Custom Elements Everywhere

Rob Dodson
@rob_dodson

Polymer Summit 2017
I've been at Google for 3+ years.
6b/ every few days I see someone say ~"WC work in all frameworks! All our problems are solved", it's pretty clear none have tried the above
every few days I see someone say ~"WC work in all frameworks! All our problems are solved", it's pretty clear none have tried the above
Custom Elements should work everywhere.
Custom Elements *should* work everywhere. So... why don't they?
Did we put the 🛒 before the the 🦄 unihorse
TWO PARTS

PART no. 1
What is a "good" Custom Element?

PART no. 2
How should frameworks act?
What is a "good" Custom Element?
Do you know if there are any reference Custom Elements I could look at which live up to this standard?

Not that I'm aware of.
HMMMM...
SCIENCE!
HowTo: Components

A collection of educational custom elements that demonstrate best practices.

Examples are meant to be read, interpreted, and ported to your own elements.
The constructor is a good place to create Shadow DOM, though you should avoid touching any attributes or Light DOM children as they may not be available yet.

connectedCallback fires when the element is inserted into the DOM. It's a good place to set the initial role, tabindex, internal state, and install event listeners.

A user may set a property on an instance of an element, before its prototype has been connected to this class. The `_upgradeProperty` method will check for any instance
HowTo: Components are a work in progress.
Check it out @ bit.ly/howto-components
| Shadow DOM | Attributes & Properties | Events |
Avoid absolutism
Shadow DOM
Does your element *need* Shadow DOM?
You gotta use Shadow DOM!

You gotta use Shadow DOM!
I DO WHAT I WANT
<howto-checkbox>

howto-checkbox.js + howto-checkbox.css
If someone puts your element inside another shadow root, you'll have to figure out how to get your styles in there.
Provide a lightweight mechanism to add styles to a custom element #468

Proposal

Allow a user to define a set of styles to apply to a custom element as an option to customElements.define. Conceptually, providing styles would make the element act as if it had a shadowRoot including a style element with the provided css. The rules used to target the element would be the same as in Shadow DOM.

customElements.define("cool-element", CoolElement, {styles: '::host { display: block; }'});

Discussion

To style a custom element that does not otherwise need a shadowRoot incurs an unfortunate
Create a shadow root if you want to self-apply styles.
We have CSS-in-JS for scoping styles, so why do we need Shadow DOM?
const template = document.createElement('template');
template.innerHTML = `<div>Count: <slot></slot></div>`;
export class CountWithShadow extends HTMLElement {
  constructor () {
    super();
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.appendChild(template.content.cloneNode(true));
  }
}

customElements.define('x-count-with-shadow', CountWithShadow);
export class CountWithoutShadow extends HTMLElement {
    constructor () {
        super();
    }
    connectedCallback() {
        this.appendChild(template.content.cloneNode(true));
    }
}
customElements.define('x-count-without-shadow', Count);
// Meanwhile, in a React component...
render () {
  const { count } = this.state; // increments every second
  return (  
    <div>
      <x-count-with-shadow>{count}</x-count-with-shadow>
      <x-count-without-shadow>{count}</x-count-without-shadow>
    </div>
  );
}
Count: 1

1

Count:
Count: 1

1

Count:

<x-count-with-shadow>
  1
</x-count-with-shadow>
Count: 1

1

Count:
Count: 1

<x-count-without-shadow>
  1
  <div>Count: <slot></slot></div>
</x-count-without-shadow>

ummm...
Count: 2

<x-count-without-shadow>
2
</x-count-without-shadow>
Place any children you create into a shadow root.
These children are your **implementation** and should be hidden.
Does your element need Shadow DOM?
Yeah, it probably does.
Without Shadow DOM you lose all guarantees of safety.
Create a shadow root if you want to **self-apply styles**.

Place any children **you create** into a shadow root.
Attributes & Properties
Hey!

HTML has a spec!
But HTML is also kinda gnarly
Accept **primitive data** as either attributes or properties; ideally, **both**.
Reflect primitive data from attribute to property, and vice versa.
class CustomVideo extends HTMLElement {
    get preload() {
        const value = this.getAttribute('preload');
        return value === null ? 'auto' : value;
    }

    set preload(value) {
        this.setAttribute('preload', value);
    }
}

Polymer Summit 2017
class CustomVideo extends HTMLElement {
  get preload() {
    const value = this.getAttribute('preload');
    return value === null ? 'auto' : value;
  }

  set preload(value) {
    this.setAttribute('preload', value);
  }
}

Polymer Summit 2017
class CustomVideo extends HTMLElement {
    get preload() {
        const value = this.getAttribute('preload');
        return value === null ? 'auto' : value;
    }

    set preload(value) {
        this.setAttribute('preload', value);
    }
}
Only accept rich data as properties.
Do **NOT** reflect rich data properties to attributes.
It's **expensive** to reflect.
A serialized object will lose its **identity**.
Accept **primitive data** as either attributes or properties.

**Reflect** primitive data from attribute to property, and vice versa.

Only accept **rich data** as properties.

Do **NOT** reflect rich data from property to attribute.
Events
It's superfluous and may cause **infinite loops**.
Do **NOT** dispatch events in response to the host setting a property (downward data flow).
Do dispatch events in response to **internal** element activity.
The element knows something changed, but the host does not.
Events

Do **NOT** dispatch events in response to the host setting a property (downward data flow).

Do dispatch events in response to **internal** element activity.
What is a "good" Custom Element?
Building Components.

A guide to **Custom Element best practices**.

Custom Elements allow you to extend HTML and define your own tags. They’re an incredibly powerful feature, but they’re also new, which means it’s not always clear how best to implement your own element.

To help teams create the best possible experiences we’ve put together this checklist which breaks down all the things we think it takes to be a well-behaved Custom Element.

### Checklist

**Shadow DOM**

Create a shadow root to encapsulate styles.

**Why?** Encapsulating styles in your element’s shadow root ensures that it will work regardless of where it is used. This is especially important if a developer wishes to place your element inside of another element’s shadow root. This applies to even simple elements like a checkbox or radio button. It might be the case that the only content inside of your shadow root will be the styles themselves.

**Example** The `<bento-checkbox>` element.

Create your shadow root in the constructor.

**Why?** The constructor is where you have exclusive knowledge of your element. It’s a great time to setup implementation details that you don’t want other elements messing around with. Doing this work in a later callback, like `connectedCallback`, means you will need to guard against situations where your element is detached and then reattached to the document.

**Example** The `<bento-checkbox>` element.
Check it out @ bit.ly/building-components
How should **frameworks** act?
Billions and billions...
Custom Elements Everywhere

Making sure frameworks and custom elements can be BFFs 🍺
Custom Elements Everywhere
Making sure frameworks and custom elements can be BFFs 🍺
Rob waves his arms around and apologizes
known unknowns
Custom Element **test scores**

- **Angular**  
  Score: 30/30

- **Preact**  
  Score: 24/30

- **React**  
  Score: 16/30

- **Vue**  
  Score: 30/30
Attributes & Properties
The developer *tells* the framework how to pass data.

The framework *determines* how to pass data.
Angular

<my-element [foo]="bar">
Angular

```html
<my-element [foo]="bar">

set the foo property equal to bar
```
Angular

\[<\text{my-element} [\text{foo}] = "\text{bar}" >\]

i.e. myElement.foo = bar
Angular

<my-element [foo]="bar">

<my-element [attr.baz]="squid">

set the baz attribute equal to squid
Angular

<my-element [foo]="bar">

<my-element [attr.baz]="squid">

i.e. myElement.setAttribute('baz', squid)
<my-element :foo="bar"> set the foo attribute equal to bar
Vue

<my-element :foo.prop="bar"> set the foo property equal to bar
React

<my-element foo={bar}>

set the foo attribute equal to bar
React

```jsx
<my-element foo={bar} />
<my-element foo="[object Object]" />
```
React

You can work around this by **imperatively** grabbing a reference to the element and setting the property on it.
RFC: Plan for Attributes in React 16 #10399

gaearon opened this issue 3 days ago • 21 comments

gaearon commented 3 days ago • edited

This is a more formal conclusion of the discussion in #7311. It is mostly (not yet fully) implemented by #10385.

This is meant to address #148.

I wrote this doc but it's mostly based on discussion with @nikhunk. I decided to write it in an attempt to formalize the behavior we want, so that if there are bugs, we can refer back to this.

Current Behavior

React only lets you use "approved" camelCase properties that look organic in JavaScript:

```javascript
// No warning
<div className="/"></div>  // => <div className="/"/>
```
Preact

<my-element foo={bar}>
Preact

```jsx
<my-element foo={bar}>

set the foo property if it’s available
```
if ('foo' in myElement)
    myElement.foo = bar
else
    myElement.setAttribute('foo', bar)
if ('foo' in myElement)
    myElement.foo = bar
else
    myElement.setAttribute('foo', bar)
Preact

```javascript
if ('foo' in myElement)
    myElement.foo = bar
else
    myElement.setAttribute('foo', bar)
```
if ('foo' in myElement) {
    myElement.foo = bar
} else {
    myElement.setAttribute('foo', bar)
}
Events
Angular

<my-element (foochanged)="onFoo()">
Angular

```html
<my-element (foochanged)="onFoo()">
  myElement.addEventListener('foochanged', onFoo)
</my-element>
```
Angular

<my-element (foochanged)="onFoo()"> lowercase
Angular

<my-element (FOOChanged)="onFoo()">
  CAPSCase
</my-element>
Angular

<my-element (fooChanged)="onFoo()">

camelCase

</my-element>
Angular

<my-element (foo-changed)="onFoo()">
  kebab-case
</my-element>
Angular

<my-element (FooChanged)="onFoo()">

PascalCase
Angular

<my-element (FoOcHaNgEd)="onFoo()">AsSh01EcAsE</my-element>
I DO WHAT I WANT
Vue

<my-element v-on:fooChanged="onFoo"/>
Vue

same behavior as Angular
<my-element onFooChanged={{onFoo}}>
React

<my-element onFooChanged={onFoo}>

unfortunately won't work.
You can work around this by **imperatively** grabbing a reference to the element and adding an event listener.
Bypass synthetic event system for Web Component events

#7901

staltz opened this issue on Oct 6, 2016 - 54 comments

staltz commented on Oct 6, 2016 • edited

To use a Web Component in React, you must directly attach an event listener to the React ref for that WC. We could change the implementation so that when React detects a custom element (tag names with dashes, like `my-component`), it will bypass the synthetic event system (and the whitelist) and just attach the event listener on the element itself.

Why bypass the synthetic event system? Because *anyway* we already need to bypass it manually when using a Web Component. I'm not so familiar with the React codebase, but this naive approach seems to work. Whoever uses Web Components in React can be responsible for whatever downsides that would cause, maybe in performance, I don't know. They are already having those (supposed) downsides, this issue is just
Preact

<my-element onFooChanged={onFoo}>
Preact

```jsx
<my-element onFooChanged={onFoo}>
  TOTES WORKS!
</my-element>
```
<my-element onFooChanged={onFoo}>
  TOTES WORKS!
  kinda. sorta. mostly...
</my-element>
Preact

```javascript
<my-element onFooChanged={onFoo}>
  myElement.addEventListener('foochanged', onFoo)
  always calls .toLowerCase()
</my-element>
```
So...
So...

it doesn't support

AsShOIEcAsE?!?!
Allow mixed case events #788

Developit wants to merge 1 commit into master from event-case

Developit commented 8 days ago
No description provided.

Coverage remained the same at 100.0% when pulling 4288e18 on event-case into e02609e on master.
So, what did we learn?
There are **best practices** we can all follow 👣
What's broken today seems **easy to fix** 🔧
amazing people 👉

HowTo: Components team

Surma  Monica  Ewa

Reviewers

Sean  Rob  Jason  Evan
Dan  Domenic  Steve  Dominic
Thank You!

Images by Simon Child, Eucalyp, Edward Boatman from the Noun Project.

Polymer Summit 2017