Contents

1 Web Components

- main features: HTML imports, HTML template, custom elements, shadow DOM

2 Elements

2.1 Defining a new element

The `customElements global is used for defining a custom element and teaching the browser about a new tag.  Call =customElements.define()` with the tag name you want to create and a JavaScript `class` that extends the base `HTMLElement`.

- example: defining a mobile drawer panel, `<app-drawer>`

```
class AppDrawer extends HTMLElement {
  constructor() { // constructor arguments can also be defined
    super(); // always call super() first
    // click listener on <app-drawer> elemnt itself
    this.addEventListener('click', e => {
      if (this.disabled) {
        return;
      }
      this.toggleDrawer();
    });
  }

  // A getter/setter for on 'open' property
  get open() {
    return this.hasAttribute('open');
  }
  set open(val) {
    if (val) {
      this.setAttribute('open', ''); // refelect prop as an HTML attr
    } else {
      this.removeAttribute('open');
    }
  }
  get disabled() {
    return this.hasAttribute('disabled');
  }
  set disabled(val) {
    if (val) {
      this.setAttribute('disabled', ''); // refelect prop as an HTML attr
    } else {
      this.removeAttribute('disabled');
    }
  }

  toggleDrawer() {
    ...
  }
}

window.customElements.define('app-drawer', AppDrawer);
```

- the custom element created above can now be used just like native HTML elements i.e. `<app-drawer></app-drawer`
  - instances of it can be declared on the page, created dynamically via JS, event listeners can be attached an so on
- `this` inside a class definition refers to the DOM itself
  - the entire DOM API is available inside the element code for example `this.children` to inspect its direct children or `this.querySelectorAll('.items')` to query nested nodes

2.1.1 Naming rules

- names of custom elements must contain a dash "-"
- the same name can only be registered once
- custom elements cannot be self-closing

2.2 Custom element reactions

A custom element can define special lifecycle hooks for running code during interesting times of its existence, these are called custom element reactions

Name	Called when
constructor	instance of the element is created or upgraded; useful for initializing state, setting up event listeners or creating a shadow dom
connectedCallback	called everytime the element is inserted into the DOM; useful for running setup code, such as fetching resources or rendering
disconnectedCallback	called everytime the element is removed from the DOM
attributeChangedCallback(attrName, oldVal, newVal)	called when an observed attribute has been added, removed, updated or replaced; also called for initial values when an element is created/upgraded; only attributes listed in the observerdAttributes property will receive this callback
adoptedCallback	the custom element has been moved into a new document

- to the above example `static get observedAttributes() { return ['disabled', 'open']}` needs to be added to the class to have `attributeChangedCallback` called for changes in those attributes

2.3 Creating an element that uses Shadow DOM

The Shadow DOM provides a way for an element to own, render and style a chunk of DOM that's separate from the rest of the page. You could for example hide away an entire within a single tag:

```
// chat app's implementation details are hidden away in Shadow DOM
<chat-app></chat-app>
```

To use Shadow DOM in a custom element, call `this.attachShadow` inside the constructor:

```
// Create template in js
let tmpl = document.createElement('template');
tmpl.innerHTML = `
  <style>:host { ... }</style> <!-- look ma, scoped styles -->
  <b>I'm in shadow dom!</b>
  <slot></slot>
`;
// or via HTML template tag
// <template id="shopping-template">
//    <b>I'm in shadow dom</b>
//    <slot></slot>
// </template>

customElements.define('x-foo-shadowdom', class extends HTMLElement {
  constructor() {
    super(); // always call super() first in the constructor.

    // Attach a shadow root to the element.
    let shadowRoot = this.attachShadow({mode: 'open'});
    shadowRoot.appendChild(tmpl.content.cloneNode(true));
  }
  ...
});
```

Example usage:

```
<x-foo-shadowdom>
  <p><b>User's</b> custom text</p>
</x-foo-shadowdom>

<!-- renders as -->
<x-foo-shadowdom>
  #shadow-root
    <b>I'm in shadow dom!</b>
    <slot></slot> <!-- slotted content appears here -->
</x-foo-shadowdom>

<br>
<h2>This is the footer.</h2>
<p>You can put stuff here.</p>
<br>
```