HTML5 Web Components

Key Concepts of Web Components

1. Custom Elements

Custom Elements allow you to define new HTML tags, enhance existing ones, or extend the functionalities of built-in HTML elements.

Creating a Custom Element

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Custom Element Example</title>
</head>
<body>
  <!-- Use the custom element -->
  <hello-world></hello-world>

  <script>
    class HelloWorld extends HTMLElement {
      constructor() {
        super();
        this.attachShadow({ mode: 'open' });
        const span = document.createElement('span');
        span.textContent = 'Hello, World!';
        this.shadowRoot.append(span);
      }

      connectedCallback() {
        console.log('HelloWorld element added to page.');
      }

      disconnectedCallback() {
        console.log('HelloWorld element removed from page.');
      }

      attributeChangedCallback(name, oldValue, newValue) {
        console.log(`Attribute: ${name} changed from ${oldValue} to ${newValue}`);
      }
      static get observedAttributes() {
        return ['data-text'];
      }

      get text() {
        return this.getAttribute('data-text');
      }

      set text(value) {
        this.setAttribute('data-text', value);
      }
    }

    customElements.define('hello-world', HelloWorld);
  </script>
</body>
</html>
```

2. Shadow DOM

The Shadow DOM provides encapsulation for DOM and CSS. It allows you to keep the markup structure, style, and behavior separate and encapsulated from the rest of the document.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Shadow DOM Example</title>
</head>
<body>
  <my-component></my-component>

  <script>
    class MyComponent extends HTMLElement {
      constructor() {
        super();
        this.attachShadow({ mode: 'open' });
        this.shadowRoot.innerHTML = `
          <style>
            p {
              color: blue;
            }
      </style>
      <p>This is inside the shadow DOM!</p>
     `;
    }
  }

  customElements.define('my-component', MyComponent);
 </script>
</body>
</html>
```

3. HTML Templates

HTML templates are inert pieces of DOM that can be cloned and added to the document. They provide a way to define markup that will not be rendered until explicitly added to the DOM.

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>HTML Template Example</title>
</head>
<body>
 <template id="my-template">
  <style>
   p {
    color: green;
   }
  </style>
  <p>This is a paragraph inside a template.</p>
 </template>

 <script>
  class MyTemplateElement extends HTMLElement {
   constructor() {
    super();
    const template = document.getElementById('my-template');
    const templateContent = template.content.cloneNode(true);
    this.attachShadow({ mode: 'open' }).appendChild(templateContent);
   }
  }

  customElements.define('my-template-element', MyTemplateElement);
 </script>

 <!-- Use the custom element -->
 <my-template-element></my-template-element>
</body>
</html>
```

Combining Technologies

You can combine these technologies to create powerful, encapsulated, and reusable components. Here’s a more comprehensive example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Web Components Example</title>
</head>
<body>
  <user-card name="John Doe" age="30"></user-card>

  <template id="user-card-template">
    <style>
      .card {
        border: 1px solid #ccc;
        padding: 16px;
        border-radius: 8px;
        max-width: 200px;
      }
      .name {
        font-weight: bold;
      }
      .age {
        color: gray;
      }
    </style>
    <div class="card">
      <div class="name"></div>
      <div class="age"></div>
    </div>
  </template>

  <script>
    class UserCard extends HTMLElement {
      constructor() {
        super();
        const template = document.getElementById('user-card-template').content;
        const shadowRoot = this.attachShadow({ mode: 'open'
}).appendChild(template.cloneNode(true));
      }

      static get observedAttributes() {
        return ['name', 'age'];
      }

      attributeChangedCallback(name, oldValue, newValue) {
        this._updateRendering();
      }

      connectedCallback() {
        this._updateRendering();
      }

      _updateRendering() {
        if (this.shadowRoot) {
          this.shadowRoot.querySelector('.name').textContent = this.getAttribute('name');
          this.shadowRoot.querySelector('.age').textContent = `Age: ${this.getAttribute('age')}`;
        }
      }
    }

    customElements.define('user-card', UserCard);
  </script>
</body>
</html>
```

Best Practices

1. Encapsulation: Use Shadow DOM to encapsulate styles and behaviors.

2. Reusability: Create custom elements that can be reused across different projects.

3. Accessibility: Ensure your custom elements are accessible (use ARIA roles and properties if necessary).

4. Performance: Optimize for performance (e.g., lazy loading of components).

Further Reading and Resources

[MDN Web Components Guide](https://developer.mozilla.org/en-US/docs/Web/Web_Components)

[Web Components Specification](https://www.w3.org/TR/webcomponents/)

[LitElement Documentation](https://lit.dev/)

[Polymer Project](https://polymer-library.polymer-project.org/)