Tuesday, December 4, 2018 9:48 AM

WEB COMPONENTS

Problems in current implementation
- Style markup
- Nesting of divs
- Undescriptive markup(Uses divs and spans)
- no native templates, each template has its own aprroach
- no bundling (bundling of html, css and js)
- no standards for building components

WEB COMPONENT
- Templates
- Custom elements
- Shadow DOM
- Imports
- Bundling

Supported in Chrome
http://jonrimmer.github.io/are-we-componentized-yet

Use Polyfill to get web components work in all modern browsers

Custom Elements
<my-navigation>

Registering custom elements
1. Create a prototype
let slickTabs = Object.create(HTMLElement.prototype);
let slickButton = Object.create(HTMLButtonElement.prototype);

2. Register the new element via registerElement
document.registerElement('slick-tabs');

3. Use it by adding to DOM or using tag on page
document.appendChild(new SlickTabs());

4. Extend a registered element
var xFooProto = Object.create(HTMLElement.prototype);

var xFooExtended = document.registerElement('x-foo-extended',
{ prototype: xFooProto, extends: 'x-foo']);

4 ways to instantiate the web component

1. Markup
<my-component />

2. new Operator

```
var myNewComponent = new myComponent();

3. createElement
var myNewComponent = document.createElement('my-component');

4. innerHTML
el.innerHTML = '<my-component />';

Example:
<script>
  var myComponentProto = Object.create(HTMLElement.prototype);
  myComponentProto.createdCallback = function() {
    this.innerHTML = '<div> Hi </div>'
  }
  var myComponentEl = document.registerElement('my-component', {
    prototype: myComponentProto
  })
 </script>

Extending custom elements
Example:
<button is="my-button" />
<script>
  var myButtonProto = Object.create(HTMLButtonElement.prototype);
  myButtonProto.createdCallback = function() {
    this.innerHTML = 'Hello';
    this.value = "Default Value";
    this.style.color = 'blue';
  }
  var myButtonProtoEl = document.registerElement('my-button', {
    prototype: myButtonProto,
    extends: 'button'
  });
</script>

Lifecycle Callback Methods
createdCallback - called when instance is create
attachedCallback - called when instance inserted into DOM
detachedCallback - called when instance removed from DOM
attributeChangedCallback - called when attributes are added, removed or updated.

Naming approaches
1. X as a prefix
  <x-gravatar>
2. Polyfill name
  <polymer-gravtar>

Check component kitchen's library for existing web components and their naming conventions used


Shadow DOM fundamentals

Scoping needs to be implemented
```

- to avoid accidental styling other parts of the page
- to avoid accidental styling of the web component by others
- to hide markup from accidental manipulation

Light DOM vs Shadow DOM (DOM - Document Object Model)
Logical DOM - Consists of both Light and Shadow
Shadow DOM - hides away the complexity of the page
Light DOM - DOM which we currently see

Shadow DOM encapsulates DOM Subtrees and styles
Existing elements
<iframe>
Clunky to read
Undescriptive
Excessive encapsulation
No clean API

<canvas>
Accessibility issues
SEO issues
cant easily compose
cant extend existing elements

Create Shadow DOM
1. Select shadow host
2. Create a shadow root