actibyti] a netmind social project

MUSTACHE

Autor: Ricardo Ahumada

GOBIERNO DE ESPAÑA
MINISTERIO DE ENERGÍA, TURISMO Y AGENDA DIGITAL
red.es
ESTRATEGIA DE EMPRENDIMIENTO Y EMPLEO JOVEN garantía juvenil
Agenda Digital para España
UNIÓN EUROPEA
Fondo Social Europeo
"El FSE invierte en tu futuro"

ÍNDICE DE CONTENIDOS

2 Mustache

Contexto

- Para generar modificaciones complejas en el DOM (como cuando recibimos una respuesta asíncrona por AJAX) es necesario crear el HTML y luego insertarlo en el elemento adecuado.
- Sin sistemas de plantilla, es necesario crear los trozos de html y dinámicamente insertarlos en el DOM utilizando javascript.
- Una forma común de hacerlo es especificar los elementos html en una String y luego modificar la propiedad InnerHTML del elemento o usar el método html() en jquery.

```
var dynamic_html = "<div><span>Highlighted</span><span>Author</span></div>";

document.getElementByID("container").innerHTML = dynamic_html;
```

Mustache

- Mustache es un sistema de plantillas “logic-less” para lenguajes de programación como javascript, ruby, python, php y java.
- La librería es opensource y puede descargarse de la página oficial en github
  - https://github.com/janl/mustache.js
- Mustache proporciona plantillas y vistas como base para crear HTML dinámico.
  - Las vistas contienen los datos json que se incluirán en las plantillas.
  - Las plantillas contienen el html de presentación o los datos con las etiquetas de plantilla para incluir datos de vista.

Instalando y usando Mustache

Instalar mustache

- Descarga el JS de mustache de su espacio github
  - https://github.com/janl/mustache.js
  - Descarga el archivo **mustache.min.js** y guárdalo en tu proyecto
- Incorpora mustache en tu aplicación JS

```
<script type="text/javascript" src="mustache.min.js" ></script>
```

Uso de Mustache

- Para usar mustache es necesario un objeto

```
var view = {
  title: "Joe",
  calc: function () {
    return 2 + 4;
  }
};
```

- Usaremos mustache para renderizar el objeto, usando el método render contra una plantilla

```
var output = Mustache.render("{{title}} spends {{calc}}", view);
```

- Los elementos marcados entre código mustache {{valor}} se reemplazarán por la invocación del atributo o método correspondiente del objeto

```
{{title}} <== view.title
```

```
{{calc}} <== view.calc
```

API mustache

- Renderizar un objeto contra una plantilla

```
Mustache.render(
  template  : String,
  view      : Object,
  partials? : Object,
) => String
```

- Parsear tags en una plantilla

```
Mustache.parse(
  template            : String,
  tags = ['{{', '}}']  : Tags,
) => String
```

- Interface Tag

```
interface Tags [String, String]
```

Plantillas mustache

- Una plantilla de mustache es un string que contiene etiquetas mustache.
- Las etiquetas están indicadas por los bigotes dobles que los rodean. {{persona}} es una etiqueta, al igual que {{# persona}}.
- En ambos ejemplos nos referimos a la persona como la clave de la etiqueta.
- Hay varias técnicas que se pueden utilizar para cargar plantillas y entregarlas a bigotes.js:
  - Templates include
  - Templates externas

Templates include

- Incluyendo la plantilla en el archivo HTML estático para evitar cargar plantillas por separado.
- Ejemplo usando jQuery
  - El HTML con la definición de la plantilla

```
<!DOCTYPE HTML>
<html>
<body onload="loadUser()">
<div id="target">Loading...</div>
<script id="template" type="x-tmpl-mustache">
Hello {{ name }}!
</script>
</body>
</html>
```

  - El JS de renderización

```
function loadUser() {
  var template = $('#template').html();
  Mustache.parse(template);   // optional, speeds up future uses
  var rendered = Mustache.render(template, {name: "Luke"});
  $('#target').html(rendered);
}
```

Templates externos

- Si las plantillas residen en archivos individuales, se pueden cargar asincrónicamente y renderizarlos
- Ejemplo usando jQuery

- La plantilla externa

```
<h1>{{name}}</h1>
```

- El JS de renderización

```
function loadUser() {
  $.get('template.htm', function(template) {
    var rendered = Mustache.render(template, {name: "Luke"});
    $('#target').html(rendered);
  });
}
```

Variables

- El tipo de etiqueta más básico es una variable simple. Una etiqueta {{name}} representa el valor de la clave de name en el contexto actual. Si no hay tal clave, no se procesa nada.
- Todas las variables son *HTML-escape* por defecto. Si se quiere procesar HTML sin escapar, se debe yar el triple bigote: {{{name}}}. También puede utilizar & para des-escapar una variable.
- Si desea que {{name}} no se interprete como una etiqueta de bigote, sino que aparezca exactamente como {{name}} en la salida, debe cambiar y restaurar el **delimitador predeterminado**.
- La notación de punto JavaScript se puede utilizar para acceder a atriburos que son propiedades de objetos en una vista.

Ejemplo variables

```
{
  "name": "Chris",
  "company": "<b>GitHub</b>"
}
```

```
* {{name}}
* {{age}}
* {{company}}
* {{{company}}}
* {{&company}}
{{=<% %>=}}
* {{company}}
<%={{ }}=%>
```

```
* Chris
*
* &amp;lt;b&amp;gt;GitHub&amp;lt;/b&amp;gt;
* <b>GitHub</b>
* <b>GitHub</b>
* {{company}}
```

Ejemplo variables II

```
{
 "name": {
   "first": "Michael",
   "last": "Jackson"
 },
 "age": "RIP"
}
```

```
* {{name.first}}
{{name.last}}
* {{age}}
```

```
* Michael Jackson
* RIP
```

Secciones

- Las secciones procesan bloques de texto una o más veces, dependiendo del valor de la clave en el contexto actual.
- Una sección comienza con una almohadilla (#) y termina con una barra (/).
  - {{#person}} comienza una sección de persona, mientras que {{/person}} la termina.
  - El texto entre las dos etiquetas se denomina "bloque" de esa sección.
- Si la clave de persona existe y no es *null*, *undefined* o *false* y; no es una lista vacía, el bloque se renderizará una o más veces.

```
{{#person}}
    Subplantilla de sección
{{/person}}
```

Ejemplo Sección

```
{
 "stooges": [
   { "name": "Moe" },
   { "name": "Larry" },
   { "name": "Curly" }
 ]
}
```

```
{{#stooges}}
<b>{{name}}</b>
{{/stooges}}
```

```
<b>Moe</b>
<b>Larry</b>
<b>Curly</b>
```

Looping

- Cuando se realiza un bucle sobre un array de cadenas, Un "." puede usarse para referirse al elemento actual de la lista.

```
{
  "musketeers": ["Athos", "Aramis",
"Porthos", "D'Artagnan"]
}
```

```
{{#musketeers}}
* {{.}}
{{/musketeers}}
```

* Athos
* Aramis
* Porthos
* D'Artagnan

Ejemplos mustache

- Examina los archivos de ejemplo de mustache

2 Handlebars

Handlebars

- Handlebars es un motor de plantilla sin lógica que genera dinámicamente HTML.
- Es una extensión de mustache con algunas características adicionales.
- Mustache es completamente logic-less, Handlebars añade una lógica mínima gracias al uso de algunos helpers (if, with, unless, each …).
- De hecho, podemos decir que Handlerbars es un superconjunto de bigote.

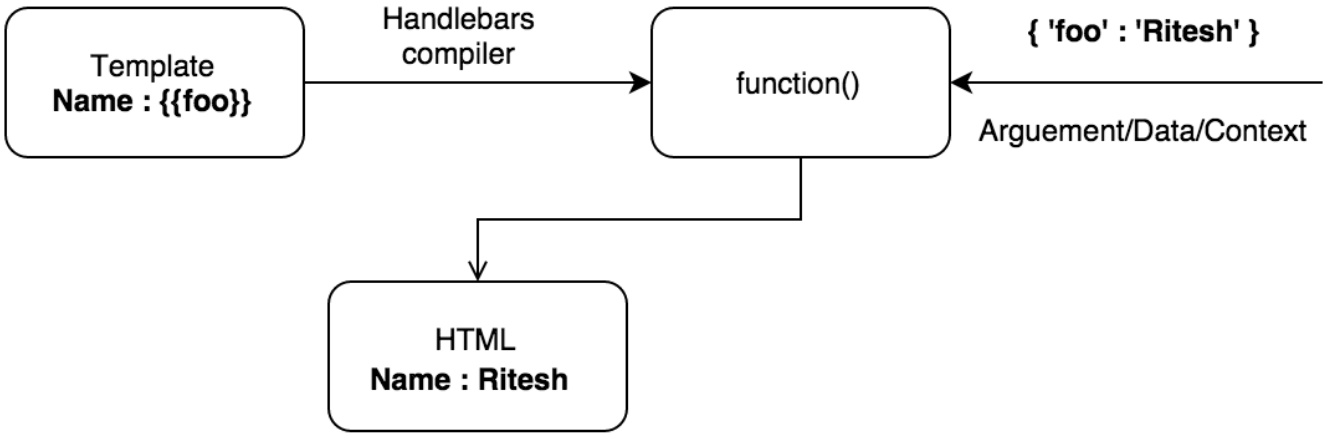
Porqué usar Handlebars

- Handlebars es uno de los más avanzados (precompilación y similares), rico en funciones y popular de todos los motores de plantillas JavaScript, y tiene la comunidad más activa.
- Handlebars es un motor de plantillas logic-less, lo que significa que hay poca o ninguna lógica en las plantillas que están en la página HTML.
- Los frameworks de vanguardia JavaScript, Meteor.js y Derby.js, utilizan Handlebars.js. Asimismo Ember.js también.
- Por su parte, aunque Backbone.js viene empaquetado con el motor de plantillas Underscore.js, es muy fácil cambiar a Handlebars.js.

Usando Handlebars

Cómo funciona

- Handlebars toma una plantilla con las variables y la compila en una función
- Esta función se ejecuta pasando un objeto JSON como argumento. Este objeto JSON se conoce como **contexto** y contiene los valores de las variables utilizadas en la plantilla
- En su ejecución, la función devuelve el HTML requerido después de sustituir las variables de la plantilla por sus valores correspondientes



Instala Handlebars

- Descarga Handlebars del sitio http://handlebarsjs.com/installation.html

- Añade el script en tu html

```
<script src="/path/to/handlebars.min.js"></script>
```

Usa Handlebars

- Primero compilamos la plantilla con la orden Handlebars.compile

```
<script id="entry-template" type="text/x-handlebars-template">
  <div class="entry">
    <h1>{{title}}</h1>
    <div class="body">
      {{body}}
    </div>
  </div>
</script>
```

```
var source   = $("#entry-template").html();
var template = Handlebars.compile(source);
```

- Ejecutamos la plantilla con el contexto

```
var context = {title: "My New Post", body: "This is my first post!"};
var html    = template(context);
```

```
<div class="entry">
  <h1>My New Post</h1>
  <div class="body">
    This is my first post!
  </div>
</div>
```

Plantillas

- Del mismo modo que en Mustache, las plantillas pueden ser inline o externas.

```
<script id="handlebars-demo" type="text/x-handlebars-template">
  <div>
    My name is {{name}}. I am a {{occupation}}.
  </div>
</script>
```

```
// Retrieve the template data from the HTML (jQuery is used here).
var template = $('#handlebars-demo').html();

// Compile the template data into a function
var templateScript = Handlebars.compile(template);

var context = { "name" : "Ritesh Kumar", "occupation" : "developer" };

// html = 'My name is Ritesh Kumar. I am a developer.'
var html = templateScript(context);

// Insert the HTML code into the page
$(document.body).append(html);
```

Sintaxis

Expresiones

- Las variables utilizadas dentro de las plantillas están rodeadas por dobles llaves {{}} y son conocidas como expresiones.

```
Mi nombre es {{name}}
```

Escape HTML

- Los guiadores pueden escapar el valor devuelto por la expresión. Por ejemplo, el carácter < se convierte en &lt. Si no se desea escaper un valor, se tiene que rodear la variable usando las llaves tridimensionales {{{variableName}}}. Por ejemplo, cuando la siguiente plantilla:

```
var context = {
  "language" : "<h3>Handlebars</h3>",
  "adjective": "<h3>awesome</h3>"
}
```

```
I am learning {{language}}. It is {{{adjective}}}.
```

```
I am learning &lt;h3&gt;Handlebars&lt;/h3&gt;. It is <h3>awesome</h3>
```

Sintaxis II

Comentarios

- Para escribir comentarios dentro de plantillas, la sintaxis es

```
{{! TypeYourCommentHere}}
```

- Los comentarios que tengan }} dentro o cualquier otro símbolo que tenga un significado especial en Handlebars, deben escribirse

```
{{! - TypeYourCommentHere--}}
```

- Por ejemplo

```
I am learning {{language}}. It is {{!--adjective--}}
```

Sintaxis III

- **Bloques**
  - Son expresiones que tienen una apertura de bloque ({{#}}) y un cierre ({{/}}).
  - Por ejemplo el bloque **if**

```
{{#if boolean}}
  Some Content here
{{/if}}
```

Paths

- Handlebars soporta rutas normales y anidadas, lo que permite buscar propiedades anidadas debajo del contexto actual.
- También se soporta el segmento de ruta **../**. Este segmento hace referencia al ámbito de la plantilla principal y no a un nivel superior en el contexto.

```
This article is available on
{{website.name}}.<br/>

{{#each names}}
  I am a {{../occupation}}. My name is
{{firstName}} {{lastName}}.<br/>
{{/each}}
```

```
var context = {
  "occupation" : "developer",
  "website" : {
    "name" : "sitepoint"
  }
  "names" : [
    {"firstName" : "Ritesh", "lastName" : "Kumar"},
    {"firstName" : "John" , "lastName" : "Doe"}
  ]
}
```

```
This article is available on sitepoint.
I am a developer. My name is Ritesh Kumar.
I am a developer. My name is John Doe.
```

Helpers

- Aunque Handlebars es un motor de plantillas sin lógica, puede ejecutar lógicas sencillas con helpers.
- Un helper es un identificador simple que puede ser seguido por parámetros (separados por un espacio), como se muestra a continuación:

```
{{#helperName parameter1 parameter2 ...}}
  Content here
{{/helperName}}
```

- Cada parámetro es una expresión Handlebars.
- Los helpers se puede acceder desde cualquier contexto de una plantilla.
- Algunos ayudantes que vienen en la distribución son **if**, **each**, **unless**, y **with**.

Helper Each

- Se usa para iterar sobre un array.
- La sintaxis es `{{#each ArrayName}} YourContent {{each}}.`
- Podemos referirnos a los elementos individuales del array usando la palabra clave **this** dentro del bloque.
- El índice del elemento del arrat se puede mostrar usando **{{@index}}**.

Ejemplo

```
{{#each countries}}
  {{@index}} : {{this}}<br>
{{/each}}

{{#each names}}
  Name : {{firstName}} {{lastName}}<br>
{{/each}}
```

```
var context = {
  "countries":["Russia","India","USA"],
  "names" : [
    {"firstName":"Ritesh","lastName":"Kumar"},
    {"firstName":"John","lastName":"Doe"}
  ]
}
```

0 : Russia
1 : India
2 : USA
Name : Ritesh Kumar
Name : John Doe

Helper If

- Es similar a una sentencia **if**.
- Si la condición evaluada se cumple, se renderizarán el bloque corresponsiente.
- También podemos especificar una sección de plantilla conocida como "else section", usando **{{else}}**.
- El helper **unless** es el inverso del **if**. Muestra el bloque cuando la condición evaluada es falsa.

```
{{#if countries}}
 The countries are present.
{{else}}
 The countries are not present.
{{/if}}
```

```
var context = {
  "countries": []
}
```

The countries are not present

Helpers personalizados

- Se pueden crear sus helpers propios para realizar lógicas complejas utilizando el sistema de expresión que proporciona Handlebars.
- Hay dos tipos de helpers:
  - **function helpers:** destinada a una única expresión
  - **block helpers:** se utiliza para expresiones de bloque.
- Los argumentos proporcionados a la función son parámetros escritos después del nombre del helper, separados por un espacio.
- Los helpers se crean utilizando el método **Handlebars.registerHelper ()**

```
Handlebars.registerHelper("HelperName", function(arguments){
  // This function is executed whenever this helper is used
})
```

Function helper

- La sintaxis para un functionn elper es **{{helperName parameter1 parameter2 ...}}.**

```
Handlebars.registerHelper("studyStatus",
function(passingYear) {
   if(passingYear < 2015) {
      return "passed";
   } else {
      return "not passed";
   }
})
```

```
{{#each students}}
  {{name}} has {{studyStatus passingYear}}.<br>
{{/each}}
```

```
var context = {
  "students":[
    {"name" : "John", "passingYear" : 2013},
    {"name" : "Doe" , "passingYear" : 2016}
  ]
}
```

John has passed.
Doe has not passed.

Block helper

- Su sintáxis es

```
{{#helperName parameter1 parameter2 ...}}
  Your content here
{{/helperName}}
```

- Cuando registramos un helper de cloque, Handlebars añade automáticamente un objeto de opciones como último parámetro a la función.
- Este objeto de opciones tiene un método **fn()** que permite cambiar el contexto del objeto temporalmente para acceder a una determinada propiedad.

Ejemplo

```
Handlebars.registerHelper("studyStatus", function(data,
options){
  var len = data.length;
  var returnData="";
  for(var i=0;i<len;i++){
    // change the value of the passingYear to
    // passed/not passed based on the conditions.
    data[i].passingYear=(data[i].passingYear < 2015) ? "passed"
: "not passed";

    // here options.fn(data[i]) temporarily changes the
    // scope of the whole studyStatus helper
    // block to data[i]. So {{name}}=data[i].name
    // in the template.
    returnData = returnData + options.fn(data[i]);

  }

  return returnData;
});
```

```
var context = {
  "students":[
    {"name" : "John", "passingYear" : 2013},
    {"name" : "Doe" , "passingYear" : 2016}
  ]
}
```

```
{{#studyStatus students}}
  {{name}} has {{passingYear}}
{{/studyStatus}}
```

John has passed.
Doe has not passed.

Plantillas parciales

- Son subplantillas que se pueden compartir entre diferentes plantillas.
- Se escriben usando {{> partialName}}.
- Antes de usarlos en HTML, necesitamos registrar el método parcial usando **Handlebars.registerPartial()**

```
Handlebars.registerPartial(
  'partialTemplate',
  '{{language}} is {{adjective}}. You are
reading this article on {{website}}.'
);

var context={
  "language"  : "Handlebars",
  "adjective": "awesome"
}
```

```
{{> partialTemplate website="sitepoint"}} <br>
{{> partialTemplate website="www.sitepoint.com"}}
```

Handlebars is awesome. You are reading this article on sitepoint
Handlebars is awesome. You are reading this article on www.sitepoint.com

Precompilación

- Como hemos visto, lo primero que hace Handlebars es compilar la plantilla en una función. Esta es una de las operaciones más costosas a realizar en el cliente.
- Podemos mejorar el rendimiento de la aplicación si precompilamos usando templateScript y luego enviamos la versión compilada al cliente.
- En este caso, la única tarea que se debe realizar en el cliente será la ejecución de la función compilada.
- Dado que el archivo precompilado es un script, podemos cargar el script en HTML como un archivo normal.

Precompilando una plantilla

Instala Handlebars

- Para instalar Handlebars es necesario tener instaldo Node
- Abre una ventana de consola y escribe

```
npm install handlebars -g
```

Crea una plantilla

- Crea una plantilla con la extensión **.handlebars** (por ejemplo demo.handlebars)

```
<div>
  My name is {{name}}. I am a {{occupation}}.
</div>
```

- Guarda **todas** las plantillas en una carpeta **templates**

Compila las plantillas

- Escribe la orden

```
handlebars path/to/templates -f templatesCompiled.js
```

  - Este comando generará un archivo denominado **templatesCompiled.js** que contiene todas las plantillas compiladas.
  - El compilador insertará las plantillas en **Handlebars.templates**.
  - Si el archivo de entrada es demo.handlebars, se insertará en **Handlebars.templates.demo**

Incluye las plantillas en el HTML

```
<script src="handlebars.runtime.js"></script>
<script src="path/to/templatesCompiled.js"></script>
```

Usa las plantillas

- Usa la plantilla usando el nombre derivado del archivo

```
var context = {
  "name" : "Ritesh Kumar",
  "occupation" : "developer"
}

var templateScript =
Handlebars.templates.demo(context);

$(document.body).append(templateScript);
```

Haciendo BananaTube SOLID

- Revisa la arquitectura del sistema implementado y realiza las modificaciones necesarias para garantizar que sigue los principios de buen diseño orgánico.

netmind

WeKnowIT

Barcelona

C. Almogàvers, 123
08018 Barcelona
Tel. 93 304.17.20
Fax. 93 304.17.22

Madrid

Plaza Carlos Trías Bertrán, 7
28020 Madrid
Tel. 91 442.77.03
Fax. 91 442.77.07

www.netmind.es

GOBIERNO DE ESPAÑA
MINISTERIO DE ENERGÍA, TURISMO Y AGENDA DIGITAL
red.es
ESTRATEGIA DE EMPRENDIMIENTO Y EMPLEO JOVEN garantía juvenil
Agenda Digital para España
UNIÓN EUROPEA
Fondo Social Europeo
"El FSE invierte en tu futuro"