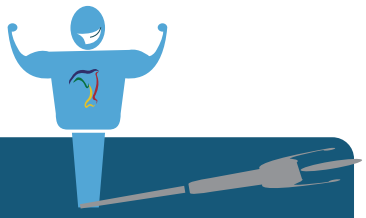
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

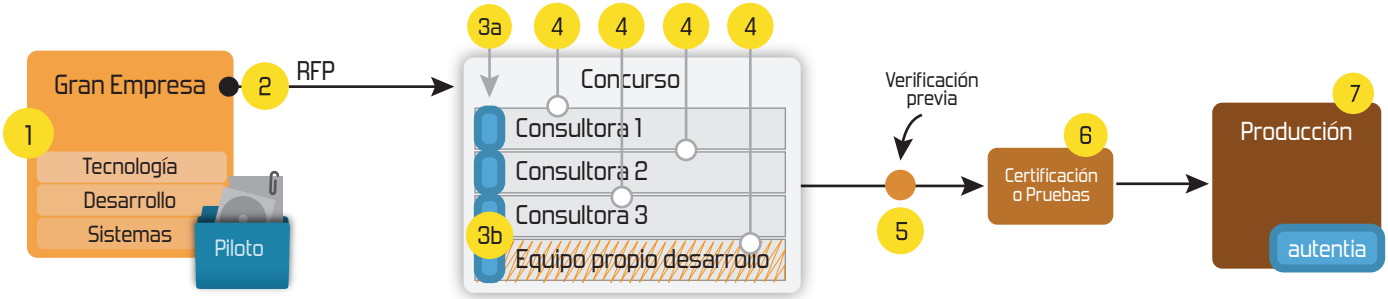
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.

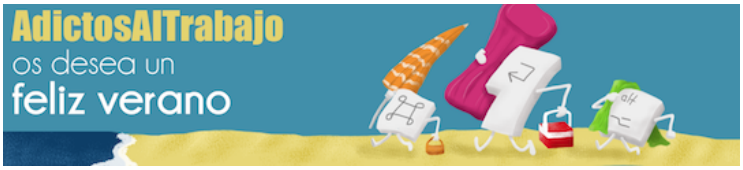


4. Cursos de formación (impartidos por desarrolladores en activo)

- Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery
- Gestor portales (Liferay)
- Gestor de contenidos (Alfresco)
- Aplicaciones híbridas
- Control de autenticación y acceso (Spring Security)
- UDDI
- Web Services
- Rest Services
- Social SSO
- SSO (Cas)
- JPA-Hibernate, MyBatis
- Motor de búsqueda empresarial (Solr)
- ETL (Talend)
- Dirección de Proyectos Informáticos.
- Metodologías ágiles
- Patrones de diseño
- TDD
- Tareas programadas (Quartz)
- Gestor documental (Alfresco)
- Inversión de control (Spring)
- BPM (jBPM o Bonita)
- Generación de informes (JasperReport)
- ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com



autentia
Soporte a desarrollo informático

Entra en Adictos a través de
E-mail
Contraseña
Entrar
Registrarme
Olvidé mi contraseña

Inicio Quiénes somos Formación Comparador de salarios Nuestros libros Más

» Estás en: Inicio Tutoriales Introducción a Polymer



Alberto Moratilla Ocaña

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero en Informática, especialidad en Ingeniería del Software. DEA en doctorado en Información, Documentación y Conocimiento. MBA en empresas de Nuevas Tecnologías. CQF.

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

Ver todos los tutoriales del autor

CURSO ONLINE DE JAVA
Domínalo y desarrola cualquier aplicación con este...

Fecha de publicación del tutorial: 2014-09-08 Tutorial visitado 292 veces Descargar en PDF

Introducción a Polymer

0.Índice de Contenidos

- 1. Entorno
- 2. ¿Qué es Polymer?
- 3. Instalación del entorno de trabajo y esqueleto de una aplicación Polymer con Yeoman
- 4. Estructura de una aplicación Web que usa Polymer
- 5. Creando nuestro propio componente de Polymer
- 6. Conclusiones

1. Entorno

Este tutorial está escrito siguiendo el siguiente entorno:

- Hardware: Acer Aspire One 753 (2 Ghz Intel Celeron, 4GB DDR2)
- Sistema Operativo: Ubuntu 14.04 LTS
- NodeJS V0.10.25

2. Introducción: ¿Qué es Polymer?

En los últimos tiempos se está produciendo un trasvase de funcionalidad desde el back-end al front-end. Hasta hace poco tiempo se procuraba que la lógica de la aplicación residiese en el servidor, utilizando el cliente para la visualización de la información e interacción básica que desencadenaba acciones que se llevaban a cabo en el servidor. En frameworks como JSF, Struts o Spring MVC el programador acaba creando páginas en servidor que son procesadas para ser enviadas al cliente sobre las que no se suele ejercer un control final.

Con la aparición de HTML5 con sus API y la mejora de JavaScript (ECMAScript), y de los intérpretes de los navegadores, en especial gracias al motor V8 de Google, se está trasladando el protagonismo de la aplicación hacia el navegador: aplicaciones en navegador potentes que se comunican con el servidor para el intercambio de datos (servicios REST) usando AJAX y JSON. Es lo que se conoce como Single Page Application o Web Application: una página Web HTML que contiene bibliotecas JavaScript que le permiten capacidades de comunicación con el servidor y de manipulación del DOM para crear una experiencia de usuario superior. De este modo el servidor queda para implementar la lógica de negocio y proveer de unos puntos de acceso a información utilizados por la aplicación Web.

Es fácil suponer que, al pasar la responsabilidad desde el servidor al cliente, la complejidad y tamaño del código Javascript aumenta de forma exponencial. Como sabemos, es mucho más fácil hacer un buen código estructurado en Java (Spring, Struts, Guice...) que hacerlo en Javascript. Surge por lo tanto la necesidad de tener algún tipo de framework que ayude a estructurar las aplicaciones en Javascript que corren en el cliente. Afortunadamente han surgido proyecos open source que facilitan estas labores como EmberJS, Backbone, Knockout o AngularJS como los más destacados.

Dentro de este desarrollo de nuevas tecnologías para el navegador pronto surge la necesidad de contar con un sistema para la reutilización de partes de las página web. Si has trabajando con HTML en el lado del servidor (JSP), es probable que hayas usado plantillas para dividir la interfaz y facilitar el trabajo. Si has usado JSF habrás podido comprobar cómo se va un paso más allá a través de los componentes que permiten de una manera sencilla crear páginas de un modo más lógico agregando elementos independientes que aunan tanto representación como modelo y comportamiento. En el nuevo entorno de HTML se busca lograr el mismo objetivo: disponer de un modo de especificar componentes Web que son reutilizables de un modo sencillo que son completos e interoperables: un menú, un mapa, un gráfico, un contenedor con paginación... Seguro que te suena.

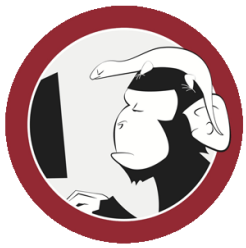
Los Web Components es el estándar propuesto por el grupo de trabajo de WebApps de W3C y que todavía está en fase de definición. Dentro de esta iniciativa se encuentra Polymer, que junto con componentes X-Tag de Mozilla son las implementaciones más destacadas de los componentes Web.

Los Web Components se basan en los siguientes pilares:

- **Templates**: son fragmentos de HTML que representan al componente de modo que cuando se haga referencia al componente se usará esta plantilla.

Catálogo de servicios Autentia





Síguenos a través de:

Últimas Noticias

» Portales, gestores de contenidos documentales y desarrollos a medida

» Comentando el libro Start-up Nation, La historia del milagro económico de Israel, de Dan Senor & Salu Singer

» Screencasts de programación narrados en Español

» Sorteo de entradas para APIdays Mediterranea

» Concurso del Día de la Madre:

Histórico de noticias

Últimos Tutoriales

» Emmet.io - el toolkit esencial para los desarrolladores Web

» Taller Lightbot programación para niños.

» Mockear la capa back con Dyson

» Automatizando los Smoke Test con TestLink y Jenkins

» Cómo hacer testing automático de un applet Java

- **HTML Import**: es la posibilidad de incluir un HTML dentro de intro mediante un tag <link rel="import" href="include.html">. De este modo el template de un Web Component formará parte del otro.
- **Shadow DOM**: permite encapsular el subarbol DOM de un componente, de modo que la interacción desde fuera y hacia dentro está controlada. Es importante para que los componentes no entren en conflictos al formar parte de la misma página.
- **Custom Elements**: para la definición de Tags específicos que representan a los web components que se han creado. En este tutorial haremos el componente contador que estará representado por el tag <c-contador> por ejemplo.

Como puedes imaginar todos estos elementos forman parte de las nuevas especificaciones que van incoporándose a HTML5 y a los navegadores. Aunque se intenta mantener la compatibilidad entre todos ellos no esperes que funcionen en versiones viejas o muy específicas. Antes de usar esta tecnología deberías repasar los requisitos funcionales.

3. Instalación del entorno de trabajo y esqueleto de una aplicación Polymer con Yeoman

Aunque es relativamente sencillo crear una aplicación Polymer desde el principio, es mucho más fácil si disponemos de un esqueleto de aplicación en el que basar nuestro desarrollo. Una de las posibilidades es bajar de su página oficial un zip con este esqueleto. La otra es usar programas especializados que nos permiten descargar este esqueleto, los ficheros de dependencias y realizar otras tareas del ciclo de desarrollo de la aplicación.

Una de las herramientas que nos permiten gestionar el ciclo de desarrollo de una aplicacion HTML5 es Yeoman, que se compone de otras tres herramientas: Bower para la gestión de paquetes, Grunt para generar, previsualizar y testear nuestros proyectos y Yo para gestionar ambos.

Vamos con la creación de un proyecto Polymer usando Yeoman en un Ubuntu 14.04 desktop recién instalado. Si ya tienes Yeoman instalado en tu equipo deberías saltarte esta parte. Si quieres bajarte el proyecto en .zip desde la página de Polymer también puedes saltar esta sección.

Para usar Yeoman necesitamos tener instalado NodeJS en nuestro sistema. Como estamos usando una versión nueva de Ubuntu, que está basada en Debian, debemos instalar una versión compatible de NodeJS que utilice como ejecutable "node". La versión estándard de node para Debian tiene como comando "nodejs" debido a un cambio de nombre. Desafortunadamente con este cambio de nombre no funciona la versión estandar, por lo que hay que instalar una retrocompatible

```
view plain print ?
01. sudo apt-get install nodejs-legacy
```

Una vez tenemos instalado nodejs en nuestro sistema podemos instalar Yeoman utilizando el gestor de paquetes de node npm.

```
view plain print ?
01. sudo npm install -g yo
```

Para que Yeoman pueda generar un proyecto de tipo Polymer necesitamos descargar el generador específico. Si no sabemos su nombre, podemos utilizar la siguiente instrucción para listar todos los disponibles en el sistema:

```
view plain print ?
01. npm search yeoman-generator
```

Haciendo un grep o mirando la salida, podremos ver ue existe un generador que se llama generator-polymer cuya descripción corresponde con lo que estamos buscando. Lo instalamos de modo global ejecutando:

```
view plain print ?
01. sudo npm install generator-polymer -g
```

Ahora nuestro Yeoman será capaz de generar un proyecto Polymer. Vamos a ello.

Creamos un directorio en nuestro ordenador con el nombre y ubicación que creamos conveniente.

```
view plain print ?
01. mkdir developer/polymer
02. cd developer/polymer
```

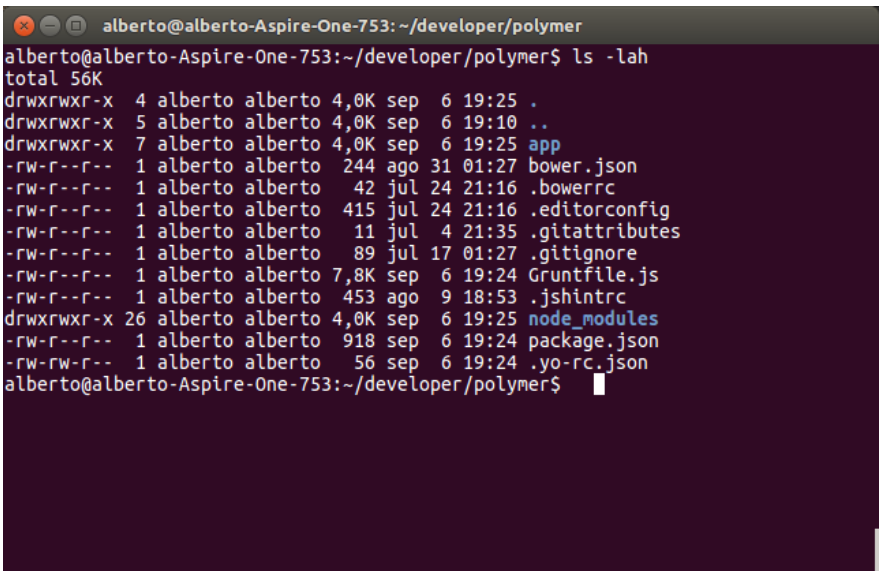
Y ejecutamos la orden para que Yeoman genere el esqueleto del proyecto.

```
view plain print ?
01. yo polymer
```

Aparecerá una pantalla interactiva preguntándonos qué modulos queremos incluir en el proyecto. Son módulos accesorios como las librerías de elementos o el soporte para SASS.



Puedes instalar todos los elementos si quieres, especialmente las librerías de componentes que aparecen al comienzo. Al concluir el proceso deberiamos tener un contenido como el siguiente:



Además de los ficheros de la aplicación que están en el directorio "app", tenemos una serie de ficheros de configuración para poder gestionar las dependencias de la aplicación, su generación y despliegue a través de las herramientas de las que se compone Yeoman: bower.json para dependencias que se tratan con Bower y Gruntfile.js para el despliegue y otras tareas (al estilo Maven de Java).

Solo falta arrancar la aplicación que viene de ejemplo básico que se ha descargado. Es muy sencillo, usando Grunt podremos lanzar un servidor HTTP en el que probar la aplicación: en el mismo directorio raiz de la aplicación ejecutamos (no olvidar ejecutar con sudo, de otro modo dará un error):

```
view plain print ?
01. sudo grunt serve
```

Antes de continuar con la aplicación un detalle: si a la hora de crear el proyecto con Yeoman le has indicado que deseas instalar los móudlos de Sass, será necesario tener en nuestro sistema Ruby y la gema de Sass correspondiente. Es muy fácil de instalar. Simplemente ejecuta:

```
view plain print ?
01. sudo apt-get install ruby
```

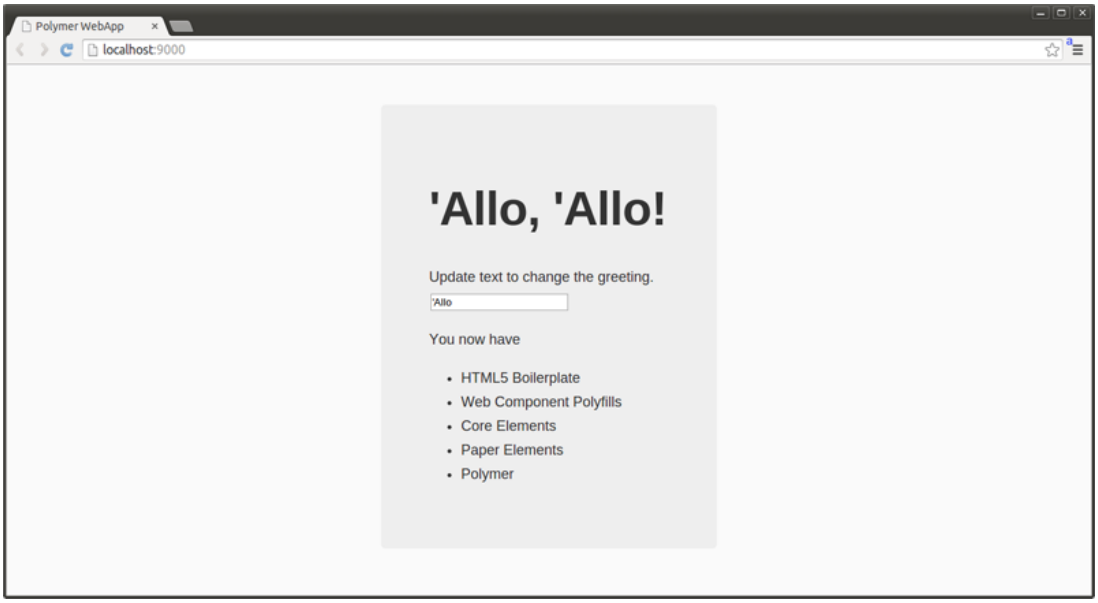
Y cuando esté Ruby instalado, instalamos la gema de Sass:

```
view plain print ?
01. sudo gem install sass
```

Ahora ya es más que posible que nuestro servidor grunt funcione correctamente. Repetimos:

```
view plain print ?
01. sudo grunt serve
```

Si todo ha ido bien, debería abrirse nuestro navegador por defecto con la página localhost:9000 y mostrarnos el contenido de la aplicación por defecto:



Usar Grunt como servidor, entre otras muchas ventajas, nos permite editar un fichero y que el navegador refresque automáticamente la página en cuanto detecte que el fichero que estamos editando se ha guardado. Si cuentas con dos pantallas, una para el editor y otra para ver el navegador, verás su utilidad.

4. Estructura de una aplicación Web que usa Polymer

Podemos abrir el directorio en el que se ha generado el proyecto de Polymer mediante un editor como por ejemplo Sublime Text. Si accedemos al directorio app podremos ver los ficheros que componen la aplicación. El resto de elementos que están a la misma altura que el directorio app son ficheros de configuración y otros directorios para Grunt, Bower y los destinos de estos.

En primer lugar podemos abrir el fichero index.html, que contiene el código de la página que compone la aplicación Web de ejemplo:

```
view plain print ?
01.     <!doctype html>
02. <html>
03.
04. <head>
05.   <meta charset="utf-8">
06.   <meta http-equiv="X-UA-Compatible" content="IE=edge">
07.   <title>Polymer WebApp</title>
08.   <meta name="description" content="">
09.   <meta name="viewport" content="width=device-width, minimum-scale=1.0, initial-scale=1.0, user-
    scalable=yes">
10.   <!-- Place favicon.ico and apple-touch-icon.png in the root directory -->
11.
12.   <!-- build:css styles/main.css -->
13.   <link rel="stylesheet" href="styles/main.css">
14.   <!-- endbuild-->
15.
16.   <script src="bower_components/platform/platform.js"></script>
17.   <!-- build:vulcanized elements/elements.vulcanized.html -->
18.   <link rel="import" href="elements/elements.html">
19.   <!-- endbuild-->
20. </head>
21.
22. <body unresolved>
23.
24.   <div class="hero-unit">
25.     <yo-greeting></yo-greeting>
26.     <p>You now have</p>
27.     <yo-list></yo-list>
28.   </div>
29.
30.   <!-- build:js scripts/app.js -->
31.   <script src="scripts/app.js"></script>
32.   <!-- endbuild-->
33. </body>
34.
35. </html>
36.
```

Como podemos ver se trata dé un código muy sencillo en el que, además de los tags meta, hojas de estilo y etiquetas básicas que componen una página web clásica, destacan una serie de elementos:

- Unos "include" que proporcionan a la página la funcionalidad necesaria para operar con Polymer. Estos include son:
  - La referencia a la plataforma de Polymer: <script src="bower_components/platform/platform.js"></script>
  - El script de inicio de la aplicación: <script src="scripts/app.js"></script>
  - La referencia a los componentes que se utilizan: <link rel="import" href="elements/elements.html">
- Unos tags un tanto "extraños" que no reconocemos en los habituales de HTML5. Si tienes algo de experiencia con AngularJS te sonará el hecho de tener etiquetas fuera del estandard. Veremos que es la forma de invocan los elmentos o componentes de Polymer, y dedicaremos un apartado más adelante.

Para ver la estructura de la aplicación, nos centraremos en el último de los include indicados:

```
view plain print ?
01. <link rel="import" href="elements/elements.html">
```

Si te fijas es un tag tipo link cuyo atributo rel tiene el valor "import". Esto es una de las características que los nuevos navegadores soportan, lo que permiten cargar el código HTML dentro de la página padre. Este archivo que se incluye

"elements/elements.html" contiene las referencias a los elementos (o componentes) de polymer que se van a invocar en la aplicación utilizando los tags de HTML que los representan. Veamos su contenido:

```
view plain print ?
<link rel="import" href="yo-list/yo-list.html">
<link rel="import" href="yo-greeting/yo-greeting.html">
<link rel="import" href="c-contador/c-contador.html">
```

Como puedes imaginar solamente contienen includes de otros ficheros .html que esta vez sí que contienen los ficheros de los tags. De nuevo se vuelve a emplear la fórmular del include de tipo "import". No es imprescindible utilizar este tipo de estructura ya que se podrían haber incluído directamente los tags, pero sí que ayuda a mantener cierto orden y estructura en nuestra aplicación Web. Seguro que ya estás imaginando otras situaciones en las que puedes emplear esta nueva característica de imports.

Por lo tanto tenemos un fichero .html por cada uno de los componentes que vienen en el ejemplo. Uno para yo-list y otro par yo-greeting. A estas alturas ya deberías ir relacionándolos con los tags que vimos en el fichero index.html. Si abrimos el fichero yo-greetings.html nos encontraremos con la estructura básica de un elemento de polymer.

```
view plain print ?
    <link rel="import" href="../../bower_components/polymer/polymer.html">

    <polymer-element name="yo-greeting"  attributes="">
    <template>
        <link rel="stylesheet" href="yo-greeting.css">
        <h1>{{ greeting }}, {{ greeting }}!</h1>
        <span>Update text to change the greeting.</span>
        <input type="text" value="{{ greeting }}">
    </template>
    <script>
    (function () {
        'use strict';

        Polymer({
            greeting : '\'Allo'
        });

    })();
    </script>
</polymer-element>
```

Podemos destacar los siguientes aspectos del código:

- Import de la biblioteca de polymer.html, que en realidad hacer referencia a polymer.js que es la verdadera biblioteca.
- La definición del nombre del elemento de polymer "yo-greeting".
- El template o representación del HTML del elemento que está formando por un h1, un span y un input. Además incluye su propia hoja de estilos que ha sido especificada en un fichero independiente.
- Una sección de script que define el modelo y el controlador de ese elemento.

Como puedes ver, un elemento de Polymer o Web Component está completamente autocontenido, es decir, tiene la vista, tiene el modelo de datos y tiene el controlador o acciones que operan sobre esos datos. Por tanto es totalmente independiente y reutilizables.

Para utilizarlo es muy sencillo. Si volvemos al fichero index.html recordarás que se hace un include de elements.html que a su vez hace un include de yo-greetings.html. Con estos includes ya podemos utilizarlo. Y para ello simplemente utilizaremos un tag con el nombre definido en la cabecera polymer-element que será <yo-greeting>. Polymer sustituirá este tag por el contenido especificado para el componente. Realmente sencillo.

Creando nuestro propio componente de Polymer

Una vez hemos visto el código de ejemplo que hemos descargado con Yeoman seguro que ya tienes en la cabeza cómo crear tu propio componente. Vamos a hacer nuestro propio elemento de Polymer sencillo, que será un contador con un botón para incrementar el número y otro para disminuir. Por tanto se compondrá de:

- Template: que contiene dos botones y un input par ver el elemento.
- Controlador: que inicializa el valor y tiene una función par subir el elemento y otra para bajarlo.

Si seguimos los pasos indicados en la descripción anterior sobre la estructura de la aplicación de Polymer básica, podremos deducir que tendremos que hacer:

- Crear un directorio y fichero en elements para nuestro nuevo tag. Como se llamará c-contador, crearemos elements/c-contador/c-contador.html. Copiaremos el ya existente yo-greetings.html para ahorrarr trabajo y lo editaremos.
- Editar elements.html para incluir la referencia al nuevo fichero que describe nuestro componente.

Con estos dos sencillos pasos ya estaremos preparados para invocar el componente con el tag dentro de index.html.

Creando el fichero c-contador.html

Este fichero define el componente por completo, tanto apariencia como datos y comportamiento. Nos basamos en uno ya existente. El contenido será el siguiente:

```
view plain print ?
<link rel="import" href="../../bower_components/polymer/polymer.html">


<polymer-element name="c-contador">
<template>
    <link rel="stylesheet"  href="c-contador.css">
    <button class="boton" on-click="{{decrementa}}">-</button>
    <input name="contador" type="text" value="{{numero}}" />
    <button href="" class="boton" on-click="{{incrementa}}">+</button>
</template>
<script>
(function() {
    'use strict';


    Polymer('c-contador',{
        numero:0,
        decrementa:function()
        {
            this.numero--;
        },
```

```
        incrementa:function()
        {
            this.numero++;
        }
    });

})();
</script>

</polymer-element>
```

Los puntos importantes de nuestro componente o elemento de Polymer son:

1. El nombre del componente indicado en el tag *polymer-element* que será "c-contador". A la hora de elegir el nombre **tenemos que tener en cuenta que este tiene que llevar un guión**. En nuestro elemento no podríamos llamarlo "contador" porque no lo lleva (daría Uncaught Error: document.registerElement: first argument ('name') must contain a dash ('-'). Argument provided was 'contador'.). Lo solucionamos llamándolo "c-contador" por ejemplo. Se ha usado "c-" de forma arbitraria. Convendría usar un nombre de la librería o algo que consideremos significativo.

2. El apartado "template" donde se define la apariencia. En primer lugar tenemos la definición de la hoja de estilos en un fichero externo llamado c-contador.css. Es una buena práctica tenerlo definido fuera, aunque se hubiera podido incluir perfectamente en el mismo código.

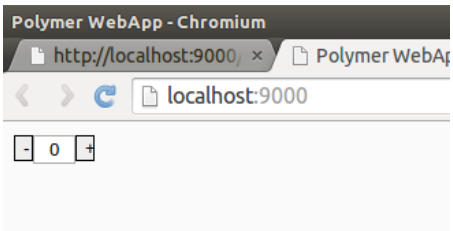
   Por otra parte tenemos dos botones definidos con el tag de HTML "button". Si te fijas en detalle, ambos tienen refrencia a la clase "boton" que está declarada en el fichero CSS. Por otra parte también tienen referencia al atributo *on-click*. Este atributo es la forma que tiene Polymer de especificar un callback para los eventos de sus elementos. Fíjate en que se llama diferente al clásico atributo onclick de HTML. Se define la función que se empleará como callback entre "double moustache" que es la notación que usa polymer para hacer referencia al modelo y controlador.

   Finalmente tenemos un input que muestra el valor. Se vuelve a emplear la notación "double moustache" que relaciona una variable con el input usando un binding de tipo "two-way". Si has usado AngularJS te sonará: un cambio de un dato en el modelo se refleja de forma automática en el DOM y viceversa. Es decir, una variable representada en un input de texto, al cambiar el valor en el input, cambiar en el modelo, y viceversa, un cambio con Javascript en esa variable hace que se cambie automáticamente en el input. Todo esto sin que el programador tenga que hacer nada.

3. La definición del controlador mediante la seccion "script" del elemento, que contiene código JavaScript. Si observas bien, por una parte tenemos la definición de la función general que es común para todas las instancias del componente, y dentro de ella la llamada a Polymer pasando como parámetro el nombre del elemento. Este fragmento de código es local a cada una de las instancias del componente. ¡Ten cuidado dónde declaras las variables!. Los miembros de Polymer no son otra cosa que la definición de los datos y funciones, por lo que está intimamente ligado con la vista. Ahí podemos ver el valor "numero" que se muestra en el inputo y las funciones que hacen de callback para los eventos de pulsación de los botones ("incrementa" y "decrementa").

   También es posible sacar a un archivo fuera todo el código de script del elemento de polymer, añadiendo el atributo src="path/to/tagname.js" al tag script, y así podremos separar el comportamiento de la representación. Al igual que las hojas de estilo seguramente querramos mantener la nomenclatura c-contador.js y el directorio en el que se encuentra.

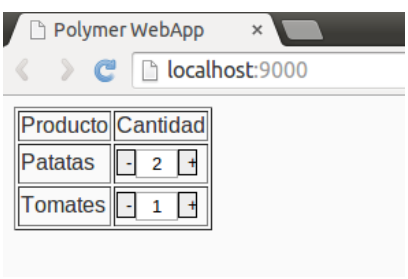
Para utilizarlo simplemente tenemos que ir al fichero raíz de la aplicación (index.html) e invocarlo dentro del código mediante el tag . Si eliminamos la referencia a los otros dos elementos, yo-greeting y yo-list, tendremos una página vacía con el contador. Si pulsamos sobre los botones podremos ver cómo sube o baja el contador. También podremos cambiar su valor introduciéndolo manualmente en el input.



Lo que acabamos de hacer es poner el componente aislado. Imagina ahora que tenemos una tabla de productos y cantidades y queremos reutilizar el contador. Es muy fácil, en la segunda columna, por cada elemento podremos una llamada al tag c-contador. Ahora todo tiene mucho más sentido:

```
view plain  print  ?
<table border="1">
    <tr>
        <td>Producto</td>
        <td>Cantidad</td>
    </tr>
    <tr>
        <td>Patatas</td>
        <td><c-contador id="contadorPatatas"></c-contador></td>
    </tr>
    <tr>
        <td>Tomates</td>
        <td><c-contador id="contadorTomates"></c-contador></td>
    </tr>
</table>
```

Obtenemos la siguiente imagen:



¿Cómo podemos interactuar con el componente para saber su valor?

La respuesta es muy sencilla: utilizando JavaScript podemos acceder a las propiedades que el componente expone. Para

saber cuántos tomates hay seleccionados en un momento determinado accedemos al valor de su "numero":

```
view plain print ?
01. var numeroTomates=document.querySelector('#contadorTomates').numero;
```

Esto es de vital importancia a la hora de que otros elementos y componentes de nuestra aplicación Web puedan interactuar con los elementos de polymer: si tenemos que utilizar el valor del contador para, por ejemplo, para indicar el número de artículos en una cesta de compra de un e-commerce, simplemente tomaremos el valor del modelo usando JavaScript.

Finalmente vamos a ver cómo podemos incluir parámetros en la llamada al componente como atributos en el tag. Usando los atributos específicos es posible tomar información de los atributos del tag que el usuario haya especificado. En el ejemplo del contador, "numero" es la variable que se contiene la información y que se está manipulando. Podemos indicar que esa variable sea también un atributo. En este caso toma el valor especificado por el programador en el código HTML, sobreescribiendo la indicada en el código.

```
view plain print ?
01. <polymer-element name="c-contador" attributes="numero">
```

Y en el código HTML:

```
view plain print ?
01. <c-contador numero="2"></c-contador>
```

De este modo el valor por defecto para esa instancia de elemento de polymer será 2. En el ejemplo de la tabla de la frutería se pueden así inicializar los valores que provengan de otras manipulaciones anteriores de la cesta de la compra.

Los atributos se pueden utilizar para muchas otras operaciones, relacionadas con atributos de clase, apariencia, o con utilidades dentro del código script que tiene el componente.

Polymer no solamente da soporte a la creación de elementos reutilizables en nuestro código como hemos visto. También incluye soporte para características que permiten crear aplicaciones Web multidispositivo y que pretenden tener una experiencia de usuario similar a las aplicaciones nativas. Por ejemplo:

- **Layout**: incluyendo atributos para soportar CSS3 Flexible Box, para poder adecuar la posición y distribución de los elementos para diferentes tamaños. (https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Flexible_boxes).
- **Estilos**: con funcionalidades para controlar la apariencia de la aplicación en el proceso de carga de elementos, así como el soporte de característica de estilo de Polyfill.
- **Soporte para gestos**: mejorando la experiencia en los dispositivos táctiles con los eventos de tipo "touch".

6. Conclusiones

En este tutorial se ha visto una breve introducción a Polymer, una tecnología surgida de la necesidad de dotar a las aplicaciones Web de modularidad que permita un desarrollo sencillo y ágil de las nuevas aplicaciones que se están realizando en este entorno. El actual desarrollo de los éstándares de HTML5 y la implicación de los principales actores del mercado Web (entre ellos Google), está haciendo posible grandes avances en estos últimos tiempos.

Polymer no es más que la punta de flecha de los Web Components, una tecnología que todavía está en desarrollo, pero que por su alcance y características está llamada a convertirse en el nuevo modo en el que se desarrollen las aplicaciones en el navegador, tanto de dispositivos clásicos como los PC como de los nuevos protagonistas del acceso a internet: móviles, tablets y derivados.

Aunque es una tecnología impactante y con grandes posibilidades, no debemos olvidar que todavía está en una fase temprana de desarrollo y que sus niveles de compatibilidad no lo hacen apto para todos los entornos. Es altamente dependiente del grado de desarrollo de los navegadores que se emplean. No obstante, sí que merece la pena hacer un seguimiento cercano, tanto por las posibilidad futuras como por los esfuerzos que están realizando los equipos de desarrollo de Google, tanto de Chrome como de desarrollo de AngularJS y Polymer, para que los programadores y usuarios podamos disfrutar de esta tecnología de forma estable en el menor tiempo posible.

Recuerda que esto solamente ha sido una breve introducción. Puedes encontrar artículos y vídeos en su Web oficial: http://www.polymer-project.org/

A continuación puedes evaluarlo:

Regístrate para evaluarlo

Por favor, vota +1 o compártelo si te pareció interesante

Share | 8+1 0

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

» Registrate y accede a esta y otras ventajas **«**

SOME RIGHTS RESERVED Esta obra está licenciada bajo licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5

IMPULSA Impulsores Comunidad ¿Ayuda?

---
sin clicks

0 personas han traído clicks a esta página

+ + + + + + + +

powered by karmacracy

Copyright 2003-2014 © All Rights Reserved | Texto legal y condiciones de uso | Banners | Powered by Autentia | Contacto

XHTML 1.0 CSS XML RSS XML ATOM