



E-Portfolio

Polymer Project

Felix Hausberger, SAP
02/04/2019

INTERNAL

Agenda

1. What Are Web Components
2. Why Web Components
3. Polymer Project
4. Lit-HTML Templating Library
5. Lit-Element Base Class
6. The Future Of Web Development And Polymer
7. Let's Build Our Own Web Components!



What Are **Web Components**?

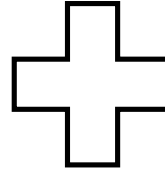


Set Of Web Platform APIs

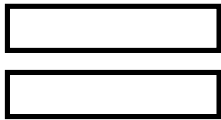


- Web Platform APIs: Global JavaScript objects available in most browsers
- Specified by World Wide Web Consortium (W3C)
- Custom, reusable, encapsulated HTML tags
- Based on four specifications
 1. Custom Elements
 2. Shadow DOM
 3. ES6 Modules
 4. HTML Templates


```
<header>
  <h1><slot></slot></h1>
  <button>Menu</button>
</header>
```



```
<my-header>Shadow DOM</my-header>
```



```
<my-header>
  <header>
    <h1>Shadow DOM</h1>
    <button>Menu</button>
  </header>
</my-header>
```

Lifecycle Callbacks

- `connectedCallback`
- `disconnectedCallback`
- `adoptedCallback`
- `attributeChangedCallback`



Why Web Components?





Why Web Components?

1. Fully featured and conform
2. Encapsulated
3. Reusable and modular
4. Maintainability
5. Brand consistency
6. Framework independent



The Polymer Project



The Polymer Project



- Open-source JavaScript library for building PWAs using web components
- Developed by a team of 29 Google developers
- Used by redesigned YouTube, Netflix, IBM, ...
- 3-Clause BSD license
- Main libraries:
 1. Lit-html
 2. Lit-element
 3. PWA-starter-kit



Lit-HTML Templating Library



- Based on tagged template literals with embedded JavaScript expressions
- Express web UI as a function of data and state
- Efficient
 - Uses web platform features like html templates
 - Only re-renders updated parts of view
- Expressive
 - Use full power of JavaScript in expressions
 - Templates can be computed and passed to and from functions
 - Text content binding can contain strings, DOM nodes, nested templates, arrays and more
- Extensible/Customizable
 - Take use of several lit-html directives
 - Customizable to create own template construction kit

General Usage:

```
const myTemplate = (name) => html`<p>Hello ${name}</p>`; //Returns a TemplateResult, lit-html is lazily rendered
render(myTemplate('World'), document.body); //Replaces all content of body node
```

Attribute Binding:

```
const myTemplate2 = (data) => html`<div ?disabled=${!data.active}>Stylish text</div>`;
render(myTemplate2({active: false}), document.body);
```

Property Binding:

```
const myTemplate3 = (data) => html`<my-list .listItems=${data.items}></my-list>`;
render(myTemplate3([1,2,3]), document.body);
```

Event Handling:

```
const myTemplate4 = () => html`<button @click=${clickHandler}>Click Me!</button>`;
render(myTemplate4(() => console.log("clicked!")), document.body);
```

Composed Templates:

```
const myTemplate5 = html`<h1>Header</h1>`;
const myTemplate6 = html`${myTemplate5} <div>Here's my main page.</div>`;
render(myTemplate6(), document.body);
```


Conditional Statements:

```
html`${user.isLoggedIn ? html`Welcome ${user.name}` : html`Please log in`}`;  
html`${() => {  
  if (user.isLoggedIn)  
    return html`Welcome ${user.name}`;  
  return html`Please log in`;  
}}`;
```

Iterative Statements:

```
const items = [1,2,3];  
const itemTemplates = [];  
  
html`<ul>${items.map((item) => html`<li>${item}</li>`)}</ul>`;  
  
items.forEach((item) => {itemTemplates.push(html`<li>${item}</li>`)});  
html`<ul>${itemTemplates}</ul>`;  
  
html`<ul>${repeat(items, (item) => item, (item, index) => html`<li>${index}: ${item}</li>`)}</ul>`;
```

Styling Templates:

```
const highlightStyles = { color: 'white', backgroundColor: 'red'};
html`<div style=${styleMap(highlightStyles)}>Hi there!</div>`;

const classes = { highlight: true, enabled: true, hidden: false };
html`<div class=${classMap(classes)}>Classy text</div>`;
```

Directives:

```
const image = {alt: "description"};
html``;

const content = fetch('./content.txt').then(r => r.text());
html`${until(content, html`<span>Loading...</span>`)};

const immutableItems = ["Are", "you", "still", "listening?"];
html`<div>${guard([immutableItems], () => immutableItems.map(item => html`${item}`))}</div>`;
```



Lit-Element Base Class

- Create

- De

- UI

- Fas

- Se

- sta

```
import { LitElement, html, css } from 'lit-element';
class HelloWorld extends LitElement {
  render() {
    return html`<p>${this.message.map(item => html`${item} `)}</p>`;
  }
  static get styles() {
    return [super.styles, css`
      p {
        font-family: Roboto;
        font-size: 30px;
        font-weight: 500;
      }
    `];
  }
  static get properties() {
    return { name: { type: String },
      message: { type: Array } };
  }
  constructor() {
    super();
    this.name = 'Felix';
    this.message = ['Hello', this.name, '!'];
  }
}
window.customElements.define('hello-world', HelloWorld);
```

nts



Lit-Element Styling

- Static styles property
 - Constructable style sheets
 - Return a single or an array of tagged template literals
- `<style>` element in HTML template in the render function
- External stylesheet linked to from your HTML template
- Use `:host` or `:host()` pseudo class to refer to host element of shadow root
- Use `::slotted()` pseudo class to refer to all light DOM children, that are inserted to the shadow DOM by a slot element
- Element selector from light DOM has priority over `:host` selector
- Inherited CSS properties and custom CSS properties are inherited through shadow root boundaries

Excursus: CSS custom properties

```
<style>
  html {
    --myBackground: rgb(67, 156, 144);
  }
</style>
```

```
return [super.styles, css`
  :host {
    display: block;
    background-color: var(--myBackground, yellow);
  }
`];
```

➔ Implement theming



Lit-Element Properties

- Property options
 - **type**: Use Lit-Element's default attribute converter
 - **noAccessor**: Whether to set up a default property accessor
 - **hasChanged**: Specify what constitutes a property change
 - **converter**: Convert between properties and attributes
 - **attribute**: Configure observed attributes
 - **reflect**: Configure reflected attributes
- Properties declared in *static get properties*
- Properties initialized in constructor
- Properties can set through markup attributes



Conversion needed

Lit-Element Type Custom Converter

```
prop1: { type: Number,  
  converter: {  
    fromAttribute: (value, type) => {  
      // `value` is a string  
      // Convert it to a value of type `type` and return it  
      return type(value);  
    },  
    toAttribute: (value, type) => {  
      // `value` is of type `type`  
      // Convert it to a string and return it  
      return value.toString();  
    }  
  },  
},  
prop2: { type: Object,  
  converter: (value, type) => {  
    // `value` is a string  
    // Convert it to a value of type `type` and return it  
    return type(value);  
  }  
}
```




Lit-Element Properties

- Lit-Element creates for all properties an observed attribute (name is lowercased property name)
- Use *attribute* property option to define an own name
- Set *attribute* property option to false if it should not be observed
- **Accessors:**
 - LitElement generates a property accessor for all declared properties
 - Use `this.<accessor name>` to invoke accessor

```
set prop1(value) {  
  const oldValue = this.prop1;  
  console.log("Changed property to " + value);  
  // Implement setter logic here...  
  this.requestUpdate('myProp', oldValue);  
}  
get prop1() {  
  
}
```




Lit-Element Properties

- hasChanged:
 - Specifies whether *update()* should be called or not

```
prop1: { type: Number,  
  hasChanged(newVal, oldVal) {  
    return newVal > oldVal;  
  },
```



Lit-Element Events

- Where to set event handlers?
 - `@event` binding in HTML template
 - `.addEventListener` in constructor
 - `firstUpdated` lifecycle callback
 - `connectedCallback` lifecycle callback
- Custom events bubble through shadow DOM boundaries if *bubble* and *composed* flag is set to true
- Event targets are set to the custom element
- Actual event origin can be found in *composedPath* property



Lit-Element Lifecycle

1. `someProperty.hasChanged`
2. `requestUpdate`
3. `performUpdate`
4. `shouldUpdate`
5. `update`
6. `render`
7. `firstUpdated`
8. `updated`
9. `updateComplete`



Update lifecycle only in response to a property change

Use a web component



```
<head>  
  <script src="./path-to/custom-elements-es5-loader.js"></script>  
  <script src="path-to/webcomponents-loader.js" defer></script>  
  <script type="module">  
    window.WebComponents = window.WebComponents || {  
      waitFor(cb){ addEventListener('WebComponentsReady', cb) }  
    }  
  
    WebComponents.waitFor(async () => {  
      import('./path-to/some-element.js');  
    });  
  </script>  
</head>  
<body>  
  <some-element></some-element>  
</body>
```


Use TypeScript decorators

```
import { LitElement, html, customElement, property } from 'lit-element';

@customElement('my-element')
export class MyElement extends LitElement {
  @property({type : String}) prop1 = 'Hello World';
  @property({type : Array}) prop2 = [1,2,3];
  @property({type : Object}) prop3= { subprop: 'prop 3 subprop value' };

  render() {
    return html`
      <p>prop1: ${this.prop1}</p>
      <p>prop2[0]:</p>${this.prop2[0]}</p>
      <p>prop3.subprop: ${this.prop3.subprop}</p>
    `;
  }
}

customElements.define('my-element', MyElement);
```


Future Of Web Development And Polymer





Future Of Web Development And Polymer

- Check out the [PWA starter kit!](#)

Applications

Frameworks

Web Components

Web Platform

References

Web components:

[Web Components Introduction](#)

[Web Components Specification](#)

[Web Components Usage](#)

[Web Components Best Practices](#)

Polymer:

[Polymer Project](#)

[Polymer Comparison To Other Frameworks](#)

Get your components!

[WebComponents](#)

[Vaadin](#)

[Material Web Components](#)

[UI5 Web Components](#)

[CustomElements.IO](#)

Thank You.

Contact Information

Felix Hausberger

Student of Applied Computer Science

Felix.Hausberger@sap.com

GitHub:

<https://github.com/fidsusj/>

SAP folgen auf



www.sap.com/contactsap

© 2018 SAP SE oder ein SAP-Konzernunternehmen. Alle Rechte vorbehalten.

Weitergabe und Vervielfältigung dieser Publikation oder von Teilen daraus sind, zu welchem Zweck und in welcher Form auch immer, ohne die ausdrückliche schriftliche Genehmigung durch SAP SE oder ein SAP-Konzernunternehmen nicht gestattet.

In dieser Publikation enthaltene Informationen können ohne vorherige Ankündigung geändert werden. Die von SAP SE oder deren Vertriebsfirmen angebotenen Softwareprodukte können Softwarekomponenten auch anderer Softwarehersteller enthalten. Produkte können länderspezifische Unterschiede aufweisen.

Die vorliegenden Unterlagen werden von der SAP SE oder einem SAP-Konzernunternehmen bereitgestellt und dienen ausschließlich zu Informationszwecken. Die SAP SE oder ihre Konzernunternehmen übernehmen keinerlei Haftung oder Gewährleistung für Fehler oder Unvollständigkeiten in dieser Publikation. Die SAP SE oder ein SAP-Konzernunternehmen steht lediglich für Produkte und Dienstleistungen nach der Maßgabe ein, die in der Vereinbarung über die jeweiligen Produkte und Dienstleistungen ausdrücklich geregelt ist. Keine der hierin enthaltenen Informationen ist als zusätzliche Garantie zu interpretieren.

Insbesondere sind die SAP SE oder ihre Konzernunternehmen in keiner Weise verpflichtet, in dieser Publikation oder einer zugehörigen Präsentation dargestellte Geschäftsabläufe zu verfolgen oder hierin wiedergegebene Funktionen zu entwickeln oder zu veröffentlichen. Diese Publikation oder eine zugehörige Präsentation, die Strategie und etwaige künftige Entwicklungen, Produkte und/oder Plattformen der SAP SE oder ihrer Konzernunternehmen können von der SAP SE oder ihren Konzernunternehmen jederzeit und ohne Angabe von Gründen unangekündigt geändert werden. Die in dieser Publikation enthaltenen Informationen stellen keine Zusage, kein Versprechen und keine rechtliche Verpflichtung zur Lieferung von Material, Code oder Funktionen dar. Sämtliche vorausschauenden Aussagen unterliegen unterschiedlichen Risiken und Unsicherheiten, durch die die tatsächlichen Ergebnisse von den Erwartungen abweichen können. Dem Leser wird empfohlen, diesen vorausschauenden Aussagen kein übertriebenes Vertrauen zu schenken und sich bei Kaufentscheidungen nicht auf sie zu stützen.

SAP und andere in diesem Dokument erwähnte Produkte und Dienstleistungen von SAP sowie die dazugehörigen Logos sind Marken oder eingetragene Marken der SAP SE (oder von einem SAP-Konzernunternehmen) in Deutschland und verschiedenen anderen Ländern weltweit. Alle anderen Namen von Produkten und Dienstleistungen sind Marken der jeweiligen Firmen.

Zusätzliche Informationen zur Marke und Vermerke finden Sie auf der Seite <https://www.sap.com/corporate/de/legal/copyright.html>.