# Web Components

Seth House <seth@eseth.com>

Utah JS Conf 2013

2013-05-17

# Outline

# An overview

# Slide notes

https://github.com/whiteinge/presentations/tree/master/utahjs_conf_2013-05-17_web-components

# What are web components

- `<element>`

# What are web components

- `<element>`
- Shadow DOM

# What are web components

- `<element>`
- Shadow DOM
- `<template>`

# What are web components

- `<element>`
- Shadow DOM
- `<template>`
- MutationObserver

# What are web components

- `<element>`
- Shadow DOM
- `<template>`
- MutationObserver
- `Object.observe()`

# What are web components

- `<element>`
- Shadow DOM
- `<template>`
- MutationObserver
- `Object.observe()`
- These specs are not yet finalized!

# Why are web components exciting?

- Embeddable widgets
  - Social media buttons

# Why are web components exciting?

- Embeddable widgets
  - Social media buttons
- Reusable element libs / element frameworks
  - Tabs, modals, nav bars, accordions, carousels, etc

# Why are web components exciting?

- Embeddable widgets
  - Social media buttons
- Reusable element libs / element frameworks
  - Tabs, modals, nav bars, accordions, carousels, etc
- Front-end MV* frameworks
  - Model driven views (MDV)

# Custom elements

# <element> / document.register()

- New HTML elements
- Extend existing elements

# <element> / document.register()

- New HTML elements
- Extend existing elements
- Element lifecycle hooks

# <element> / document.register()

- New HTML elements
- Extend existing elements
- Element lifecycle hooks
- Import / share external components
- Using standard web techniques

# Example: declarative style

Register the element once:

mybutton.html

```
<element name="x-mybutton" extends="button">
    <template></template>
    <style></style>
    <script></script>
</element>
```

Use anywhere:

index.html

```
<link rel="import" href="x-mybutton.html">
<x-mybutton>Detonate</x-mybutton>
```

# Example: imperative style

```
document.register('x-mybutton');
```

# Extend existing elements

- Create new element object from an element prototype
- Extends `HTMLElement` by default

```
document.register('x-mybutton', {
    prototype: Object.create(
        window.HTMLButtonElement.prototype),
});
```

# Add constructor reference

- Always available via standard `document.createElement`

# Add constructor reference

- Always available via standard `document.createElement`
- Explicitly add element constructor to `window` object

mybutton.html

```
<element name="x-mybutton" extends="button"
         constructor="MyButton">
    <script>
        MyButton.prototype = {
            explode: function(e) {
                this.innerHTML = "Boom!"; },
        };
    </script>
</element>
```

# Access as a regular element

index.html

```
<link rel="import" href="x-mybutton.html">

<script>
    var b = new MyButton();
    b.addEventListener('click', function(e) {
        e.target.explode();
    });
    document.body.appendChild(b);
</script>
```

# Getters / setters

```
document.register('x-mybutton', {
    prototype: Object.create(
            window.HTMLButtonElement.prototype, {
        bar: {
            get: function() { return 'bar' },
        },
    }),
});

console.log(
    document.querySelector('x-mybutton').bar);
```

# Lifecycle

mybutton.html

```
<element name="x-mybutton" extends="button">
    <script>
        this.lifecycle({
            created: function() {},
            inserted: function() {},
            removed: function() {},
            attributeChanged: function() {},
        });
    </script>
</element>
```

# Shadow DOM

# Encapsulation

- Styles inside a shadow root are scoped
- Styles outside a shadow root don't apply
  - Can opt-in
  - `resetStyleInheritance`, `applyAuthorStyle`

# Encapsulation

- Styles inside a shadow root are scoped
- Styles outside a shadow root don't apply
  - Can opt-in
  - `resetStyleInheritance`, `applyAuthorStyle`
- Browsers already host hidden DOM
  - Browser-native controls
  - `<input type="date">`
  - `<video src="...">`

# Creating a shadow DOM

```
var shadow = host.createShadowRoot();
shadow.innerHTML = "<p>Things</p>";
```

# Template

# <template>

```
<template>
    <img src="" class="avatar">
    <div class="comment"></div>
</template>
```

- Clonable blueprint

# <template>

```
<template>
    <img src="" class="avatar">
    <div class="comment"></div>
</template>
```

- Clonable blueprint
- Parsed not rendered (`<script type="text/template">`)

# <template>

```
<template>
    <img src="" class="avatar">
    <div class="comment"></div>
</template>
```

- Clonable blueprint
- Parsed not rendered (`<script type="text/template">`)
- Inert until activated
  - Images not loaded, scripts not run, media not played

# <template>

```
<template>
    <img src="" class="avatar">
    <div class="comment"></div>
</template>
```

- Clonable blueprint
- Parsed not rendered (`<script type="text/template">`)
- Inert until activated
  - Images not loaded, scripts not run, media not played
- Activated by appending to a DOM node

# Example

```
<template>
    <content select=".comment"></content>
</template>
```

# Object.observe()

# Data binding

- It'll change your religion

# Data binding

- It'll change your religion
- Watch a POJO (plain ol' JavaScript object) for changes

# Why

- Update DOM when object changes
  - MDV

# Why

- Update DOM when object changes
  - MDV
- Persist object to storage backend
  - Current state
  - Changes over time

# Why

- Update DOM when object changes
  - MDV
- Persist object to storage backend
  - Current state
  - Changes over time
- Constraints (computed properties)

# Allows good control over ordering

For example:

1. Update value
2. Recalc computed properties
3. Persist new values

# Replaces getters & setters or dirty-checking

- Getters / setters

# Replaces getters & setters or dirty-checking

- Getters / setters
  - Performant
  - Either
    - ES5 getters / setters
    - Call functions instead of referencing values

# Replaces getters & setters or dirty-checking

- Getters / setters
  - Performant
  - Either
    - ES5 getters / setters
    - Call functions instead of referencing values
- Dirty checking

# Replaces getters & setters or dirty-checking

- Getters / setters
  - Performant
  - Either
    - ES5 getters / setters
    - Call functions instead of referencing values
- Dirty checking
  - Usually invoked when data *can* change to check if data *did* change
  - Potentially expensive (many fast updates)
  - Usually checks entire object

# Replaces getters & setters or dirty-checking

- Getters / setters
  - Performant
  - Either
    - ES5 getters / setters
    - Call functions instead of referencing values
- Dirty checking
  - Usually invoked when data *can* change to check if data *did* change
  - Potentially expensive (many fast updates)
  - Usually checks entire object
  - Angular team benchmarked replacing dirty checking with `Object.observe()` in Chrome Canary
    - Dropped from 40ms to 2ms
    - 20x–40x faster

# Example

```
var myobj = {};
Object.observe(myobj, function(changes) {
    changes.forEach(function(change) {
        // new, updated, deleted, reconfigured
        change.type;
        // affected object
        change.object;
        // affected property name
        change.name;
        // value of property before the change
        change.oldValue;
    });
});

Object.unobserve(el, callback);
```

# ES5 getters/setters

- ES5 getters/setters (e.g., computed properties) are not observed

```
Object.defineOwnProperty(obj, 'val', {
    get: function() { return thing },
    set: function(val) { thing = val },
});
```

# ES5 getters/setters

- ES5 getters/setters (e.g., computed properties) are not observed

```
Object.defineOwnProperty(obj, 'val', {
    get: function() { return thing },
    set: function(val) { thing = val },
});
```

- Not a solvable problem
- You must include this functionality yourself inline or by decorating

# MutationObserver

# What

- Triggered by DOM changes
  - Adding removing elements
  - Changing elements
  - Changing element attributes

# What

- Triggered by DOM changes
  - Adding removing elements
  - Changing elements
  - Changing element attributes
- Observer not listener
- Callback triggered at end of DOM changes with list of all changes

# Replaces Mutation Events

- Fired too often (fired for each change)
- Slow (event based)
- Deprecated
- Stability problems

# Why

- Browser extensions
  - Google Voice extension listens for text changes to transform phone number patterns into hyperlinks.
  - JS libs enhancing HTML; Dojo implementing a combo box, tough to monitor changes after setting it up
- Framework / library authors

# Example

```
var observer = new MutationObserver(function(mutations
    mutations.forEach(function(record) {
        record.addedNodes; // nodes
    });
});

observer.observe(el, {
    childList: true, // child insert/remove
    subtree: true, // observer subtree root at el
    characterData: true, // textContent changes
    attribute: true, // changes to attributes
});

observer.disconnect();
```

# Model driven views (MDV)

# The big picture

- Two-way data binding without any code

# The big picture

- Two-way data binding without any code

```
<polymer-panels
        on-select="panelSelectHandler"
        selected="{{selectedPanelIndex}}">
</polymer-panels>
```

# Templating and data binding

```
<ul id="example">
    <template iterate>
        <li>{{ name }}
        <ul>
            <template iterate="skills">
                <li></li>
            </template>
        </ul>
        </li>
    </template>
</ul>

<script>
    document.querySelector('#example').model = [
        {name: 'Sally', skills: ['carpentry']},
        {name: 'Helen', skills: ['weaving', 'omnipoten
```

# Outline

# Frameworks

# Angular

- *Not* a polyfill
- `Object.observe()` -like data-binding (POJO)
- `document.register()` -like custom elements (Directives)
- MDV-like templating

# Dart

- `http://www.dartlang.org/`
- Web components (`<element>`)
- Templates (`<template>`)
- Encapsulation (emulates Shadow DOM)
- Data binding (watchers)
- MDV (DOM templating)

# Polymer

- `http://polymer-project.appspot.com/`
- Formerly Toolkitchen; fomerly Toolkitchensink

# Polymer

- `http://polymer-project.appspot.com/`
- Formerly Toolkitchen; fomerly Toolkitchensink
- `platform.js` (31 KB)
  - Polyfills (shadow DOM, custom elements, mutation observer, MDV)
- `polymer.js`
  - Web application framework

# Polymer

- `http://polymer-project.appspot.com/`
- Formerly Toolkitchen; fomerly Toolkitchensink
- `platform.js` (31 KB)
  - Polyfills (shadow DOM, custom elements, mutation observer, MDV)
- `polymer.js`
  - Web application framework
- Custom functional elements
- Custom UI widget elements

# Polymer

- `http://polymer-project.appspot.com/`
- Formerly Toolkitchen; fomerly Toolkitchensink
- `platform.js` (31 KB)
  - Polyfills (shadow DOM, custom elements, mutation observer, MDV)
- `polymer.js`
  - Web application framework
- Custom functional elements
- Custom UI widget elements
- Working with Mozilla to ensure compat between shims
- Browser support: evergreen

# X-Tag

- `http://x-tags.org/`
- `https://github.com/x-tag`
- Originally a proof-of-concept
  - Begat the true polyfill

# Libraries

# Mozilla's web-components

- `https://github.com/mozilla/web-components`
- `document.register()` polyfill
  - Lifecycle events
  - Prototypical element inheritance
  - (1.9 KB)
- Browser support: ES5

# Object.observe()

- `https://github.com/jdarling/Object.observe`
  - Uses polling and getters / setters
  - Can miss very quick changes
- `https://github.com/KapIT/observe-shim`
  - Requires manually checking for changes (?)

# Watch.JS

- *Not* a polyfill for `Object.observe()`

# Watch.JS

- *Not* a polyfill for `Object.observe()`
- `https://github.com/melanke/Watch.JS`
- Automatic getters / setters
- Overrides `.push()` etc
- Macro-level dirty-checking
- (1.4 KB)
- Browser support: ES5

# DOM-based templating with data binding

- Rivets
  - `http://rivetsjs.com/`
  - Two-way data binding
    - Pluggable backends
  - DOM-based templating
  - (2.3 KB)

# DOM-based templating with data binding

- Rivets
  - `http://rivetsjs.com/`
  - Two-way data binding
    - Pluggable backends
  - DOM-based templating
  - (2.3 KB)
- JS-Bind
  - `http://www.js-bind.com/`

# DOM-based templating with data binding

- Rivets
  - `http://rivetsjs.com/`
  - Two-way data binding
    - Pluggable backends
  - DOM-based templating
  - (2.3 KB)
- JS-Bind
  - `http://www.js-bind.com/`
  - (6.9 KB)
- Knockout
  - `http://knockoutjs.com/`

# DOM-based templating with data binding

- Rivets
  - `http://rivetsjs.com/`
  - Two-way data binding
    - Pluggable backends
  - DOM-based templating
  - (2.3 KB)
- JS-Bind
  - `http://www.js-bind.com/`
  - (6.9 KB)
- Knockout
  - `http://knockoutjs.com/`
- Many others