A Practical Guide to Vanilla Web Components

WEB COMPONENT FUNDAMENTALS

Leon Revill
WEB ARCHITECT

@revillweb www.revillweb.com

Overview

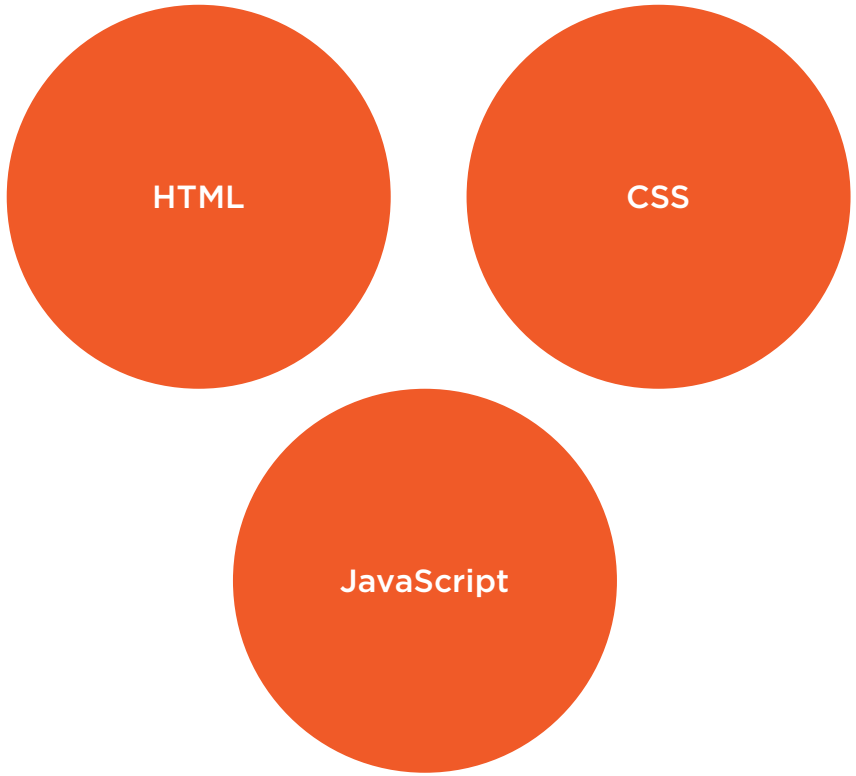
Learn about the building blocks of a web component

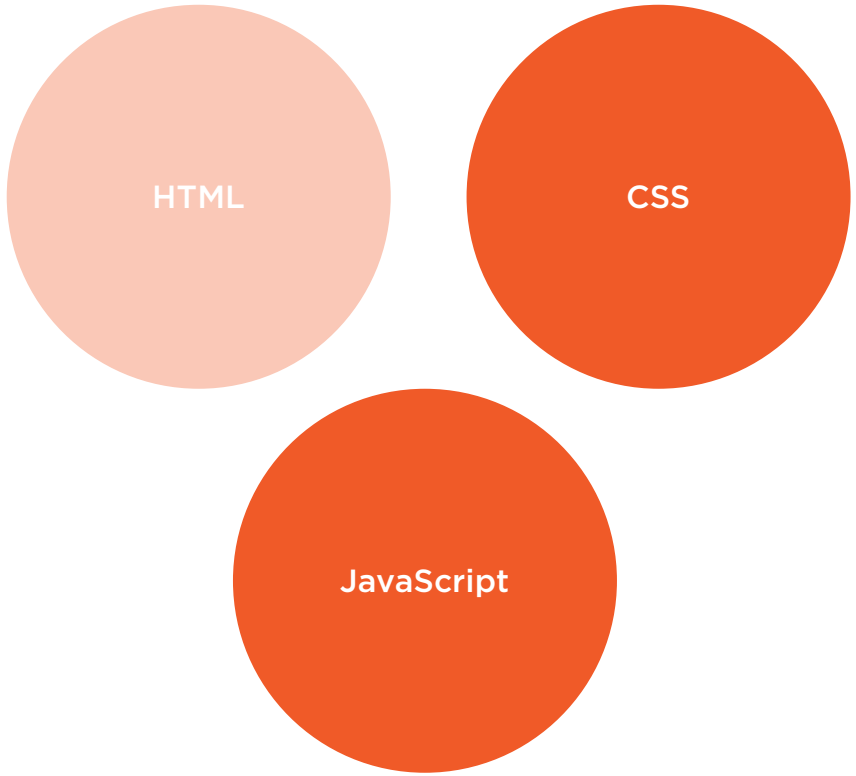
The four web component sub-specs

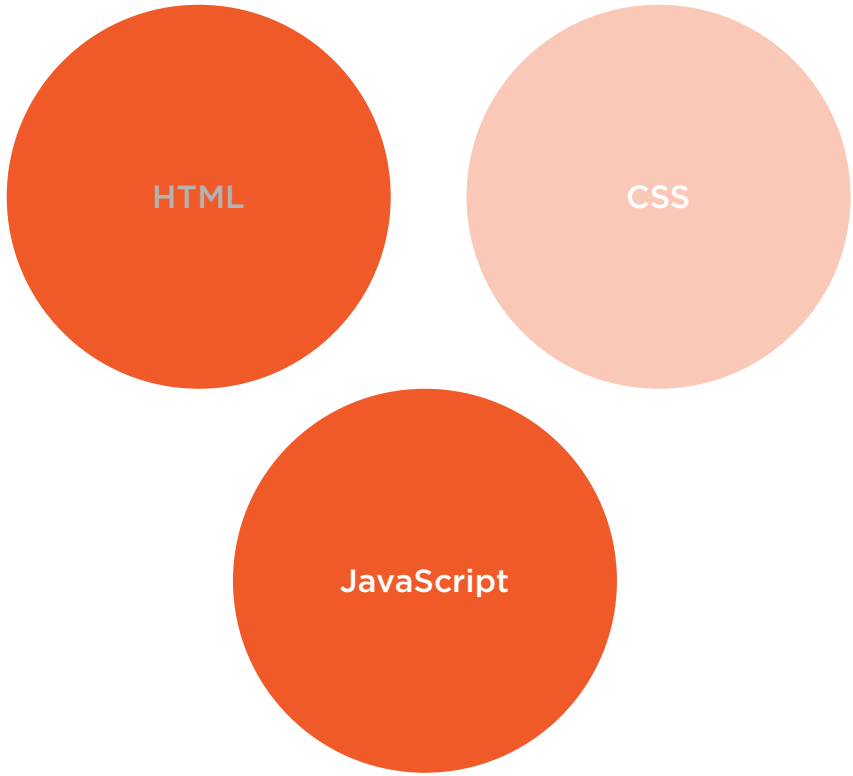
- What each specification provides
- How to use each of the specs

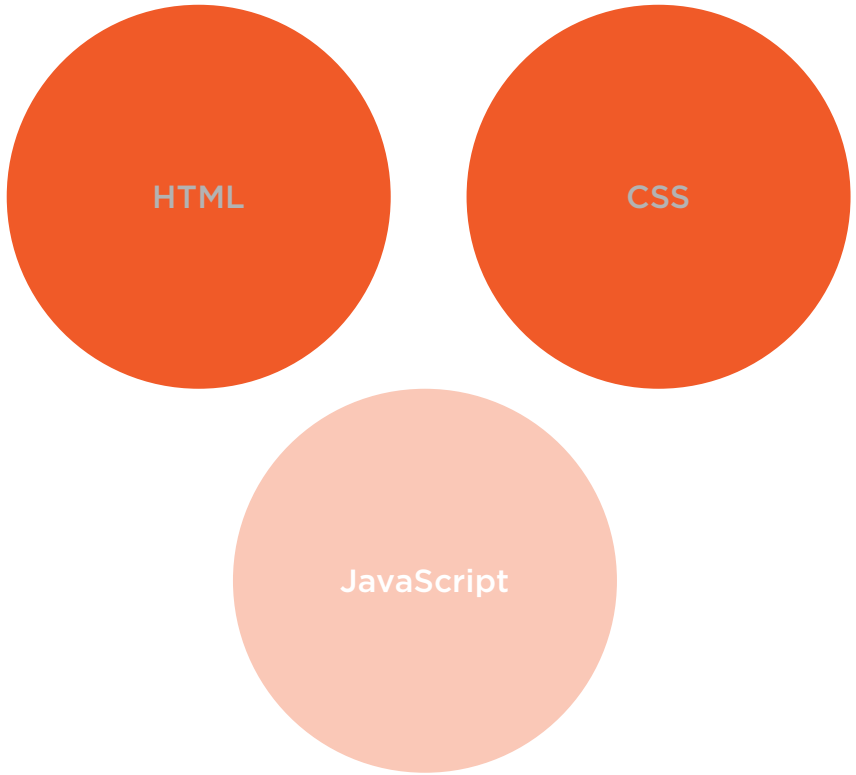
Set the foundation for more advanced concepts in this course

What Are Web Components?


HTML
CSS
JavaScript


HTML
CSS
JavaScript


HTML
CSS
JavaScript


HTML
CSS
JavaScript

WEB COMPONENTS

What Are “Vanilla” Web Components?

Polymer

Polymer

SkateJS

Polymer

SkateJS

x-tag


JS

Why Should We Care About Web Components?

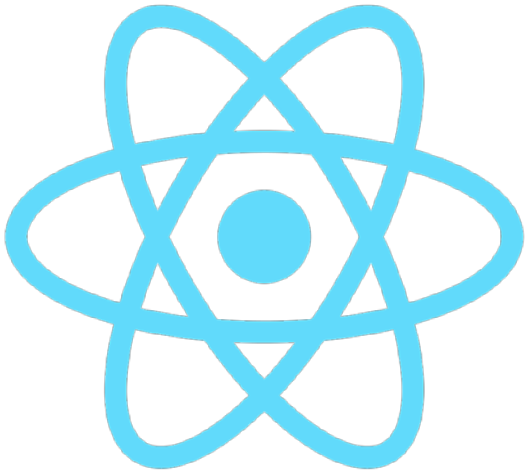
jQuery

jQuery

AngularJS

jQuery

AngularJS



React

```
<!DOCTYPE html>
<html lang="en">
<head>









</head>
<body>

</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
   <script src=“jquery.min.js”></script>









</head>
<body>

</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
   <script src=“jquery.min.js”></script>
   <script src=“jquery-plugin.js”></script>









</head>
<body>

</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
   <script src=“jquery.min.js”></script>
   <script src=“jquery-plugin.js”></script>

   <link rel="stylesheet" href=“jquery-ui.min.css">




</head>
<body>

</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
   <script src=“jquery.min.js”></script>
   <script src=“jquery-plugin.js”></script>

   <link rel="stylesheet" href=“jquery-ui.min.css">
   <link rel="stylesheet" href=“jquery-plugin.css">




</head>
<body>

</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
   <script src=“jquery.min.js”></script>
   <script src=“jquery-plugin.js”></script>

   <link rel="stylesheet" href=“jquery-ui.min.css">
   <link rel="stylesheet" href=“jquery-plugin.css">

   <script src=“jquery-plugin2.js”></script>
   <link rel="stylesheet" href=“jquery-plugin2.css">
</head>
<body>

</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
   <script src=“my-web-component.js”></script>









</head>
<body>

</body>
</html>
```

Style Encapsulation

```
<!DOCTYPE html>
<html lang="en">
 ▶ <head>…</head>
 ▼ <body> == $0
    ▶ <p>…</p>
    ▼ <my-component>
      ▼ #shadow-root (open)
        ▶ <style>…</style>
        ▶ <p>…</p>
      </my-component>
      <script src="dist/bundle.js"></script>
   </body>
</html>
```

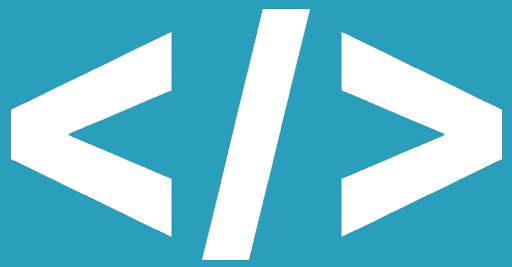
Style encapsulation

Fewer dependencies & prerequisites

Framework / library agnostic

Longer life components

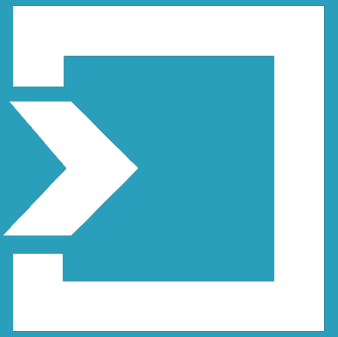
The Web Components Specifications



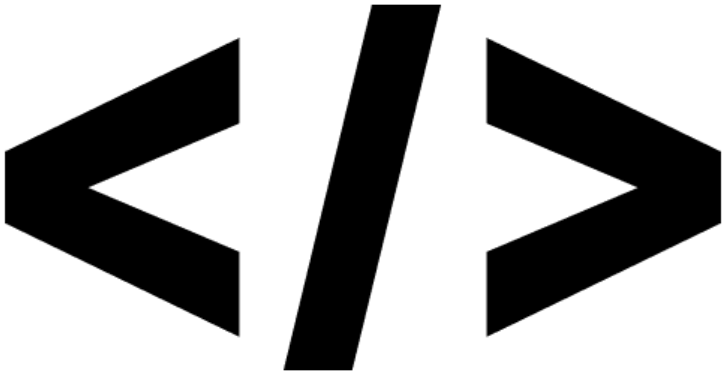
Custom Elements

Shadow DOM

HTML Templates



HTML Imports



Custom Elements

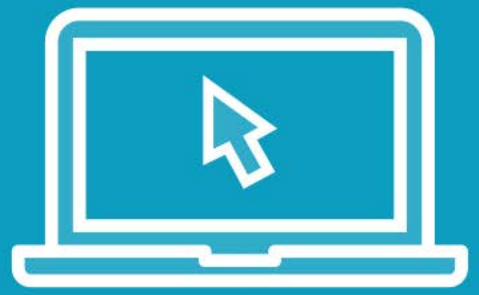
Web Components are DOM elements

Create your own element

Add custom properties, methods and functionality

Extend native elements and custom elements

Demo



Using Custom Elements

Basic anatomy of DOM elements

How to create a custom element

The lifecycle callback methods

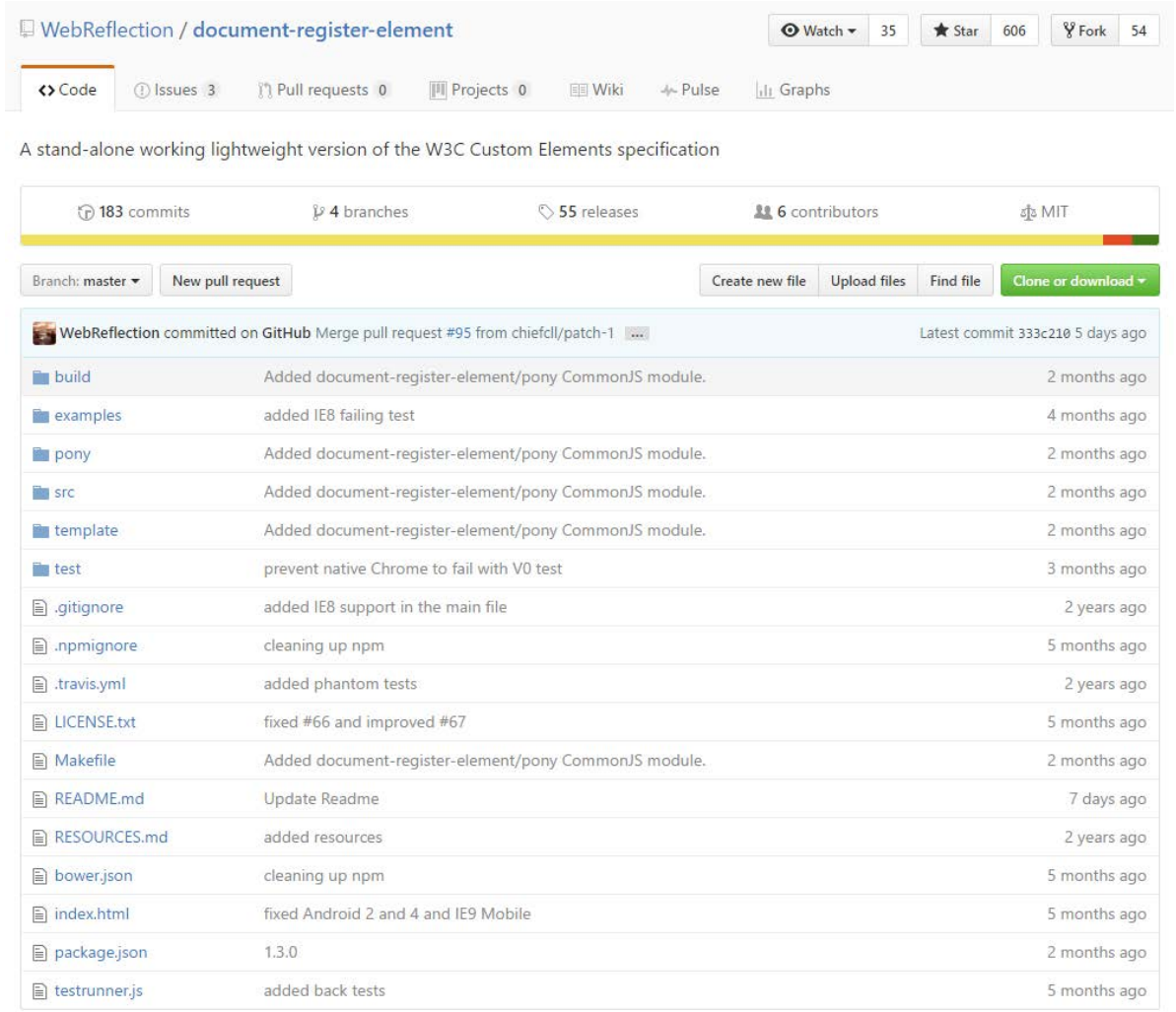
Extending a custom element

Extending a native element

"Custom Elements allow you to define your own interface with its own methods and properties"

Custom Elements V1 Polyfill

https://goo.gl/EcC7Qg



WebReflection / document-register-element

Watch 35 | Star 606 | Fork 54

Code | Issues 3 | Pull requests 0 | Projects 0 | Wiki | Pulse | Graphs

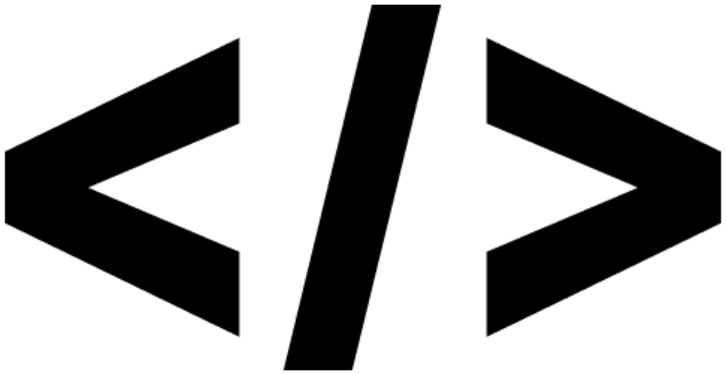
A stand-alone working lightweight version of the W3C Custom Elements specification

183 commits | 4 branches | 55 releases | 6 contributors | MIT

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

WebReflection committed on GitHub Merge pull request #95 from chiefcll/patch-1 ... Latest commit 333c210 5 days ago

build	Added document-register-element/pony CommonJS module.	2 months ago
examples	added IE8 failing test	4 months ago
pony	Added document-register-element/pony CommonJS module.	2 months ago
src	Added document-register-element/pony CommonJS module.	2 months ago
template	Added document-register-element/pony CommonJS module.	2 months ago
test	prevent native Chrome to fail with V0 test	3 months ago
.gitignore	added IE8 support in the main file	2 years ago
.npmignore	cleaning up npm	5 months ago
.travis.yml	added phantom tests	2 years ago
LICENSE.txt	fixed #66 and improved #67	5 months ago
Makefile	Added document-register-element/pony CommonJS module.	2 months ago
README.md	Update Readme	7 days ago
RESOURCES.md	added resources	2 years ago
bower.json	cleaning up npm	5 months ago
index.html	fixed Android 2 and 4 and IE9 Mobile	5 months ago
package.json	1.3.0	2 months ago
testrunner.js	added back tests	5 months ago



Custom Elements

The Custom Elements spec is one of the more important parts of Web Components

Understand basic anatomy of DOM elements

Know what callback methods the Custom Elements API provides

Have seen how to extend both custom and native DOM elements

Custom Elements: Further Reading

https://goo.gl/Ykp35L

Custom Elements v1: Reusable Web Components

☆☆☆☆☆

By Eric Bidelman

Engineer @ Google working on Polymer, Chrome, and the web

TL;DR

With Custom Elements, web developers can **create new HTML tags**, beef-up existing HTML tags, or extend the components other developers have authored. The API is the foundation of web components. It brings a web standards-based way to create reusable components using nothing more than vanilla JS/HTML/CSS. The result is less code, modular code, and more reuse in our apps.

Introduction

★ **Note:** This article describes the new Custom Elements v1 spec. If you've been using custom elements, chances are you're familiar with the v0 version shipped in Chrome 33. The concepts are the same, but the v1 spec has important API differences. Keep reading to see what's new or check out the section on History and browser support for more info.

The browser gives us an excellent tool for structuring web applications. It's called HTML. You may have heard of it! It's declarative, portable, well supported, and easy to work with. Great as HTML may be, its vocabulary and extensibility are limited. The HTML living standard lacks a way to automatically associate JS behavior with your markup... until now.

Custom elements are the answer to modernizing HTML; filling in the missing pieces, and bundling structure with behavior. If HTML doesn't provide the solution to a problem, we can create a custom element that does. **Custom elements teach the browser new tricks while preserving the benefits of HTML**.

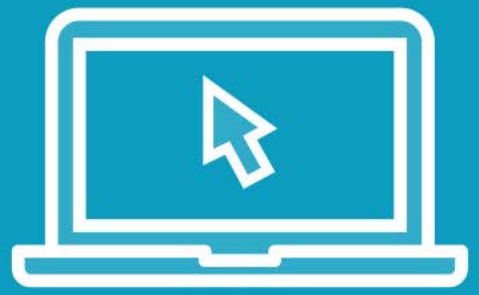
Shadow DOM

Shadow DOM is a sub-DOM tree attached to a DOM element

Shadow DOM provides true encapsulation for elements

There are two types of Shadow DOM: “open” and “closed”

Demo



Using Shadow DOM

Add a shadow root to an element

Interacting with the Shadow DOM

The Shadow DOM in action

Shadow DOM: Further Reading

https://goo.gl/t8nPzf

Shadow DOM v1: Self-Contained Web Components

☆☆☆☆☆

By Eric Bidelman

Engineer @ Google working on Polymer, Chrome, and the web

TL;DR

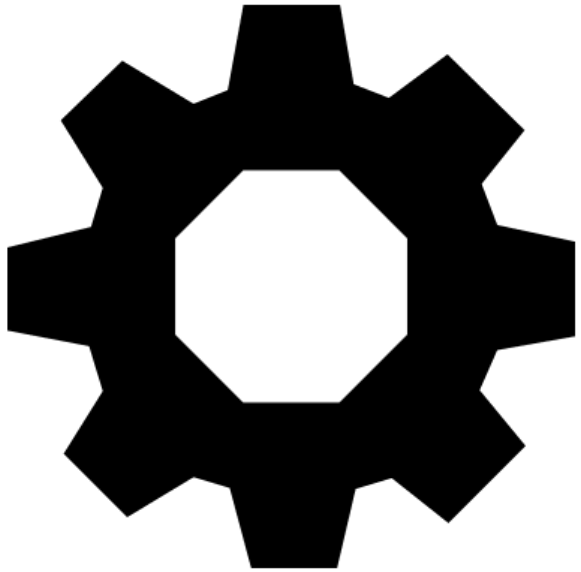
Shadow DOM removes the brittleness of building web apps. The brittleness comes from the global nature of HTML, CSS, and JS. Over the years we've invented an exorbitant number of tools to circumvent the issues. For example, when you use a new HTML id/class, there's no telling if it will conflict with an existing name used by the page. Subtle bugs creep up, CSS specificity becomes a huge issue (`!important` all the things!), style selectors grow out of control, and performance can suffer. The list goes on.

Shadow DOM fixes CSS and DOM. It introduces **scoped styles** to the web platform. Without tools or naming conventions, you can **bundle CSS with markup**, hide implementation details, and **author self-contained components** in vanilla JavaScript.

Introduction

★ **Note: Already familiar with Shadow DOM?** This article describes the new Shadow DOM v1 spec. If you've been using Shadow DOM, chances are you're familiar with the v0 version that shipped in Chrome 35, and the webcomponents.js polyfills. The concepts are the same, but the v1 spec has important API differences. It's also the version that all major browsers have agreed to implement, with implementations already in Safari Tech Preview and Chrome Canary. Keep reading to see what's new or check out the section on History and browser support for more info.

Shadow DOM is one of the four Web Component standards: HTML Templates, Shadow DOM, Custom elements and HTML Imports.



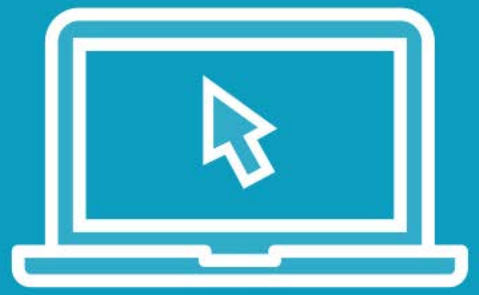
HTML Templates

Give you the ability to create reusable segments of HTML within your apps

The <template> element is not automatically rendered by the browser

Templates can be used to define markup for specific segments of an application and then rendered on-the-fly as required

Demo



Using HTML Templates

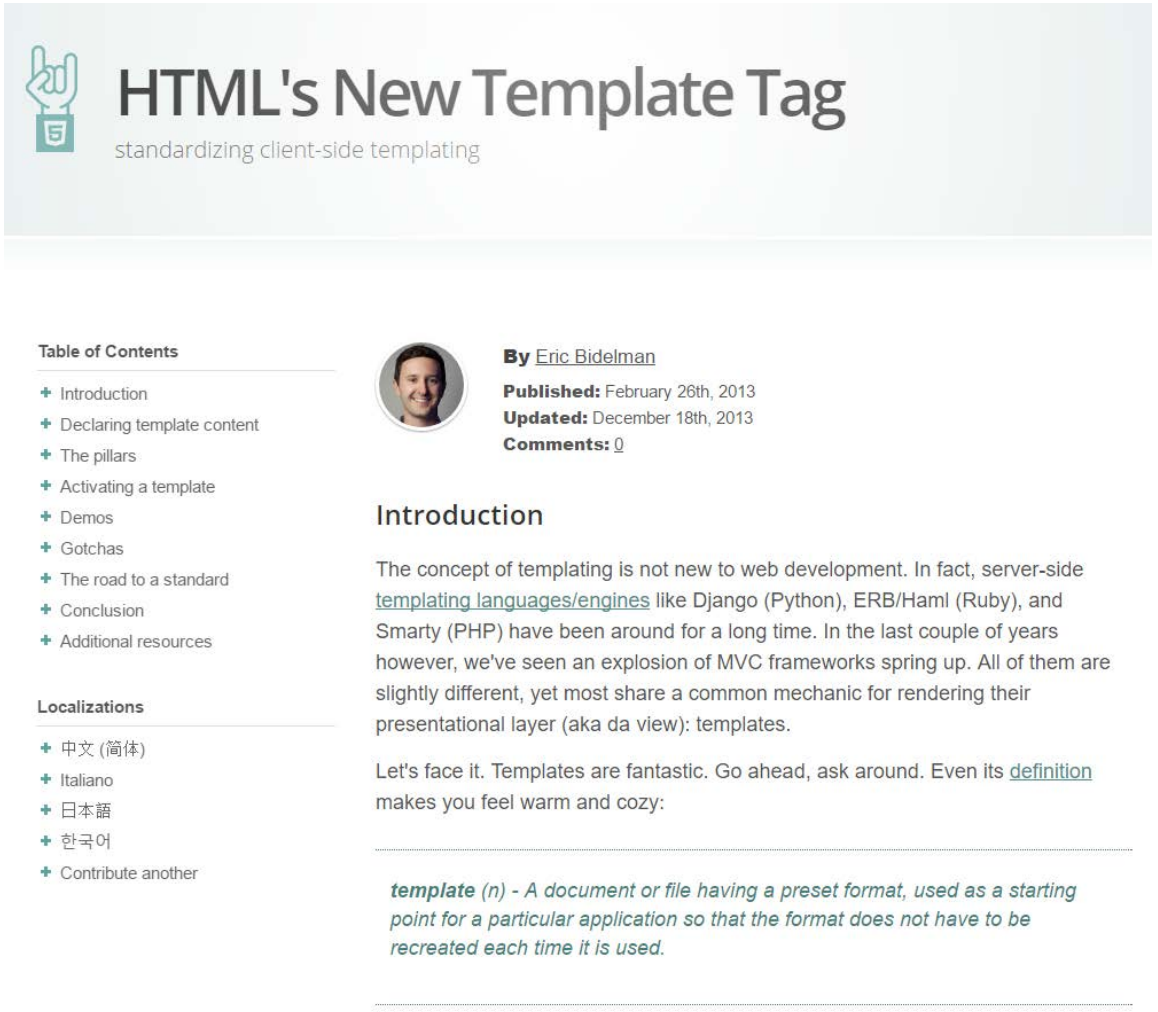
Creating a HTML template

Accessing and rendering it in JavaScript

A real-world example

HTML Templates: Further Reading

https://goo.gl/LQezsL



HTML's New Template Tag

standardizing client-side templating

Table of Contents

- Introduction
- Declaring template content
- The pillars
- Activating a template
- Demos
- Gotchas
- The road to a standard
- Conclusion
- Additional resources

Localizations

- 中文 (简体)
- Italiano
- 日本語
- 한국어
- Contribute another

By Eric Bidelman

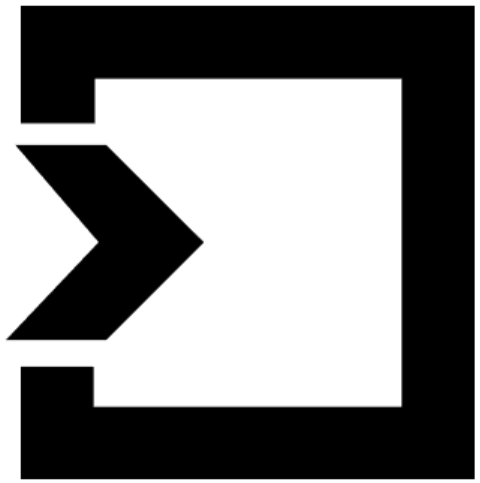
Published: February 26th, 2013
Updated: December 18th, 2013
Comments: 0

Introduction

The concept of templating is not new to web development. In fact, server-side templating languages/engines like Django (Python), ERB/Haml (Ruby), and Smarty (PHP) have been around for a long time. In the last couple of years however, we've seen an explosion of MVC frameworks spring up. All of them are slightly different, yet most share a common mechanic for rendering their presentational layer (aka da view): templates.

Let's face it. Templates are fantastic. Go ahead, ask around. Even its definition makes you feel warm and cozy:

> ***template*** *(n) - A document or file having a preset format, used as a starting point for a particular application so that the format does not have to be recreated each time it is used.*



HTML Imports

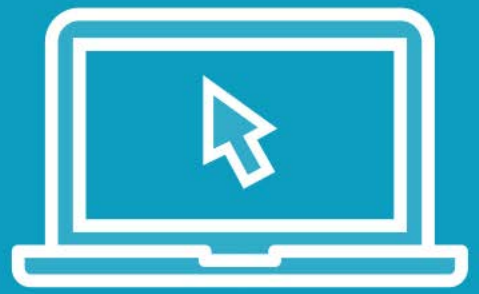
Include a HTML document just like a .css file

Often used in conjunction with HTML templates

Designed to be the packaging mechanism for Web Components

A contentious specification

Demo



Using HTML Imports

Import a HTML document into a page

Access the contents of the document from JavaScript

HTML Imports: Further Reading

https://goo.gl/D6fM5f

Table of Contents

- Why imports?
- Getting started
- Using the content
- Delivering Web Components
- Performance considerations
- Things to remember
- Conclusion

Localizations

- 中文 (简体)
- 日本語
- 한국어
- Contribute another

By Eric Bidelman

Published: November 11th, 2013
Updated: December 18th, 2013
Comments: 3

Why imports?

Think about how you load different types of resources on the web. For JS, we have `<script src>`. For CSS, your go-to is probably `<link rel="stylesheet">`. For images it's `<img>`. Video has `<video>`. Audio, `<audio>`.... Get to the point! The majority of the web's content has a simple and declarative way to load itself. Not so for HTML. Here's your options:

1. **`<iframe>`** - tried and true but heavy weight. An iframe's content lives entirely in a separate context than your page. While that's mostly a great feature, it creates additional challenges (shrink wrapping the size of the frame to its content is tough, insanely frustrating to script into/out of, nearly impossible to style).

Summary

What Web Components are

What fundamental problems Web Components solve

Which web standards give Web Components their super powers

How to use each of the Web Component standards

- Custom Elements
- Shadow DOM
- HTML Templates
- HTML Imports

Covered Web Component fundamentals and built a strong learning foundation