elemente

Was es ist

Web Components ist die übliche Sammelbezeichnung für Browserfunktionen, mit denen Autoren eigene HTML-Elemente definieren können. Eine Seite verwendet ein solches Element anschließend wie jedes andere Tag. Im Englischen wird der Begriff auch kleingeschrieben. Die Standards benennen die Teile einzeln: Custom Elements sowie die Elemente `template` und `slot` im HTML-Standard, Shadow Trees im DOM-Standard. Der Sammelbegriff steht im Titel einer W3C Working Group Note vom 24. Juli 2014, *Introduction to Web Components*. Das W3C hat dieses Dokument inzwischen eingestellt und als nicht mehr zu referenzieren gekennzeichnet.

Ein Custom Element wird durch eine JavaScript-Klasse definiert, die `HTMLElement` erweitert und mit `customElements.define()` unter einem Namen registriert wird. Die Definition legt fest, wie der HTML-Parser das Element erzeugt und wie es auf Änderungen reagiert. Danach lässt es sich aus Markup, mit `document.createElement()` oder mit `new` erstellen. Sein Name beginnt mit einem kleinen ASCII-Buchstaben, enthält keine großen ASCII-Buchstaben und muss einen Bindestrich enthalten. Dieser trennt eigene Namen von der Sprache selbst: Laut HTML-Standard werden HTML, SVG und MathML keine neuen Elemente mit Bindestrich im Namen erhalten.

Es gibt zwei Arten. Ein autonomes Custom Element wie das standardeigene `<flag-icon country="nl">` beginnt als einfaches Element ohne eingebaute Bedeutung. Ein angepasstes eingebautes Element erweitert dagegen ein vorhandenes. Eine Klasse, die `HTMLButtonElement` erweitert, mit `{ extends: "button" }` registriert und als `<button is="plastic-button">` verwendet wird, behält alles, was einen Button ausmacht: Fokusverhalten, Formularübermittlung und das Attribut `disabled`.

Ein Shadow Tree ist ein gesonderter Knotenbaum, der an einem Element hängt. Dieses Element heißt Host; seinen gewöhnlichen Baum nennt der DOM-Standard Light Tree. `attachShadow()` erzeugt eine Shadow Root im offenen oder geschlossenen Modus. Nur im offenen Modus gibt die Eigenschaft `shadowRoot` des Hosts diese Wurzel zurück. Ein `slot` -Element im Shadow Tree markiert, wo die eigenen Kinder des Hosts erscheinen. Es zeigt die zugewiesenen Knoten oder, wenn keine vorhanden sind, seinen eigenen Inhalt. Ein benannter Slot nimmt Kinder auf, deren Attribut `slot` seinen Namen trägt. Das Element `template` enthält ein nicht gerendertes HTML-Fragment, das ein Skript klonen kann. Mit dem Attribut `shadowrootmode` stellt ein `template` eine deklarative Shadow Root dar. Ein Shadow Tree lässt sich deshalb auch direkt ins Markup schreiben.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was sind Web Components?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Oberflächen, die schnell laden, auf dem Smartphone funktionieren und sich in allen Browsern Ihrer Kunden gleich verhalten.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. Full-Stack-GIS-Kartenentwicklung geleitet — Full-Stack-Kartenentwicklung von der Datenbank bis zum Browser

Full-Stack-GIS-Kartenentwicklung geleitet Kontakt aufnehmen →

Warum es wichtig ist

Ein erfundenes Tag war früher wenig nützlich. Rückblickend bezeichnet der HTML-Standard solche Elemente als ungültig und von geringem praktischem Nutzen. Ein Custom Element ist dagegen ein vollwertiger Teil des Dokuments. Der Parser erzeugt es, DOM-Methoden erstellen es, und seine Klasse entscheidet, was bei Attributänderungen oder beim Einfügen in eine Seite und beim Entfernen daraus geschieht.

Die Definition muss nicht zuerst eintreffen. Ein vom Parser bereits erzeugtes Element wird zum Custom Element, sobald eine Definition für seinen Namen registriert ist. Der HTML-Standard nennt das Upgrading. Bis dahin gilt es als undefiniertes Element, ähnlich einem `span` : Text und Links darin erscheinen unverändert. Der Standard sagt ausdrücklich, dass Upgrades es erlauben, den Inhalt eines Custom Elements schrittweise zu erweitern.

Ein Custom Element kann auch an Formularen und am Barrierefreiheitsbaum teilnehmen. Eine Klasse, die `static formAssociated = true` setzt und `attachInternals()` aufruft, erhält ein `ElementInternals`-Objekt. Dessen `setFormValue()` gibt dem Element einen Wert, den sein Formular wie den jedes anderen Bedienelements übermittelt. Dasselbe Objekt legt eine voreingestellte ARIA-Rolle und ARIA-Zustände fest. Diese gelten, sofern der Seitenautor sie nicht durch das Attribut `role` oder die passenden `aria-*`-Attribute überschreibt. Es kann außerdem benutzerdefinierte Zustände enthalten, die Stylesheets mit der Pseudoklasse `:state()` auswählen.

Für ein Team sorgt der Shadow Tree dafür, dass eine Komponente in sich gekapselt ist: Gewöhnliche Seitenselektoren reichen nicht in ihren internen Shadow Tree hinein, und die Komponente legt fest, welche Teile von außen gestaltet werden dürfen. Geerbte Werte und Custom Properties können weiterhin hineingelangen. Über Slots angezeigter Inhalt aus dem Light Tree bleibt den Seitenselektoren zugänglich. Teilen sich mehrere Bibliotheken eine Seite, bietet der HTML-Standard lokal begrenzte Registries, erzeugt mit `new CustomElementRegistry()`. So können ihre Elemente nebeneinander bestehen, ohne dass die Bibliotheken ihre Namen abstimmen müssen.

Wie es in der Praxis aussieht

Eine Produktkarte zeigt das Zusammenspiel. Die Seite hält den Inhalt als gewöhnliches Markup vor. Das Element, das in einen benannten Slot gehört, erhält das Attribut `slot`:

```
<product-card>
  <span slot="name">Oak stool</span>
  <a href="/shop/oak-stool/">See the details</a>
</product-card>
<script type="module" src="product-card.mjs"></script>
```

Das Modul definiert das Element. Die Vorlage wird einmal aufgebaut, und jede Karte erhält eine eigene Kopie davon in einer offenen Shadow Root:

```
const template = document.createElement("template");
template.innerHTML = `
  <h2 part="title"><slot name="name">Unnamed product</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);
}
```

Die Seite gestaltet die Überschrift über den einen Teil, den die Komponente nach außen freigibt:

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

Bevor `product-card.mjs` ausgeführt wurde, ist die Karte ein undefiniertes Element. Leser sehen den Produktnamen und den Link als gewöhnlichen Inline-Inhalt. Beim Ausführen registriert `define()` die Klasse, und die bereits vorhandene Karte erhält ihr Upgrade. Ihr Konstruktor hängt eine Shadow Root an und füllt sie mit einem Klon des Vorlageninhalts. Das `span` wird dem Slot namens `name` zugewiesen und erscheint innerhalb der Überschrift. Der Link hat kein Attribut `slot` und landet deshalb im unbenannten Standardslot. Bei einer Karte ohne Namen erscheint stattdessen der eigene Inhalt des Slots, „Unnamed product“. Die Prüfung mit `customElements.get()` verhindert, dass eine zweite, über ein anderes Bundle eintreffende Modulkopie bei `define()` eine Ausnahme auslöst.

Gemeinsame Komponenten helfen nur, wenn sie verwendet werden. Das Portfolio dieser Website enthält einen Fall dazu. Das Designsystem für iOS 26 „Liquid Glass“ geht von einer Oberfläche aus, in der jeder neue Bildschirm Buttons, Badges und Karten neu erfindet, jeweils etwas anders. Jede Änderung betrifft dadurch fünf eigens gebaute Varianten. Die dort beschriebene Antwort ist ein verbindlicher Komponentensatz, aus dem jeder Bildschirm zusammengesetzt wird. Dazu gehören Card, Label, Button, GlassIconButton und DirectoryGrid. Lint-Regeln lehnen selbst gebaute Badges oder Chips ab und verweisen auf die verbindliche Komponente. Die Seite sagt nicht, ob diese Komponenten Custom Elements sind. Die Arbeitsweise lässt sich aber

übertragen: eine Definition je Komponente, durch Werkzeuge durchgesetzt statt durch das Gedächtnis eines Reviewers. Die Custom-Element-Registry erzwingt die erste Hälfte, indem sie eine zweite Definition für denselben Namen ablehnt.

Wo es schiefgeht

Ein Name mit Bindestrich ersetzt eine Rolle. „Ein autonomes Custom Element hat keine besondere Bedeutung: Es repräsentiert seine Kinder“, sagt der HTML-Standard. Sein Beispiel `taco-button` zeigt den Aufwand. Um als Button zu funktionieren, benötigt das Element `tabindex`, die ARIA-Rolle `button`, einen zugänglichen Namen, `aria-disabled`, Tastaturhandler, die Enter und Leertaste in Klicks umsetzen, und Stile für den deaktivierten Zustand. Selbst dann kann ein Custom Element kein voreingestelltes Fokusverhalten deklarieren. `tabindex="0"` entspricht deshalb möglicherweise nicht dem Buttonverhalten der Plattform. Baut das Element auf einem vorhandenen auf, verringert die Erweiterung eines eingebauten Elements laut Standard die Arbeit sowohl für seinen Autor als auch für seine Nutzer. Bei einem autonomen Element sollten die voreingestellte Rolle und die Zustände über `ElementInternals` festgelegt werden, statt dies jeder verwendenden Seite zu überlassen.

Der Konstruktor erledigt die eigentliche Arbeit. Der HTML-Standard stellt Regeln für Custom-Element-Konstruktoren auf. Zuerst kommt `super()` ohne Argumente. Der Konstruktor darf Attribute und Kinder des Elements nicht lesen, weil ein neu erzeugtes statt nachträglich aufgewertetes Element noch keine hat. Er darf dem Element auch keine eigenen Attribute oder Kinder hinzufügen. Ein Verstoß kann ein Element ergeben, das weder der Parser noch DOM-Methoden erzeugen können. In den Konstruktor gehören Anfangszustand, Voreinstellungen, Event Listener und die Shadow Root; Abruf und Rendering gehören in `connectedCallback`.

`connectedCallback` **soll nur einmal laufen.** Er kann mehrfach laufen. Standardmäßig löst das Verschieben eines Elements erst `disconnectedCallback` und dann `connectedCallback` aus. Das Element setzt seinen Zustand zurück, als wäre es entfernt und wieder eingefügt worden. Einmalige Einrichtung braucht ein Flag als Schutz. In `disconnectedCallback` sollte aufgeräumt werden, was `connectedCallback` gestartet hat. Soll ein Element beim Verschieben mit `moveBefore()` seinen Zustand behalten, wird `connectedMoveCallback` definiert. Schon dessen Vorhandensein ersetzt das übliche Paar, selbst bei leerem Funktionsrumpf.

Ein Name wird zweimal definiert. `customElements.define()` löst einen `NotSupportedError` aus, wenn der Name oder die Klasse bereits registriert ist, und einen `SyntaxError`, wenn der Name ungültig ist, etwa weil ihm ein Bindestrich fehlt oder er einen Großbuchstaben enthält. Vor der Definition sollte `customElements.get()` prüfen, ob sie bereits vorhanden ist. Eine Bibliothek, die neben anderen bestehen muss, sollte eine eigene lokal begrenzte Registry erhalten.

Der geschlossene Modus gilt als Schutz. Er verbirgt die Wurzel vor der Eigenschaft `shadowRoot` des Hosts. Für Event Listener außerhalb dieses geschlossenen Shadow Trees verbirgt er außerdem interne Knoten aus dem zusammengesetzten Ereignispfad. CSS Shadow Parts sagt ausdrücklich, dass Shadow Trees bewusst keine Sicherheitsgrenze bilden, sondern eine Hilfe für Seitenautoren sind. Geheimnisse gehören nicht in Komponenten. Der geschlossene Modus wird wegen seiner Wirkung auf äußeren Code gewählt, nicht als Schloss.

`attachShadow()` **wird aufgerufen, wo es nicht funktionieren kann.** Der DOM-Standard erlaubt eine Shadow Root nur auf Elementen mit einem gültigen Custom-Element-Namen und auf einer kurzen Liste eingebauter Elemente. Dazu gehören `div`, `section`, `p` und `span`, aber nicht `button`. Auf einem angepassten eingebauten Button löst der Aufruf einen `NotSupportedError` aus. Bei einem Element mit bereits vorhandener Wurzel verhält er sich zudem anders: Ist diese deklarativ und hat denselben Modus, leert der Aufruf sie und gibt sie leer zurück. Andernfalls löst er eine Ausnahme aus. Vorher sollte deshalb nach einer vorhandenen Shadow Root gesucht werden, damit eine vom Server im Markup gelieferte deklarative Wurzel erhalten bleibt, statt gelöscht zu werden.

Seitenstile sollen bis ins Innere reichen. CSS Shadow Parts nennt die Grenze eine Styling-Barriere und hält fest, dass sie eine Komponente zugleich schwerer gestaltbar macht, wenn die Seite das möchte. Elemente, die eine Seite gestalten darf, werden mit `part` freigegeben und von außen mit `::part()` angesprochen. Eine in einer anderen verschachtelte Komponente benötigt `exportparts` auf dem inneren Host, um ihre Teile bis zur Seite weiterzureichen. Farben und Abstände, die der Seite folgen sollen, werden aus Custom Properties auf dem Host gelesen. Sie erreichen den Shadow Tree durch Vererbung.

Der Inhalt existiert erst nach dem Skript. Eine Komponente, die ihren gesamten Text mit JavaScript schreibt, zeigt Lesern bis zum Laden der Definition nichts davon. Scheitert das Skript, erscheint überhaupt nichts. Der Inhalt sollte

wie bei der Produktkarte im Light Tree stehen und Slots zugewiesen werden. Das Upgrade erweitert dann, was bereits vorhanden ist. Muss Inhalt warten, verbirgt ihn das Beispiel des HTML-Standards, bis `customElements.whenDefined()` für jedes Element aufgelöst wurde, das `:not(:defined)` entspricht.

Weiterführende Lektüre

- HTML Standard: Custom elements — Namensregeln, Anforderungen an Konstruktoren, Upgrades, formulargebundene Elemente und `ElementInternals`, mit durchgearbeiteten Beispielen.
- DOM Standard: Shadow tree — Shadow Roots und ihre Hosts, Slots, die ihnen zugewiesenen Knoten und die Schritte hinter `attachShadow()`.
- HTML Standard: Scripting — das Element `template` mit seinem Attribut `shadowrootmode` und das Element `slot`.
- CSS Shadow Parts — der W3C-Arbeitsentwurf vom 16. Dezember 2025, der `part`, `exportparts` und das Pseudoelement `::part()` definiert.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was sind Web Components?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/web-components/