An Introduction to Web Components

Posted At : February 2, 2015 9:00 AM | Posted By : Jeffry Houser
Related Categories: JavaScript, Professional, Web Components

I wrote this up for the DotComIt newsletter in February, and wanted to post it here too. This is my introduction to Web Components.

What are Web Components?

Web Components are reusable browser widgets. They are being created as a set of standards by the W3C. In a practical sense, Web Components are like Angular directives, but independent of any singular JavaScript framework and built right into the browser. You use Web Components to bundle markup and styles into your own encapsulated, reusable entity.

There are four separate elements of web components:

- **Custom Elements**: A custom element is, basically, your own HTML tag.
- **HTML Imports**: An HTML Import is like an import in other programming languages, such as ActionScript or Java. They basically point to a piece of encapsulated code and make it available for use in the page.
- **Templates**: Templates allow you to define fragments of an HTML page for reuse.
- **Shadow DOM**: The Shadow DOM is used to define functional boundaries between the Web Component and the page that uses it. This, essentially, defines your API. It prevents conflicts between CSS and named HTML elements inside the Web Component and outside of it.

This article will server as an introduction to all these aspects.

One warning before we get into it; Web Components are a new technology and not supported by all browsers. These samples were built and tested in Chrome, but do not seem to work elsewhere.

Get Polymer

Polymer is a common framework used for building web components, and we're going to make use of Polymer in this article. You can find information about getting Polymer. You'll just need to put the polymer files in a web accessible directory. I put mine in "/bower_components/polymer/polymer.html".

Sample 0: No Web Component

I want to start this with a simple example. This sample will be a web page, that has some centered text on it:

```
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>Web Components: Sample 0</title>
    <style>
        .centeredContent{
            text-align: center;
        }
    </style>
</head>
<body>
<div class="centeredContent" >
```

```
    Hello World!
</div>
</body>
</html>
```

The "Hello World" text is put inside a div. The text is centered with the centeredContent style. This is about as simple as it comes. Your browser should look something like this:

Check out a running sample.

For the remainder of this article we are going to iterate over a web component that will center its text. This is a simple sample for example purposes, but can be much more complex.

Sample 1: Centered Text Web Component

Let's create our first Web Component. Let's start with a simple web page:

```
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>Web Components: Sample 1: A Simple Component</title>
</head>
<body>
</body>
</html>
```

Since we will be using the Polymer framework, the first step is to import it. Put the import statement in the head of the HTML Page:

```
<link rel="import" href="bower_components/polymer/polymer.html">
```

The import statement is based off the link tag, which I primarily use to import external style sheets. The syntax should be familiar to you. The rel says import and the href is the link to the HTML file that you're importing; in this case the polymer framework.

Next, define the web component with the polymer-element tag:

```
<polymer-element name="my-HelloWorld" noscript>
</polymer-element>
```

This tag can be added to the body of the page. The name of our component is specified as my-HellowWorld; but currently it does nothing. A template is needed for the component, so inside the polymer-element add the template tag:

```
<template>
</template>
```

So far so good! Next, add the content inside the template; which will be the style for centering content and the div that includes the content:

```
<style>
    .centeredContent{
        text-align: center;
    }
</style>
<div class="centeredContent" >
    Hello World inside component!
</div>
```

Finally, you can use the Web Component as if it were a standard HTML tag:

```
<my-HelloWorld ></my-HelloWorld>
<my-HelloWorld ></my-HelloWorld>
```

See this in action:

This example touched on three of primary elements of creating Web Components. We used an import to import the polymer framework. We used a template to create our own template. And we created a custom element that is used to create an instance of the web component.

Sample 2: Defining Content with Attributes

The Web Component we've created doesn't have much utility yet. It only displays items and does not give us any external control over the component. We can change that, by adding support for attributes to the component. We are going to add a single attribute, named text and that attribute will be used to determine the content that the component will center.

First, add the attributes attribute to the polymer-element:

```
<polymer-element name="my-HelloWorld" attributes="text" noscript>
```

When we want to display the text inside the div, we can do this:

```
<div class="centeredContent" >
    {{text}}
</div>
```

The double curly bracket syntax that is prevalent in Angular, so you should be familiar with it from my past writings. Flex uses a similar syntax--except with just a single curly bracket. The syntax does binding under the hood. It means that whenever the text value changes, the internal component display will also update. Here is the code that uses the

attribute:

```
<my-HelloWorld text="Hello World attribute!" ></my-HelloWorld>
<my-HelloWorld text="Hello World attribute 2!" ></my-HelloWorld>
```

The attribute we defined is used just like any other attribute on an HTML tag.

Try this in action, and you should see a screen similar to this:

Sample 3: Defining Content Body Text

In addition to adding attributes, you can also access the tag's body text from within the component. Start from sample 1 to create this modification. Inside the template you can reference the body text, or inner HTML, of the tag using the content tag:

```
<div class="centeredContent" >
    <content></content>
</div>
```

Then you specify the content to send into the tag like this:

```
<my-HelloWorld >Hello World Body Text!</my-HelloWorld>
<my-HelloWorld >Hello World Body Text 2!</my-HelloWorld>
```

The results can be shown here:

Sample 4: Separate Files

In real development you won't want to define your Web Components in the same file. Doing so limits reuse, which defeats the purpose of componentizing your code in the first place. Let's create a new file, component4.html and move our component into it:

```
<link rel="import" href="bower_components/polymer/polymer.html">

<polymer-element name="my-HelloWorld" attributes="text" noscript>
    <template>
        <style>
            .centeredContent{
                text-align: center;
            }
        </style>
        <div class="centeredContent" >
            {{text}}
        </div>
```

```
    </template>
</polymer-element>
```

This code contains the import of the polymer framework. The code includes the polymer definition of the my-HelloWorld component and also contains the text attribute and the template with our style and the centered content. The main page becomes much simpler now:

```
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>Web Components: Sample 3: A Component in a Different File</title>
    <link rel="import" href="Component4.html">
</head>
<body>
<my-HelloWorld text="Hello World Separate File" ></my-HelloWorld>
<my-HelloWorld text="Hello World Separate File 2" ></my-HelloWorld>

</body>
</html>
```

This new main page imports Component4.html, but does not import the Polymer framework, because that is imported directly into the component, and not used in the main page.

This is the sample here:

Sample 5: Shadow DOM Isolation

One thing I didn't want to go into details on, but did want to touch on was the shadow DOM. The shadow DOM means that the DOM inside the Web Component is isolated from the DOM of the main page. This will prevent styles from different components, or your main application, from interfering with the main component styles. Let's take a look at Component5.html:

```
<link rel="import" href="bower_components/polymer/polymer.html">

<polymer-element name="my-HelloWorld" attributes="text" noscript>
    <template>
        <style>
            .alignedContent{
                text-align: center;
            }
        </style>
        <div class="alignedContent" >
            {{text}}
        </div>
    </template>
</polymer-element>
```

This is pretty similar to the example we saw in sample 4. The main difference is that I named the style alignedContent instead of centeredContent. The text inside the component will center the content. But, let's use the same style inside the main index file to right align content:

```
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>Web Components: Sample 3: Shadow DOM Samples</title>
    <link rel="import" href="Component5.html">
    <style>
        .alignedContent{
            text-align: right;
        }
    </style>

</head>
<body>

<my-HelloWorld text="Hello World Shadow DOM Example" ></my-HelloWorld>

<div class="alignedContent" >
    Shadow DOM Example
</div>
</body>
</html>
```

To reiterate, the alignedContent style will center content in the Web Component, but will right align content in the main file.Run this code and you'll see:

You can see the top text is centered while the bottom text is right aligned. Two styles with the same name do not interfere with each other. This is why we love encapsulation.

When using browser developer tools by default the Shadow DOM element is hidden; however you can enable them if you want. This is a great debugging tool because if you're just using a component, you may want to focus on your own code without drilling down into the code of the component that you use. A parallel in the Flex world is that you often want to debug your own code, but do not need to drill into the code behind the Flex Framework code or code from complimentary frameworks such as Robotlegs or Swiz.

Final thoughts

Here are some good resources for reading up on web components:

- http://webcomponents.org/
- http://css-tricks.com/modular-future-web-components/
- http://webcomponents.org/presentations/components-101-intro-to-fundamental-changes-in-html/

Web Components, as an evolving standard, are not supported in all browsers yet. The polymer framework helps by providing support for browsers that don't have web components implemented natively. That is why most examples about web components use the framework. I like the concept, and this is something to watch in the future.