handlebars

Fork me on GitHub

Handlebars provides the power necessary to let you build **semantic templates** effectively with no frustration.

Handlebars is largely compatible with Mustache templates. In most cases it is possible to swap out Mustache with Handlebars and continue using your current templates. Complete details can be found here.

Installation

Getting Started

Handlebars templates look like regular HTML, with embedded handlebars expressions.

```
<div class="entry">
  <h1>{{title}}</h1>
  <div class="body">
    {{body}}
  </div>
</div>
```

A handlebars expression is a `{{`, some contents, followed by a `}}`

Learn More: Expressions

You can deliver a template to the browser by including it in 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>
```

Compile a template in JavaScript by using `Handlebars.compile`

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

It is also possible to precompile your templates. This will result in a smaller required runtime library and significant savings from not having to compile the template in the browser. This can be especially important when working with mobile devices.

Learn More: Precompilation

Get the HTML result of evaluating a Handlebars template by executing the template with a context.

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

var html    = template(context);
```

results in

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

Learn More: Execution

HTML Escaping

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

```
<div class="entry">
  <h1>{{title}}</h1>
  <div class="body">
    {{{body}}}
  </div>
</div>
```

with this context:

```
{
  title: "All about <p> Tags",
  body: "<p>This is a post about &lt;p&gt; tags</p>"
}
```

results in:

```
<div class="entry">
  <h1>All About &lt;p&gt; Tags</h1>
  <div class="body">
    <p>This is a post about &lt;p&gt; tags</p>
  </div>
</div>
```

Handlebars will not escape a `Handlebars.SafeString`. If you write a helper that generates its own HTML, you will usually want to return a `new Handlebars.SafeString(result)`. In such a circumstance, you will want to manually escape parameters.

```
Handlebars.registerHelper('link', function(text, url) {
  text = Handlebars.Utils.escapeExpression(text);
  url  = Handlebars.Utils.escapeExpression(url);

  var result = '<a href="' + url + '">' + text + '</a>';

  return new Handlebars.SafeString(result);
});
```

This will escape the passed in parameters, but mark the response as safe, so Handlebars will not try to escape it even if the "triple-stash" is not used.

Block Expressions

Block expressions allow you to define helpers that will invoke a section of your template with a different context than the current. These block helpers are identified by a `#` preceeding the helper name and require a matching closing mustache, `/`, of the same name.

Let's consider a helper that will generate an HTML list:

```
{{#list people}}{{firstName}} {{lastName}}{{/list}}
```

If we have the following context:

```
{
  people: [
    {firstName: "Yehuda", lastName: "Katz"},
    {firstName: "Carl", lastName: "Lerche"},
    {firstName: "Alan", lastName: "Johnson"}
  ]
}
```

we would create a helper named `list` to generate our HTML list. The helper receives the `people` as its first parameter, and an options hash as its second parameter. The options hash contains a property named `fn`, which you can invoke with a context just as you would invoke a normal Handlebars template.

```
Handlebars.registerHelper('list', function(items, options) {
  var out = "<ul>";

  for(var i=0, l=items.length; i<l; i++) {
    out = out + "<li>" + options.fn(items[i]) + "</li>";
  }

  return out + "</ul>";
});
```

When executed, the template will render:

```
<ul>
  <li>Yehuda Katz</li>
  <li>Carl Lerche</li>
  <li>Alan Johnson</li>
</ul>
```

Block helpers have more features, such as the ability to create an `else` section (used, for instance, by the built-in `if` helper).

Since the contents of a block helper are escaped when you call `options.fn(context)`, Handlebars does not escape the results of a block helper. If it did, inner content would be double-escaped!

Learn More: Block Helpers

Handlebars Paths

Handlebars supports simple paths, just like Mustache.

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

Handlebars also supports nested paths, making it possible to look up properties nested below the current context.

```
<div class="entry">
  <h1>{{title}}</h1>
  <h2>By {{author.name}}</h2>

  <div class="body">
    {{body}}
  </div>
</div>
```

That template works with this context

```
var context = {
  title: "My First Blog Post!",
  author: {
    id: 47,
    name: "Yehuda Katz"
  },
  body: "My first post. Wheeeee!"
};
```

This makes it possible to use Handlebars templates with more raw JSON objects

This makes it possible to use Handlebars templates with more raw JSON objects.

Nested handlebars paths can also include `../` segments, which evaluate their paths against a parent context.

```
<h1>Comments</h1>

<div id="comments">
  {{#each comments}}
  <h2><a href="/posts/{{../permalink}}#{{id}}">{{title}}</a></h2>
  <div>{{body}}</div>
  {{/each}}
</div>
```

Even though the link is printed while in the context of a comment, it can still go back to the main context (the post) to retrieve its permalink.

The exact value that `../` will resolve to varies based on the helper that is calling the block. Using `../` is only necessary when context changes, so children of helpers such as `each` would require the use of `../` while children of helpers such as `if` do not.

```
{{permalink}}
{{#each comments}}
  {{../permalink}}

  {{#if title}}
    {{../permalink}}
  {{/if}}
{{/each}}
```

In this example all of the above reference the same `permalink` value even though they are located within different blocks. This behavior is new as of Handlebars 4, the release notes discuss the prior behavior as well as the migration plan.

Handlebars also allows for name conflict resolution between helpers and data fields via a `this` reference:

```
<p>{{./name}} or {{this/name}} or {{this.name}}</p>
```

Any of the above would cause the `name` field on the current context to be used rather than a helper of the same name.

Template comments with `{{!-- --}}` or `{{! }}`.

You can use comments in your handlebars code just as you would in your code. Since there is generally some level of logic, this is a good practice.

```
<div class="entry">
  {{!-- only output author name if an author exists --}}
  {{#if author}}
    <h1>{{firstName}} {{lastName}}</h1>
  {{/if}}
</div>
```

The comments will not be in the resulting output. If you'd like the comments to show up. Just use html comments, and they will be output.

```
<div class="entry">
  {{! This comment will not be in the output }}
  <!-- This comment will be in the output -->
</div>
```

Any comments that must contain `}}` or other handlebars tokens should use the `{{!-- --}}` syntax.

Helpers

Handlebars helpers can be accessed from any context in a template. You can register a helper with the `Handlebars.registerHelper` method.

```
<div class="post">
  <h1>By {{fullName author}}</h1>
  <div class="body">{{body}}</div>

  <h1>Comments</h1>

  {{#each comments}}
  <h2>By {{fullName author}}</h2>
  <div class="body">{{body}}</div>
  {{/each}}
</div>
```

when using this context and helpers:

```
var context = {
  author: {firstName: "Alan", lastName: "Johnson"},
  body: "I Love Handlebars",
  comments: [{
    author: {firstName: "Yehuda", lastName: "Katz"},
    body: "Me too!"
  }]
};

Handlebars.registerHelper('fullName', function(person) {
  return person.firstName + " " + person.lastName;
});
```

results in:

```
<div class="post">
  <h1>By Alan Johnson</h1>
  <div class="body">I Love Handlebars</div>

  <h1>Comments</h1>
```

```
  <h2>By Yehuda Katz</h2>
  <div class="body">Me Too!</div>
</div>
```

Helpers receive the current context as the `this` context of the function.

```
<ul>
  {{#each items}}
  <li>{{agree_button}}</li>
  {{/each}}
</ul>
```

when using this context and helpers:

```
var context = {
  items: [
    {name: "Handlebars", emotion: "love"},
    {name: "Mustache", emotion: "enjoy"},
    {name: "Ember", emotion: "want to learn"}
  ]
};

Handlebars.registerHelper('agree_button', function() {
  var emotion = Handlebars.escapeExpression(this.emotion),
      name = Handlebars.escapeExpression(this.name);

  return new Handlebars.SafeString(
    "<button>I agree. I " + emotion + " " + name + "</button>"
  );
});
```

results in:

```
<ul>
  <li><button>I agree. I love Handlebars</button></li>
  <li><button>I agree. I enjoy Mustache</button></li>
  <li><button>I agree. I want to learn Ember</button></li>
```

```
</ul>
```

If your helper returns HTML that you do not want escaped, make sure to return a new `Handlebars.SafeString`.

Literals

Helper calls may also have literal values passed to them either as parameter arguments or hash arguments. Supported literals include numbers, strings, `true`, `false`, `null` and `undefined`.

```
{{agree_button "My Text" class="my-class" visible=true counter=4}}
```

Partials

Handlebars partials allow for code reuse by creating shared templates. Rendering this template

```
<div class="post">
  {{> userMessage tagName="h1" }}

  <h1>Comments</h1>

  {{#each comments}}
    {{> userMessage tagName="h2" }}
  {{/each}}
</div>
```

when using this partial and context:

```
Handlebars.registerPartial('userMessage',
    '<{{tagName}}>By {{author.firstName}} {{author.lastName}}</{{tagName}}>'
    + '<div class="body">{{body}}</div>');
```

```
var context = {
```

```
var context = {
  author: {firstName: "Alan", lastName: "Johnson"},
  body: "I Love Handlebars",
  comments: [{
    author: {firstName: "Yehuda", lastName: "Katz"},
    body: "Me too!"
  }]
};
```

results in:

```
<div class="post">
  <h1>By Alan Johnson</h1>
  <div class="body">I Love Handlebars</div>

  <h1>Comments</h1>

  <h2>By Yehuda Katz</h2>
  <div class="body">Me Too!</div>
</div>
```

Learn More: Partials

Built-In Helpers

Handlebars offers a variety of built-in helpers such as the `if` conditional and `each` iterator.

Learn More: Built-In Helpers

API Reference

Handlebars offers a variety of APIs and utility methods for applications and helpers

Handlebars offers a variety of API's and utility methods for applications and helpers.

Learn More: API Reference

Found a documentation issue? Tell us!