Handlebar js

Handlebar js is an extention to mustache, a js template library for injecting expressions in the html.

1. Install

Npm install handlebars -g

Found all the libs in the following directory,

/c/Users/xli/AppData/Roaming/npm/node\_modules/handlebars/lib

1.Expression. {{}}

1. <div class="entry">
2. <h1>{{title}}</h1>
3. <div class="body">
4. {{body}}
5. </div>
6. </div>

use a <script> tag

<script id="entry-template" type="text/x-handlebars-template">

<div class="entry">

<h1>{{title}}</h1>

<div class="body">

{{body}}

</div>

</div>

</script>

It is important that you put the template inside a <script>-tag. Do not put it into the HTML directly or the HTML-parser might modify it (for example, if it [contains a table](https://html.spec.whatwg.org/multipage/parsing.html#unexpected-markup-in-tables)).

Starting project to test it,

Include the library in the <head>

// From File

<script src="handlebars-v2.0.0.js"></script>

// From CDN

<script src="https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/2.0.0/handlebars.js"></script>

Once you have the library in place, you can start writing templates. The recommended way of adding templates to your page is by including them in <script> tags with a special type. The type attribute is important, otherwise the browser will attempt to parse them as JavaScript (which they are not).

Templates need to be compiled to a JavaScript function before use. You can see an example below. Note that we are using jQuery to make working with the DOM easier, but Handlebars can work perfectly without it.

**Using nodejs**

To test this out using nodejs, here are the examples,

Test1.js

var Handlebars = require('handlebars');

var source = "<ul>{{#people}}<li>{{#link}}{{name}}{{/link}}</li>{{/people}}</ul>";

Handlebars.registerHelper('link', function(context, block) {

return '<a href="/people/' + this.id + '">' + this.name + '</a>';

});

var template = Handlebars.compile(source);

var data = { "people": [

{ "name": "Alan", "id": 1 },

{ "name": "Yehuda", "id": 2 }

]};

var result = template(data);

console.log(result);

run it,

C:\Users\xli\Desktop>node test1.js

<ul><li><a href="/people/1">Alan</a></li><li><a href="/people/2">Yehuda</a></li></ul>

So ‘link’ is the generated tag/place holder, to use it you have to do #link, and /link as beginning and closing bracket.

Test2.js

var Handlebars = require('handlebars');

var myInfo = "My name is {{name}} and I live at {{street}} in {{city}}, {{state}}";

var template = Handlebars.compile(myInfo );

var json = {name: "derek", street: "123 main st", city: "Pittsburgh", state:"PA"};

var result = template(json);

console.log(result);

output is,

C:\Users\xli\Desktop>node test2.js

My name is derek and I live at 123 main st in Pittsburgh, PA

The above is relatively simple.

**2. Executing using html**

Js.html, basic five lines to show the handlebar runs out of the box, and render the result with pure js.

<!DOCTYPE html>

<html>

<head>

<title>Title of the document</title>

<script src=*"https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/4.0.10/handlebars.min.js"*></script>

</head>

<body>

<h1>Test of handlebar </h1>

<!--

This example uses simple js dom traversal.

-->

<div id=*"data"*></div>

<script>

**var** myInfo="My name is {{name}} and I live at {{street}} in {{city}}, {{state}}";

**var** template=Handlebars.compile(myInfo);

**var** json={name: "derek", street: "123 main st", city: "Pittsburgh", state:"PA"};

**var** data=template(json);

document.getElementById("data").innerHTML+=data;

</script>

</body>

</html>

The five basic steps are,

Define the template, compile the template, feed data into the compiled template, and render the final result.

Jquery.html, simple dom traversal to show the final result in the render div.

<!DOCTYPE html>

<html>

<head>

<title>Title of the document</title>

<script src=*"https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/4.0.10/handlebars.min.js"*></script>

<script src=*"https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"*></script>

</head>

<body>

<h1>Test of handlebar </h1>

<!--This example uses jquery dom traversal.

Everything is in the pure javascript script block.

-->

<div id=*"data"*></div>

<script>

**var** myInfo="My name is {{name}} and I live at {{street}} in {{city}}, {{state}}";

**var** template=Handlebars.compile(myInfo);

**var** json={name: "derek", street: "123 main st", city: "Pittsburgh", state:"PA"};

**var** data=template(json);

$("#data").html(data);

</script>

</body>

</html>

Please check the code comment in the following file.

Jquery\_escape.html

<!DOCTYPE html>

<html>

<head>

<title>Title of the document</title>

<script src=*"https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/4.0.10/handlebars.js"*></script>

<script src=*"https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"*></script>

</head>

<body>

<!--

This example shows about the escape example of handlebar.

The following things should be noted:

1/ the reference of context in a json format using dot.

2/ since the template is recommended to be put inside <script> tag with special

type = text/x-handlebars-template. This way it keeps it clean.

3/ Pure js in <script> tag, where as you can get the template using dom traversal of pure js

or jquery. Then pass the template to the compile function to get a compiled template.

Then pass the data/json to the compile template object and generate the output result.

4/ render the output using jquery or js innerHTML

----->

<script id=*"expressions-template"* type=*"text/x-handlebars-template"*>

{{description.escaped}}

{{example}}

<br><br>

{{description.unescaped}}

{{{example}}}

</script>

<div id=*"content-placeholder"*></div>

<script>

// Grab the template script, using jquery

**var** theTemplateScript = $("#expressions-template").html();

// Compile the template

**var** theTemplate = Handlebars.compile(theTemplateScript);

// Define our data object

**var** context={

"description": {

"escaped": "Using {{}} brackets will result in escaped HTML:",

"unescaped": "Using {{{}}} will leave the context as it is:"

},

"example": "<button> Hello </button>"

};

// Pass our data to the template

**var** theCompiledHtml = theTemplate(context);

// Add the compiled html to the page

$('#content-placeholder').html(theCompiledHtml);

</script>

</body>

</html>

One misunderstand is the following writing,

<!DOCTYPE html>

<html>

<head>

<title>Title of the document</title>

<script src=*"https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/4.0.10/handlebars.min.js"*></script>

</head>

<body>

<h1>wrong exmaple of handlebar </h1>

<!--

This file will not work as type="text/x-handlebars-template" does not work as

normal js definition thus nothing else that has to be js will run.

-->

<div id=*"data"*></div>

<script type=*"text/x-handlebars-template"*>

var myInfo="My name is {{name}} and I live at {{street}} in {{city}}, {{state}}";

var template=Handlebars.compile(myInfo);

var json={name: "derek", street: "123 main st", city: "Pittsburgh", state:"PA"};

var data=template(json);

document.getElementById("data").innerHTML+=data;

</script>

</body>

</html>

Another jquery example, that is using document.ready().

<!DOCTYPE html>

<html>

<head>

<title>Title of the document</title>

<script src=*"https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/4.0.10/handlebars.min.js"*></script>

<script src=*"https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"*></script>

</head>

<body>

<h1>Test of handlebar </h1>

<!--This example uses jquery dom traversal.

Everything is in the pure javascript script block.

Uses a document.ready function will load it upon page loading

-->

<div id=*"data"*></div>

<!-- The #each helper iterates over an array of items. -->

<script id=*"example-template"* type=*"text/x-handlebars-template"*>

<!-- people is looked up on the global context, the one we pass to the compiled template -->

{{#each people}}

<!-- Here the context is each individual person. So we can access its properties directly: -->

<p>{{firstName}} {{lastName}}</p>

{{/each}}

</script>

<script>

$(**function** () {

// Grab the template script

**var** theTemplateScript = $("#example-template").html();

// Compile the template

**var** theTemplate = Handlebars.compile(theTemplateScript);

// This is the default context, which is passed to the template

**var** context = {

people: [

{ firstName: 'Homer', lastName: 'Simpson' },

{ firstName: 'Peter', lastName: 'Griffin' },

{ firstName: 'Eric', lastName: 'Cartman' },

{ firstName: 'Kenny', lastName: 'McCormick' },

{ firstName: 'Bart', lastName: 'Simpson' }

]

};

// Pass our data to the template

**var** theCompiledHtml = theTemplate(context);

// Add the compiled html to the page

$(document.body).append(theCompiledHtml);

});

</script>

</body>

</html>

**3. Context**

Context is the template’s feed data, usually in json. To iterate through it.

In the <script> template part, we wrote this,

<script id=*"example-template"* type=*"text/x-handlebars-template"*>

{{#each people}}

<!-- Here the context is each individual person. So we can access its properties directly: -->

<p>{{firstName}} {{lastName}}</p>

{{/each}}

</script>

In the <script> js part, we wrote this,

**var** context = {

people: [

{ firstName: 'Homer', lastName: 'Simpson' },

{ firstName: 'Peter', lastName: 'Griffin' },

{ firstName: 'Eric', lastName: 'Cartman' },

{ firstName: 'Kenny', lastName: 'McCormick' },

{ firstName: 'Bart', lastName: 'Simpson' }

]

};

**4. Helpers**

Handlebars doesn't allow you to write JavaScript directly within templates. Instead, it gives you helpers. These are JavaScript functions that you can call from your templates, and help you reuse code and create complex templates. To call a helper, just use it as an expression - {{helpername}}. You can pass parameters as well - {{helpername 12345}}, which are passed as parameters to your helper function.

You can write your own registered helper function to use that with, and the following example shows you how, Handlebars.registerHelper(‘functionName’, function(str) {});

Since this is a js function, you can put it in the js <script> section.

<script id=*"built-in-helpers-template"* type=*"text/x-handlebars-template"*>.

Customed\_helper.html

<!DOCTYPE html>

<html>

<head>

<title>Title of the document</title>

<script src=*"https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/4.0.10/handlebars.js"*></script>

<script src=*"https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"*></script>

<script id=*"built-in-helpers-template"* type=*"text/x-handlebars-template"*>

{{#each animals}}

<p>

The {{capitalize this.name}} says

{{#if this.noise}}

"{{this.noise}}".

{{else}}

nothing since its a {{this.name}}.

{{/if}}

</p>

{{/each}}

</script>

</head>

<body>

<div class=*"content-placeholder"*></div>

<script>

$(**function** () {

// Register a helper

Handlebars.registerHelper('capitalize', **function**(str){

// str is the argument passed to the helper when called

str = str || '';

**return** str.slice(0,1).toUpperCase() + str.slice(1);

});

// Grab the template script

**var** theTemplateScript = $("#built-in-helpers-template").html();

// Compile the template

**var** theTemplate = Handlebars.compile(theTemplateScript);

// We will call this template on an array of objects

**var** context = {

animals:[

{

name: "cow",

noise: "moooo"

},

{

name: "cat",

noise: "meow"

},

{

name: "fish",

noise: ""

},

{

name: "farmer",

noise: "Get off my property!"

}

]

};

// Pass our data to the template

**var** theCompiledHtml = theTemplate(context);

// Add the compiled html to the page

$('.content-placeholder').html(theCompiledHtml);

});

</script>

</body>

</html>

**Create block helpers**

Block helpers is some sort of helpers for the blocks, similar to #each and #if.

For any of these helpers, you use {{#xxx}}, {{/xxx}} in the template for control.

In the following example, you are going to see

<!DOCTYPE html>

<!--This example you actually creates a block helper-->

<html>

<head>

<title>Title of the document</title>

<script src=*"https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/4.0.10/handlebars.js"*></script>

<script src=*"https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"*></script>

<script id=*"block-expressions-template"* type=*"text/x-handlebars-template"*>

<p> The <b> {{#uppercase}} konami {{/uppercase}} </b> Code is a cheat code that appears in many video games.</p>

<p>During the title screen before the game demo begins, the player could press the following sequence of buttons on the game controller to enable the cheat:</p>

<p>{{#uppercase}}{{code}}{{/uppercase}}</p>

<p>The code is also present as an Easter egg on a number of websites.</p>

</script>

</head>

<body>

<div class=*"content-placeholder"*></div>

<script>

$(**function** () {

// Grab the template script

**var** theTemplateScript = $("#block-expressions-template").html();

// This is our block helper

// The name of our helper is provided as the first parameter - in this case 'uppercase'

Handlebars.registerHelper('uppercase', **function**(options) {

// "this" is the context that existed when calling the helper.

// The options object has a special function - fn. This is a

// compiled version of the template that is contained between the opening and closing

// blocks of this helper. To get a string, call fn with the context:

**return** options.fn(**this**).toUpperCase();

});

// Compile the template

**var** theTemplate = Handlebars.compile(theTemplateScript);

// Define our data object

**var** context = {

"code": "up up down down left right left right b a select start"

};

// Pass our data to the template

**var** theCompiledHtml = theTemplate(context);

// Add the compiled html to the page

$('.content-placeholder').html(theCompiledHtml);

});

</script>

</body>

</html>

Other building block helpers,

{{#if}} htmlAndData {{else}} htmlAndData{{/if}}

{{#unless}} {{/unless}}

Eg:

<div class="entry">

{{#unless license}}

<h3 class="warning">WARNING: This entry does not have a license!</h3>

{{/unless}}

</div>

{{#each}} {{/each}}

<ul class="people\_list">

{{#each people}}

<li>{{this}}</li>

{{/each}}

</ul>

When used with this context,

{

people: [

"Yehuda Katz",

"Alan Johnson",

"Charles Jolley"

]

}

will generate

<ul class="people\_list">

<li>Yehuda Katz</li>

<li>Alan Johnson</li>

<li>Charles Jolley</li>

</ul>

You can use the this expression in any context to reference the current context.

**Use key, value and childKey and childValue**

{{#each array as |value key|}}

{{#each child as |childValue childKey|}}

{{key}} - {{childKey}}. {{childValue}}

{{/each}}

{{/each}}

**@key, @index**

**{{this}}, refers to the current iterating object.**

**@key, the current index of the object in the object iterations/list.**

**@index, same as @key.**

{{#each array as |value key|}}

{{#each child as |childValue childKey|}}

{{key}} - {{childKey}}. {{childValue}}

{{/each}}

{{/each}}

Will create a key and value variable that children may access without the need for depthed variable references. In the example above, {{key}} is identical to {{@../key}} but in many cases is more readable.

<!DOCTYPE html>

<!--This example you actually creates a block helper-->

<html>

<head>

<title>Title of the document</title>

<script src=*"https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/4.0.10/handlebars.js"*></script>

<script src=*"https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"*></script>

<script id=*"example-template"* type=*"text/x-handlebars-template"*>

<ul class="people\_list">

{{#each people as |value key|}}

<li>{{@index}}.{{@key}}.{{key}}-{{value}}</li>

{{/each}}

</ul>

</script>

</head>

<body>

<div id=*"content-placeholder"*></div>

<script>

$(**function** () {

// Grab the template script

**var** theTemplateScript = $("#example-template").html();

// Compile the template

**var** theTemplate = Handlebars.compile(theTemplateScript);

// This is the default context, which is passed to the template

**var** context = {

people: [

{ firstName: 'Homer', lastName: 'Simpson' },

{ firstName: 'Peter', lastName: 'Griffin' },

{ firstName: 'Eric', lastName: 'Cartman' },

{ firstName: 'Kenny', lastName: 'McCormick' },

{ firstName: 'Bart', lastName: 'Simpson' }

]

};

// Pass our data to the template

**var** theCompiledHtml = theTemplate(context);

// Add the compiled html to the page

//$(document.body).append(theCompiledHtml);

$("#content-placeholder").html(theCompiledHtml);

});

</script>

</body>

</html>

The following registered a function named “bold” and it is actually {{#bold}}{{/bold}}, and it is taking one argument. This argument is the text that it is used to quote. The options.fn means the object function itself. options.fn(**this**) means the function takes itself and it literally did not do anything.

Handlebars.registerHelper('bold', **function**(options) {

**return** **new** Handlebars.SafeString(

'<b>'

+ options.fn(**this**)

+ '</b>');

});

<div class="entry">

<h1>{{title}}</h1>

<div class="body">

{{#noop}}{{body}}{{/noop}}

</div>

</div>

The noop helper (short for "no operation") will receive an options hash. This options hash contains a function (options.fn) that behaves like a normal compiled Handlebars template. Specifically, the function will take a context and return a String.

**Ref:** <http://handlebarsjs.com/block_helpers.html> **(TODO)**

<http://handlebarsjs.com/>

**With block**

You can optionally provide an {{else}} section which will display only when the passed value is empty.

{{#with author}}

<p>{{name}}</p>

{{else}}

<p class="empty">No content</p>

{{/with}}

<!DOCTYPE html>

<!--This example you actually creates a block helper-->

<html>

<head>

<title>Title of the document</title>

<script src=*"https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/4.0.10/handlebars.js"*></script>

<script src=*"https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"*></script>

<script id=*"example-template"* type=*"text/x-handlebars-template"*>

<h1>{{title}}</h1>

<ul class="people\_list">

{{#with author}}

<li>{{firstName}} - {{lastName}}</li>

{{/with}}

</ul>

<br/> <br/>

<ul class="people\_list">

{{#with author as |me|}}

<li>{{me.firstName}} - {{me.lastName}}</li>

{{/with}}

</ul>

</script>

</head>

<body>

<div id=*"content-placeholder"*></div>

<script>

$(**function** () {

// Grab the template script

**var** theTemplateScript = $("#example-template").html();

// Compile the template

**var** theTemplate = Handlebars.compile(theTemplateScript);

// This is the default context, which is passed to the template

**var** context = {

title: 'with block testing',

author: { firstName: 'Homer', lastName: 'Simpson' }

};

// Pass our data to the template

**var** theCompiledHtml = theTemplate(context);

// Add the compiled html to the page

//$(document.body).append(theCompiledHtml);

$("#content-placeholder").html(theCompiledHtml);

});

</script>

</body>

</html>

5. escaping

Handlebars HTML-escapes values returned by a {{expression}}. If you don't want Handlebars to escape a value, use the "triple-stash", {{{}}}.