Web Compontents

2017-07-14 kamijin-fanta

IT Create Club

Web Componentsとは

- ウェブページ・ウェブアプリケーションのためのAPI群
- 再利用可能で、カプセル化されたコンポーネントを提供する
- 4大要素
  - Custom Elements: カスタムタグを定義することが出来る
  - HTML Template: 実行時に動的に構築するHTMLを提供
  - Shadow DOM: カプセル化されたスタイル・マークアップを提供
  - HTML imports: HTML文書の再利用性の規定

https://www.webcomponents.org/introduction

発展途上の技術

	Chrome	Firefox	IE 11+/ Edge	Opera	Safari 9+	Chrome (Android)	Safari (iOS 9+)
Template	Native	Native	Partial	Native	Native	Native	Native
HTML Imports	Native	Polyfill	Polyfill	Native	Polyfill	Native	Polyfill
Custom Elements	Native	Polyfill	Polyfill	Native	Partial	Native	Partial
Shadow DOM	Native	Polyfill	Polyfill	Native	Partial	Native	Partial

Notes:

- Templates are supported in Edge, but not IE.
- Safari supports custom elements starting in 10.3.
- Safari supports shadow DOM starting in 10.2, but as of 10.3 there are still some known issues.
- Older versions of the Android Browser may run into some issues - please file an issue if you run into a problem on this browser. Chrome for Android is supported.

https://www.polymer-project.org/2.0/docs/browsers

- 今回は生APIを触って覚える
- 幸いChromeが割りと対応している
- Polyfill(JSでの先行実装的な)の実装なのか本来の仕様なのか分からなくなるの嫌

1. Custom Elements - 概要

> **カスタム要素** は、カスタムである要素である

- 開発者がカスタムタグを自由に定義することが出来る
  i. HTMLElementを継承して1からエレメントを作る
  ii. 既存のビルドインエレメントを拡張する
- カスタムエレメントの名前はハイフンを使用する
  - NG: `customelement` / OK: `custom-element`
  - SVGで使用されているハイフンを含む要素名はダメ
    - `annotation-xml` `font-face`
- 文書のツリー上に要素が追加されることを `接続(connect)` と言う
- HTML Standardで規定

https://triple-underscore.github.io/HTML-custom-ja.html

1. Custom Elements - コード例

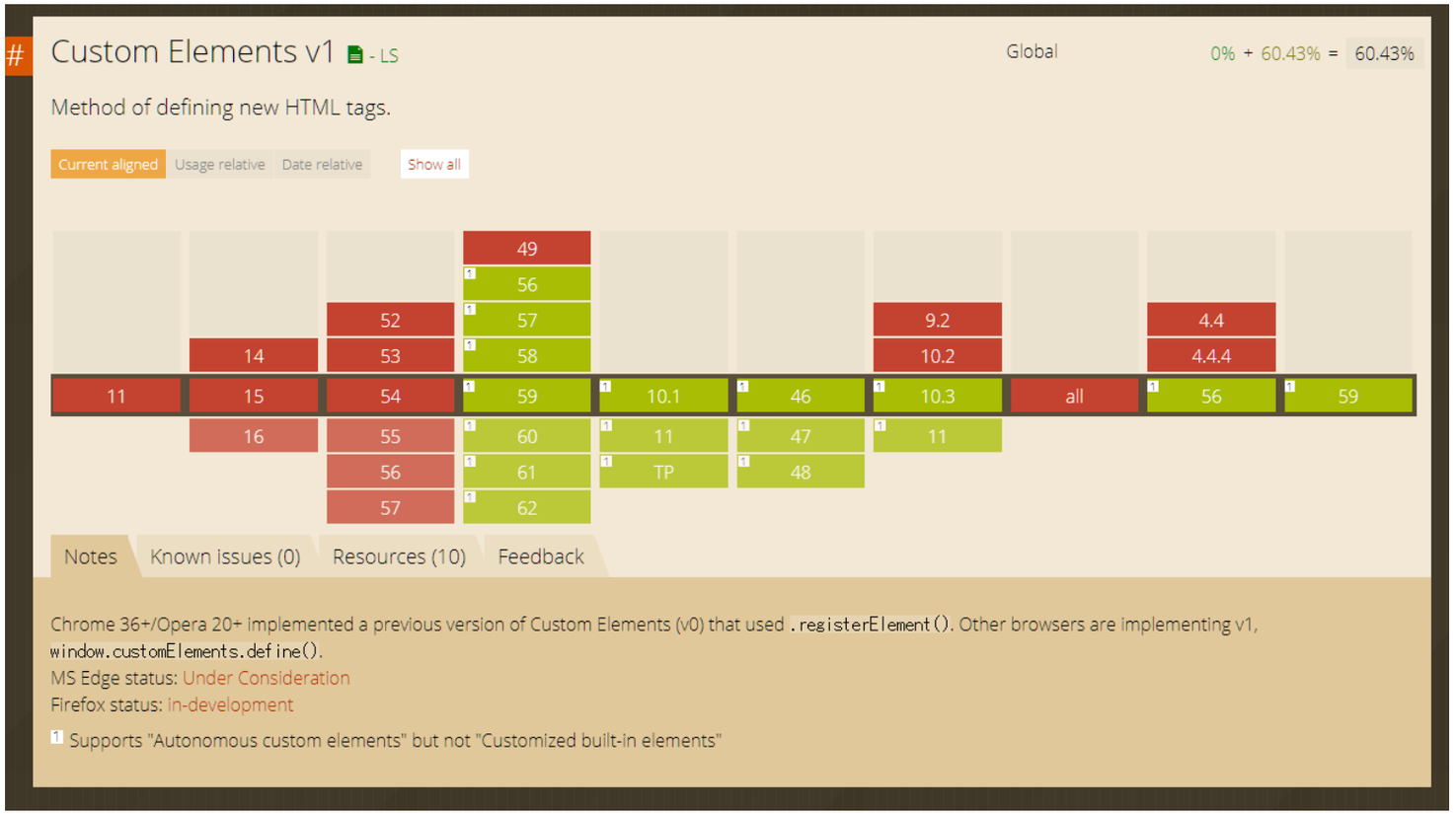
```
<script>
class SampleElement extends HTMLElement {
  // 何もしない
}
window.customElements.define('sample-element', SampleElement);
</script>

<sample-element />
```

```
<script>
class SampleElement extends HTMLElement {
  connectedCallback() {  // 接続後にテキストを変更
    this.textContent = 'Hello Component';
  },
}
window.customElements.define('sample-element', SampleElement);
</script>

<sample-element />
```

1. Custom Elements - ビルドインエレメントの拡張



> Supports "Autonomous custom elements" but not "Customized built-in elements"

1. Custom Elements - 重要そうな所

- バージョン
  - Custom Elements v0
    - `document.registerElement()`
  - Custom Elements v1
    - `window.customElements.define()`
- ビルドインエレメントの拡張
  - アクセシビリティを保ったまま、既存のエレメントを拡張できる
  - でも、そんなこと実際にするのか…？　という声も

2. HTML Template - 概要

- JSからクローン・挿入できる、HTMLのフラグメントを提供
  - 懐かしの `jquery-tmpl` っぽさ
  - `element.content` でテンプレートの内容を返す
- template内部のDOMは、HTMLロード時には実行されない
  - 構文が有っていれば、属性などが適合する必要は無い
- HTML Standardで規定

```
<template>
  <img src="{{src}}" alt="{{alt}}">
</template>
```

https://triple-underscore.github.io/HTML-scripting-ja.html#the-template-element

2. HTML Template - コード例

```
<template id="twitter-template">
  <h3 class="screen_name"></h3>
  <div class="description"></div>
</template>

<div id="container"></div>

<script>
  var data = [
    { screen_name: 'twitter', description: 'official account' },
    { screen_name: 'scala_lang', description: 'The Scala Programming Language.' },
  ]
  var template = document.querySelector('#twitter-template').content;

  for (var account of data) {
    var clone = template.cloneNode(true);
    for (var key in account) {
      clone.querySelector(`.${key}`).textContent = account[key];
    }
    document.querySelector('#container').appendChild(clone);
  }
</script>
```

2. HTML Template - 重要そうな所

- テンプレートのノードを複製する方法は2種類ある
  - `document.importNode(templateContent, true)`
  - `templateContent.cloneNode(true)`
- 引数のtrueは両方共deepCopyするかの指定
- 2つの違いはノード内の文章を更新するタイミング
  - `document.importNode()`：クローンするタイミング
  - `cloneNode()`：appendChild()するタイミング

3. Shadow DOM - 概要

- ホストに接続された、別のノードツリー(ShadowTree)を提供する
  - スタイル・イベントなどを分離する
  - RootDocument/別ShadowTreeとのID重複が許可
- slotと呼ばれる要素が含まれる事が有る
- DOM Standardで規定

https://triple-underscore.github.io/DOM4-ja.html#shadow-trees

3. Shadow DOM - ツリー

通常

```
<my-header>
  <header>
    <h1>
    <button>
```

ShadowDOMを使用した場合

```
<my-header>
  #shadow-root
    <header>
      <h1>
      <button>
```

RootDocument側からh1,buttonのスタイルを変更できない

3. Shadow DOM - コード例

```
<template id="sample-template">
  <style> p { color: orange; } </style>
  <p>pセレクタでスタイルを設定しているけど、影響するのはここだけ</p>
</template>
<script>
class ShadowDomSample extends HTMLElement {
  connectedCallback () {
    var template = document.querySelector('#sample-template');
    var clone = document.importNode(template.content, true);
    var shadow = this.attachShadow({mode: 'open'});
    shadow.appendChild(clone);
  }
}
window.customElements.define('shadow-dom-sample', ShadowDomSample);
</script>

<shadow-dom-sample></shadow-dom-sample>
<p>Shadow DOMの外</p>
```

3. Shadow DOM - 実行結果

```
▼<shadow-dom-sample>
  ▼#shadow-root (open)
      <style>
            p { color: orange; }
          </style>
      <p>pセレクタでスタイルを設定しているけど、影響するのはここだけ</p>
  </shadow-dom-sample>
  <p>Shadow DOMの外</p>
```

3. Shadow DOM - slotを使用した合成

```
<template id="slot-template">
  <h3>Title: <slot name="title"></slot></h3>
  <div>
    <slot name="content">default message</slot>
  </div>
</template>

<shadow-dom-slot>
  <span slot="title"><i>Italic</i> Title</span>
</shadow-dom-slot>
```

```
▼<shadow-dom-slot>
  ▼#shadow-root (open)
    ▼<h3>
       "Title: "
      ▼<slot name="title">
         <span>
       </slot>
     </h3>
    ▼<div>
      ▼<slot name="content">
         "default message"
         #text
       </slot>
     </div>
  ▼<span slot="title">
     <i>Italic</i>
     " Title"
   </span>
 </shadow-dom-slot>
```

3. Shadow DOM - スタイル

- ShadowTreeの内外のスタイルは限定的にしか干渉しあわない
  - セレクタは分離されている ex: `*`

3. Shadow DOM - スタイル 例外1

- 継承可能な `font` `color` はプロパティは例外的に中に伝達

```
▼<style>
        .demo.styling {
          color: white;
          font-family: "ヒラギノ明朝 ProN W6", "HiraMinProN-W6", "HG明朝E", "ＭＳ Ｐ明朝", "MS PMincho", "MS 明朝", serif;
          font-weight: bold;
        }

  </style>
▼<div class="demo styling">
  ▼<shadow-dom-styling>
    ▼#shadow-root (open)
      <style>
            div {
              background: #333;
            }
          </style>
      <div>hogehoge</div>
    </shadow-dom-styling>
```

3. Shadow DOM - スタイル 例外2

- `:host` `:host(selector)`
  - Shadow Rootのホストに対して、スタイルを適用する

```
#shadow-root
  <style>
    /* custom elements default to display: inline */
    :host {
      display: block;
    }
    /* set a special background when the host element
       has the .warning class */
    :host(.warning) {
      background-color: red;
    }
  </style>
```

3. Shadow DOM - スタイル 例外3

- 変数は、ShadowTreeの外にあるものが使用できる
- インタフェースのように、変更できるプロパティを公開する

```
/* Inside shadow tree */
:host {
  background-color: var(--my-theme-color);
  @apply --my-custom-mixin;
}

/* root document tree */
html {
  --my-theme-color: red;
  --my-custom-mixin: {
    color: white;
    background-color: blue;
  }
}
```

3. Shadow DOM - 重要そうな所

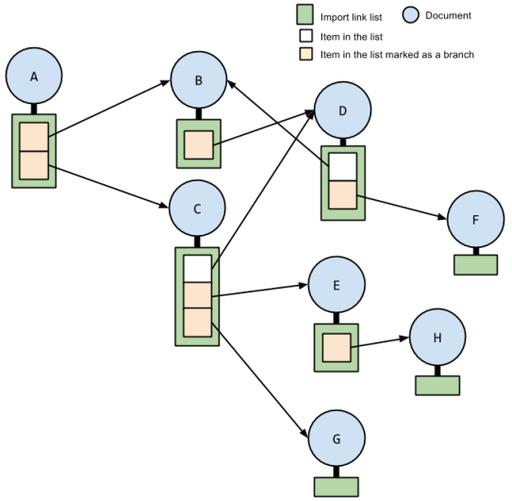
- パフォーマンス
  - 他ドキュメントとのid重複が許可→id多用するべき
  - 外部セレクタや他のエレメントの状況を計算する必要が無く高速
- shadowRootの `mode`　デフォルト値は無い (Apple vs Google戦争の結末)
  - `open`：親エレメントから、 `element.shadowRoot` で内部を参照できる
  - `closed`： `element.shadowRoot` は常にnullとなる
- `Element.createShadowRoot` の作成方法は非推奨
- slotは多段もサポートされている
- 通常ShadowRootをイベントが越えて伝達されることはない
  - CustomEventでフラグを指定すると可能になる

```
// composed フラグを設定して、ShadowRootの境界を超えるイベント作成
new CustomEvent('my-event', {bubbles: true, composed: true});
```

https://www.polymer-project.org/2.0/docs/devguide/shadow-dom

4. HTML imports - 概要

- コンポーネントを小さなHTMLに分割し、ブラウザで依存関係の解決を行う
- これまでのコンテンツを取り込む方法
  - ifream: コンテンツのサイズの決定を行いにくい・重い
  - XHR/fetch: JSでの読み込み制御が面倒・難しい



https://w3c.github.io/webcomponents/spec/imports/

4. HTML imports - コミュニティの流れ

- Mozillaが実装する気がない
  - Firefox OSで1年以上コンポーネントやってたけど、AMD,Commonjsで依存解決すれば良い
  - polyfillで頑張れよ
  - スコープがぶつかる場合の考慮は？
- Appleが `Isolated-Imports` を提案
  - コンポーネントはJSで直接グローバルオブジェクトにアクセス出来ない
  - Mozillaも支持？
  - クロスオリジンでの安全なインポートを実現
  - Shadow DOMのCloseも、隔離を実現したいから提案した
  - `document.exportElements("custom-xbox", ...)` としてエクスポートする

https://hacks.mozilla.org/2015/06/the-state-of-web-components/

https://github.com/w3c/webcomponents/blob/gh-pages/proposals/Isolated-Imports-Proposal.md

Web Component

- 生で触るAPIかと言われるとそうじゃない気がする
- 比較的新しいAPIなのに、歴史的経緯がガツガツ乗ってて楽しい

周辺技術

- Polymer
  - Google
  - WebComponents関連APIをラップし、使いやすくしている
  - 適当にPolyfillも当てながら非対応ブラウザに対応
  - Polymer独自機能
    - Data System: 要素・プロパティへのバインディング
    - Event: ShadowDOMの子要素へのイベントハンドルを提供
- Vuejs
  - Web Componentの思想をベースとしたフレームワーク
  - HTML Import相当をCommonjsとして扱い、バンドルする
  - https://jp.vuejs.org/v2/guide/comparison.html#Polymer
- AngaulIr
  - Google
  - 割りとWeb標準を意識してる
  - Polymerとの併用も意識

参考資料

- 歴史・仕様とか
  - 仕様書は基本WHATWGのLiving Standardを見るのが良い
    - https://html.spec.whatwg.org/multipage/
    - https://triple-underscore.github.io/DOM4-ja.html
    - https://momdo.github.io/html/
  - https://w3c.github.io/webcomponents/spec/imports/
  - Mozilla `The state of Web Components` https://hacks.mozilla.org/2015/06/the-state-of-web-components/
  - https://www.polymer-project.org/2.0/docs/devguide/shadow-dom
- チュートリアル系
  - https://www.webcomponents.org/introduction
  - https://developers.google.com/web/fundamentals/getting-started/primers/customelements?hl=ja
  - https://www.html5rocks.com/en/tutorials/webcomponents/imports/

おわり