Develop modern HTML components with Web Components

A 20 Minutes Introduction

Overview

- HTML components and Web Components
- Core Technologies of Web Components

Web Components

- Custom Elements
- Shadow DOM
- Templates and Slots

Custom Elements

- Custom HTML tags created by developers.
- Extend existing elements or create entirely new ones.
- Lifecycles:
  - `connectedCallback()`: inserted into the DOM.
  - `disconnectedCallback()`: removed from the DOM.
  - `attributeChangedCallback()`: attributes change.

Ex 1: Basic Custom Element

```
class MyElement extends HTMLElement {
  connectedCallback() {
    this.innerHTML = "<p>Hello, World!</p>";
  }
}
customElements.define('my-element', MyElement);
```

```
<!-- Usage: -->
<my-element></my-element>
```

> Simple example of a custom element that adds "Hello, World!" to the DOM.

Ex 2: Custom Button with Attributes

```
class MyButton extends HTMLElement {
  constructor() {
    super();
    this.addEventListener('click', () => alert('Button clicked!'));
  }
  connectedCallback() {
    this.innerHTML = `<button>${this.getAttribute('label')}</button>`;
  }
}
customElements.define('my-button', MyButton);
```

```
<!-- Usage: -->
<my-button label="Click me"></my-button>
```

Ex 3: Customize Built-in Element

```
class ClickableParagraph extends HTMLParagraphElement {
  constructor() {
    super();
    this.addEventListener('click', () => alert('Paragraph clicked!'));
  }
  connectedCallback() {
    this.innerHTML = `Click me: ${this.getAttribute('content')}`;
    this.style.cursor = 'pointer'; // Make it clear that the element is clickable.
    this.style.color = 'blue'; // Add some style.
  }
}
customElements.define('clickable-p', ClickableParagraph, { extends: 'p' });
```

```
<!-- Usage: -->
<p is="clickable-p" content="Click me for something"></p>
```

Ex 5: Custom Modal Element

```
<!-- Usage: -->
<custom-modal id="myModal">
  <p slot="modal-text">This is your modal text.</p>
</custom-modal>
<button onclick="document.getElementById('myModal').open()">Open Modal</button>
```

```
class CustomModal extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.innerHTML = `
      <style>
        .modal {
          display: none;
          // redact because there is no space here.

        }
        .modal-content {
          background-color: white;
          // redact because there is no space here.
        }
      </style>
      <div class="modal">
        <div class="modal-content">
          <span slot="modal-text">
            This is not a modal text you're looking for.
          </span>
        </div>
      </div>
    `;
  }

  connectedCallback() {
    this.modal = this.shadowRoot.querySelector('.modal');
    this.modal.addEventListener('click', () => this.close());
  }

  open() {
    this.modal.style.display = 'block';
  }

  close() {
    this.modal.style.display = 'none';
  }
}

customElements.define('custom-modal', CustomModal);
```

Shadow DOM

- An encapsulated DOM subtree.
- Isolates styles and markup from the main document.
- Avoid CSS and JavaScript conflicts in complex web applications.
  - Also, make it harder to access Global CSS.

Ex : Basic Shadow DOM

```
class MyShadowElement extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.innerHTML = `
      <style> p { color: blue; } </style>
      <p>Hello from Shadow DOM</p>
    `;
  }
}
customElements.define('my-shadow-element', MyShadowElement);
```

```
<!-- Usage: -->
<my-shadow-element></my-shadow-element>
```

Declarative Shadow DOM

```
<my-shadow-element>
  <template shadowroot="open">
    <style>
      p { color: blue; }
    </style>
    <p>Hello from Declarative Shadow DOM</p>
  </template>
</my-shadow-element>
```

Shadow DOM: Pros and Cons

- Isolates component styles and DOM structure from the rest of the page.
- Reusable encapsulated components.
- Styles defined within a shadow root are isolated from the global scope.

- Developers must learn how to manage scoped styles and lifecycle events.
- Components in different shadow trees cannot easily communicate.

Ex : Custom Card Component

```
<!-- Usage: -->
<custom-card>
  <h2 slot="title">Card Title</h2>
  <p slot="content">Card content goes here.</p>
</custom-card>
```

```
class CustomCard extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.innerHTML = `
      <style>
        .card {
          border: 1px solid #ddd;
          padding: 20px;
          box-shadow: 2px 2px 10px rgba(0, 0, 0, 0.1);
        }
      </style>
      <div class="card">
        <slot name="title"> [object Object] </slot>
        <slot name="content"> [object Object] </slot>
      </div>
    `;
  }
}
customElements.define('custom-card', CustomCard);
```

Templates

- A way to define HTML chunks for later reuse.
- Templates are not rendered when the page loads, only when explicitly instantiated.

Templates Example

```
<template id="myTemplate">
  <p>This is content from the template.</p>
</template>
<!-- <template> is hidden and won't appear on the page initially. -->


<div id="contentArea"></div>
```

```
const template = document.getElementById('myTemplate');
const contentArea = document.getElementById('contentArea');

contentArea.appendChild(template.content.cloneNode(true));
```

Templates: Pros and Cons

- HTML structures defined once and reused multiple times.
- Templates are not displayed until needed, improving performance.

- Must be activated via JavaScript.
- Templates are static by nature.

Slots

- Mechanism for distributing content inside custom elements.
- Named and default slots for content distribution.

Advantages of Using Web Components

- ...

Advantages of Using Web Components

- Encapsulation of styles and behavior
- Reusability across frameworks (React, Vue, Angular)
- Can fit into most project structures

Where to go next

https://custom-elements-everywhere.com/

(google "github custom elements everywhere")

Q & A