Fork me on GitHub

handlebars

Handlebars expressions are the basic unit of a Handlebars template. You can use them alone in a `{{mustache}}`, pass them to a Handlebars helper, or use them as values in hash arguments.

Basic Usage

The simplest Handlebars expression is a simple identifier:

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

This expression means "look up the `title` property in the current context". Block helpers may manipulate the current context, but they do not change the basic meaning of an expression.

Actually, it means "look for a helper named `title`, then do the above", but we'll get to that soon enough.

Handlebars expressions can also be dot-separated paths.

```
<h1>{{article.title}}</h1>
```

This expression means "look up the `article` property in the current context. Then look up the `title` property in the result".

Handlebars also supports a deprecated `/` syntax, so you could write the above template as:

```
<h1>{{article/title}}</h1>
```

Identifiers may be any unicode character except for the following:

Whitespace `!` `"` `#` `%` `&` `'` `(` `)` `*` `+` `,` `.` `/` `;` `<` `=` `>` `@` `[` `\` `]` `^` `` ` `` `{` `|` `}` `~`

To reference a property that is not a valid identifier, you can use segment-literal notation, `[`:

```
{{#each articles.[10].[#comments]}}
  <h1>{{subject}}</h1>
  <div>
    {{body}}
  </div>
{{/each}}
```

In the example above, the template will treat the `each` parameter roughly equivalent to this javascript: `articles[10]['#comments']`

You may not include a closing `]` in a path-literal, but all other characters are fair game.

JavaScript-style strings, `"` and `'`, may also be used vs. `[` pairs.

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

```
{{{foo}}}
```

Helpers

A Handlebars helper call is a simple identifier, followed by zero or more parameters (separated by space). Each parameter is a Handlebars expression.

```
{{{link story}}}
```

In this case, `link` is the name of a Handlebars helper, and story is a parameter to the helper. Handlebars evaluates parameters in exactly the same way described above in "Basic Usage".

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

  return new Handlebars.SafeString(
    "<a href='" + url + "'>" + text + "</a>"
  );
});
```

When returning HTML from a helper, you should return a Handlebars SafeString if you don't want it to be escaped by default. When using SafeString all unknown or unsafe data should be manually escaped with the `escapeExpression` method.

You can also pass a simple String, number, or boolean as a parameter to Handlebars helpers.

```
{{{link "See more..." story.url}}}
```

In this case, Handlebars will pass the link helper two parameters: the String `"See more..."` and the result of evaluating `story.url` in the current context.

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

  return new Handlebars.SafeString(
    "<a href='" + url + "'>" + text + "</a>"
  );
});
```

You could use the exact same helper with dynamic text based on the value of `story.text`:

```
{{{link story.text story.url}}}
```

Handlebars helpers can also receive an optional sequence of key-value pairs as their final parameter (referred to as hash arguments in the documentation):

```
{{{link "See more..." href=story.url class="story"}}}
```

The keys in hash arguments must each be simple identifiers, and the values are Handlebars expressions. This means that values can be simple identifiers, paths, or Strings.

```
Handlebars.registerHelper('link', function(text, options) {
  var attrs = [];

  for (var prop in options.hash) {
    attrs.push(
        Handlebars.escapeExpression(prop) + '="'
        + Handlebars.escapeExpression(options.hash[prop]) + '"');
  }

  return new Handlebars.SafeString(

    "<a " + attrs.join(" ") + ">" + Handlebars.escapeExpression(text) + "</a>"
  );
});
```

Handlebars provides additional metadata, such as Hash arguments, to helpers as a final parameter.

Handlebars also offers a mechanism for invoking a helper with a block of the template. Block helpers can then invoke that block zero or more times with any context it chooses.

Learn More: Block Helpers

Subexpressions

Handlebars offers support for subexpressions, which allows you to invoke multiple helpers within a single mustache, and pass in the results of inner helper invocations as arguments to outer helpers. Subexpressions are delimited by parentheses.

```
{{outer-helper (inner-helper 'abc') 'def'}}
```

In this case, `inner-helper` will get invoked with the string argument `'abc'`, and whatever the `inner-helper` function returns will get passed in as the first argument to `outer-helper` (and `'def'` will get passed in as the second argument to `outer-helper`.

Whitespace Control

Template whitespace may be omitted from either side of any mustache statement by adding a `~` character by the braces. When applied all whitespace on that side will be removed up to the first handlebars expression or non-whitespace character on that side.

```
{{#each nav ~}}
  <a href="{{url}}">
    {{~#if test}}
      {{~title}}
    {{~^~}}
      Empty
    {{~/if~}}
  </a>
{{~/each}}
```

with this context:

```
{
  nav: [
    {url: 'foo', test: true, title: 'bar'},
    {url: 'bar'}
  ]
}
```

results in output sans newlines and formatting whitespace:

```
<a href="foo">bar</a><a href="bar">Empty</a>
```

This expands the default behavior of stripping lines that are "standalone" helpers (only a block helper, comment, or partial and whitespace).

```
{{#each nav}}
  <a href="{{url}}">
    {{#if test}}
      {{title}}
    {{^}}
      Empty
    {{/if}}
  </a>
{{~/each}}
```

will render

```
<a href="foo">
    bar
</a>
<a href="bar">
    Empty
</a>
```

Escaping

Handlebars content may be escaped in one of two ways, inline escapes or raw block helpers. Inline escapes created by prefixing a mustache block with `\`. Raw blocks are created using `{{{{` mustache braces.

```
\{{escaped}}
{{{{raw}}}}
  {{escaped}}
{{{{/raw}}}}
```

Raw blocks operate in the same manner as other block helpers with the distinction of the child content is treated as a literal string.

Found a documentation issue? Tell us!