参考：<http://keenwon.com/992.html>

使用Handlebars，你可以轻松创建语义化模板，Mustache模板和Handlebars是兼容的，所以你可以将Mustache导入Handlebars以使用 Handlebars 强大的功能。

Complete details can be found [here](https://github.com/wycats/handlebars.js#differences-between-handlebarsjs-and-mustache).

**开始**

Handlebars模板看起来和HTML一样，只是嵌入了 handlebars 表达式

<div class="entry">

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

<div class="body">

{{body}}

</div>

</div>

handlebars表达式以{{开头，中间写一些内容，以}}结尾。

[**更多内容：表达式**](http://keenwon.com/1005.html)

# Handlebars.js 表达式

Posted in [前端](http://keenwon.com/category/frontend) By [KeenWon](http://keenwon.com/author/semanwmj) On 2014年4月3日 Views: 2,056

表达式是Handlebars模板的基本组成部分，你可以单独的以{{mustache}}的形式使用，传递给Handlebars，或者当做hash的参数。

### 基础

最简单的Handlebars表达式是简单的标识

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

这个表达式的意思是：查找当前数据上下文(context)的title属性。块helpers可以操作当前的数据上下文，但是不影响表达式的意思。

表达式也可以是用“点”分割的路径

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

这个表达式的意思是：查找当前数据上下文的article属性，然后在结果中查找title属性。handlebars也支持已经弃用/分隔符，所以上面的表达式也可以写成

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

表达式可以包含任意的字符，除了：空格! " # % & ' ( ) \* + , . / ; <= > @ [ \ ] ^ ` { | } ~

如果表达式不是有效的字符，可以使用[:

{{#each articles[10].[#comments]}}

<h1>{{subject}}</h1>

<div>

{{body}}

</div>

{{/each}}

上例中，模板会循环articles[10]['#comments']。

Handlebars会编译{{expression}}返回的值，如果你不想结果被编译，可以使用{{{

{{{foo}}}

### Helpers

helper调用是个简单的标识符，跟着零个或者多个参数（用空格分隔），每一个参数都是Handlebars表达式

{{{link story}}}

上例中，link是helper的名称，story是helper的参数。Handlebars就像上文所说的处理参数

Handlebars.registerHelper('link', function(object) {

return new Handlebars.SafeString(

"<a href='" + object.url + "'>" + object.text + "</a>"

);

});

当helper返回html时，如果你不想被编码的话，需要返回Handlebars安全字符串。

也可以传递一个字符串当做helper的参数：

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

上例中，Handlebars会传给helper两个参数：字符串See more...和当前上下文的story.url

Handlebars.registerHelper('link', function(text, url) {

return new Handlebars.SafeString(

"<a href='" + url + "'>" + text + "</a>"

);

});

你也可以使用同样的helper，通过动态字符串story.text

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

Handlebars helper 可以接受个任意顺序的“键值对”作为最后一个参数（文档中被称为hash）

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

hash的key必须是简单的标识符，value必须是表达式（就是字符串，路径，标识符）

Handlebars.registerHelper('link', function(text, options) {

var attrs = [];

for(var prop in options.hash) {

attrs.push(prop + '="' + options.hash[prop] + '"');

}

return new Handlebars.SafeString(

"<a " + attrs.join(" ") + ">" + text + "</a>"

);

});

Handlebars提供额外的元数据，例如hash，作为helper的最后一个参数。

Handlebars也提供了一种机制，调用helper with部分模板。块级helper以后可以调用零次或者多次，with任意的数据上下文。

[更多资料：块级Helpers](http://keenwon.com/1033.html)

# Handlebars.js 块级Helpers

Posted in [前端](http://keenwon.com/category/frontend) By [KeenWon](http://keenwon.com/author/semanwmj) On 2014年4月3日 Views: 2,024

块级的Helpers可以在新的数据上下文下自定义迭代器和其他helpers。

### 基础

下面定义一个简单的块级Helpers

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

noop helper接受一个option hash，这个option包含一个函数（option.fn），它就和普通编译出的模板一样，接受一个数据上下文，返回一个字符串。

1. Handlebars.registerHelper('noop', **function**(options) {
2. **return** options.fn(**this**);
3. });

Handlebars可以通过this调用当前的数据上下文。

### The with helper

基于刚才的noop helper，with helper的使用就很明显了。使用with helper，我们可以限定任意的数据上下文。

1. **<div** **class**="entry"**>**
2. **<h1>**{{title}}**</h1>**
3. {{#with story}}
4. **<div** **class**="intro"**>**{{{intro}}}**</div>**
5. **<div** **class**="body"**>**{{{body}}}**</div>**
6. {{/with}}
7. **</div>**

当你的JSON包含很多重要的属性时，这个helper会非常有用。你需要一直重复父级的名称，例如使用下面的上下文

1. {
2. title: "First Post",
3. story: {
4. intro: "Before the jump",
5. body: "After the jump"
6. }
7. }

with helper和上面的noop很像。接受一个参数，渲染到{{mustache}}块中。

1. Handlebars.registerHelper('with', **function**(context, options) {
2. **return** options.fn(context);
3. });

### 简单的迭代器

块级helper的一个最常见的用法是自定义迭代器。事实上，所有的内置helper都块级helper一样。下面我们看一下内置的each helper如何工作。

1. **<div** **class**="entry"**>**
2. **<h1>**{{title}}**</h1>**
3. {{#with story}}
4. **<div** **class**="intro"**>**{{{intro}}}**</div>**
5. **<div** **class**="body"**>**{{{body}}}**</div>**
6. {{/with}}
7. **</div>**
8. **<div** **class**="comments"**>**
9. {{#each comments}}
10. **<div** **class**="comment"**>**
11. **<h2>**{{subject}}**</h2>**
12. {{{body}}}
13. **</div>**
14. {{/each}}
15. **</div>**

上例中，我们循环comments数组

1. Handlebars.registerHelper('each', **function**(context, options) {
2. **var** ret = "";
4. **for**(**var** i=0, j=context.length; i<j; i++) {
5. ret = ret + options.fn(context[i]);
6. }
8. **return** ret;
9. });

传入数据上下文（context），循环它们，生成一个字符串，返回。

现在，我们很容易看出如何实现更加复杂的迭代器。例如生成一个<ul>列表

1. {{#list nav}}
2. **<a** **href**="{{url}}"**>**{{title}}**</a>**
3. {{/list}}

使用下面的数据

1. {
2. nav: [
3. { url: "http://www.yehudakatz.com", title: "Katz Got Your Tongue" },
4. { url: "http://www.sproutcore.com/block", title: "SproutCore Blog" },
5. ]
6. }

list helper和原生的each helper不一样

1. Handlebars.registerHelper('list', **function**(context, options) {
2. **var** ret = "<ul>";
4. **for**(**var** i=0, j=context.length; i<j; i++) {
5. ret = ret + "<li>" + options.fn(context[i]) + "</li>";
6. }
8. **return** ret + "</ul>";
9. });

### 条件判断

另一个常见的应用就是条件判断。同样的，内置的if和unless像普通的块级helper一样实现。

1. {{#if isActive}}
2. **<img** **src**="star.gif" **alt**="Active"**>**
3. {{/if}}

控制结构通常不改变当前的数据上下文，但是会根据一些变量判断是否渲染相应的块。

1. Handlebars.registerHelper('if', **function**(conditional, options) {
2. **if**(conditional) {
3. **return** options.fn(**this**);
4. }
5. });

当使用条件判断的时候，常常需要在判断结果为false的时候插入相应的HTML块，Handlebar使用else解决这个问题。

1. {{#if isActive}}
2. **<img** **src**="star.gif" **alt**="Active"**>**
3. {{else}}
4. **<img** **src**="cry.gif" **alt**="Inactive"**>**
5. {{/if}}

Handlebar使用options.inverse来生成else的内容，如果没有inverse函数，模板会自动创建一个空函数。

1. Handlebars.registerHelper('if', **function**(conditional, options) {
2. **if**(conditional) {
3. **return** options.fn(**this**);
4. } **else** {
5. **return** options.inverse(**this**);
6. }
7. });

### Hash 参数

和普通的helper一样，块级helper可以接受一个hash作为最后一个参数。我们来改进一下listhelper，让<ul>接受任意多的属性。

1. {{#list nav id="nav-bar" class="top"}}
2. **<a** **href**="{{url}}"**>**{{title}}**</a>**
3. {{/list}}

Handlebar把最后的hash传给opton.hash，这样可以接受任意参数，如果模板没有传hash参数，option.hash会是{}。

1. Handlebars.registerHelper('list', **function**(context, options) {
2. **var** attrs = Em.keys(options.hash).map(**function**(key) {
3. key + '="' + options.hash[key] + '"';
4. }).join(" ");
6. **return** "<ul " + attrs + ">" + context.map(**function**(item) {
7. **return** "<li>" + options.fn(item) + "</li>";
8. }).join("\n") + "</ul>";
9. });

块级Helper还可以传递私有变量，这样就可以传入数据上下文之外的数据。

例如，我们生成一个list，包含当前列的序号。

1. {{#list array}}
2. {{@index}}. {{title}}
3. {{/list}}
4. Handlebars.registerHelper('list', **function**(context, options) {
5. **var** out = "<ul>", data;
7. **for** (**var** i=0; i<context.length; i++) {
8. **if** (options.data) {
9. data = Handlebars.createFrame(options.data || {});
10. data.index = i;
11. }
13. out += "<li>" + options.fn(context[i], { data: data }) + "</li>";
14. }
16. out += "</ul>";
17. **return** out;
18. });

通过data传递私有变量。

### 空白处理

使用~会省略模板任意一侧的空格

1. {{#each nav ~}}
2. **<a** **href**="{{url}}"**>**
3. {{~#if test}}
4. {{~title}}
5. {{~^~}}
6. Empty
7. {{~/if~}}
8. **</a>**
9. {{~/each}}

使用下面的数据

1. {
2. nav: [
3. {url: 'foo', test: **true**, title: 'bar'},
4. {url: 'bar'}
5. ]
6. }

结果输出无换行和空格

1. **<a** **href**="foo"**>**bar**</a><a** **href**="bar"**>**Empty**</a>**

### 子表达式

子表达式允许你调用多个helper，内部表达式的结果可以当做参数传递给外部的表达式。子表达式有括号分割。

1. {{outer-helper (inner-helper 'abc') 'def'}}

上例中，'abc'会传递给inner-helper，同时inner-helper的返回值会当成outer-helper的第一个参数。

//=================20170922

你可以使用<script>标签引入handlebars模板：

1. **<script** **id**="entry-template" **type**="text/x-handlebars-template"**>**
2. template content
3. **</script>**

在javascript中使用Handlebars.compile编译模板：

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

你也可以预编译你的模板，然后只需引入更小的运行时库（handlebars.runtime.js），避免在浏览器中编译，提高性能，这在移动设备中显得更重要。

[**更多内容：预编译**](http://keenwon.com/1013.html)

传入数据上下文（context），handlebars会执行并生成HTML：

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

得到的结果是：

1. **<div** **class**="entry"**>**
2. **<h1>**My New Post**</h1>**
3. **<div** **class**="body"**>**
4. This is my first post!
5. **</div>**
6. **</div>**

[**更多内容：执行handlebars**](http://keenwon.com/1016.html)

**HTML编码**

在handlebars里，{{expression}}会返回一个经过编码的HTML，如果你不希望被编码，可以使用{{{

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

使用这样的数据上下文：

1. {
2. title: "All about <p> Tags",
3. body: "<p>This is a post about &lt;p&gt; tags</p>"
4. }

结果是：

1. **<div** **class**="entry"**>**
2. **<h1>**All About &lt;p&gt; Tags**</h1>**
3. **<div** **class**="body"**>**
4. **<p>**This is a post about &lt;p&gt; tags**</p>**
5. **</div>**
6. **</div>**

handlebars不会编码Handlebars.SafeString。如果你自定义一个helper，返回一段HTML代码，你需要返回new Handlebars.SafeString(result)。此时，你需要手动对内容进行编码：

1. Handlebars.registerHelper('link', **function**(text, url) {
2. text = Handlebars.Utils.escapeExpression(text);
3. url = Handlebars.Utils.escapeExpression(url);
5. **var** result = '<a href="' + url + '">' + text + '</a>';
7. **return** **new** Handlebars.SafeString(result);
8. });

这里将会对传入的参数进行编码，返回值是“安全的”，所以就算你不使用{{{，handlebars也不会再次编码了。

**块表达式**

块表达式允许你定义helper，用不同的数据上下文（context）调用一段模板。下面我们定义一个生成列表的helper：

1. {{#list people}}{{firstName}} {{lastName}}{{/list}}

如果我们的数据是这样的：

1. {
2. people: [
3. {firstName: "Yehuda", lastName: "Katz"},
4. {firstName: "Carl", lastName: "Lerche"},
5. {firstName: "Alan", lastName: "Johnson"}
6. ]
7. }

我们创建一个叫list的helper来生成列表，helper接受people作为第一个参数，一个option对象（hash）作为第二个参数。option包含一个属性fn，他可以调用一个context就像普通模板一样。

1. Handlebars.registerHelper('list', **function**(items, options) {
2. **var** out = "<ul>";
4. **for**(**var** i=0, l=items.length; i<l; i++) {
5. out = out + "<li>" + options.fn(items[i]) + "</li>";
6. }
8. **return** out + "</ul>";
9. });

执行后，得到：

1. **<ul>**
2. **<li>**Yehuda Katz**</li>**
3. **<li>**Carl Lerche**</li>**
4. **<li>**Alan Johnson**</li>**
5. **</ul>**

块表达式有很多特性，例如，可以创建一个else块（内置的if helper就有else块）。另外，如果options.fn(context)对内容编码过了，handlebars就不会helper内容进行编码了，否则就编码两次了。

[**更多内容：块表达式**](http://keenwon.com/1033.html)

**Handlebars 路径**

Handlebars支持简单的路径

1. **<p>**{{name}}**</p>**

也支持嵌套路径，可以查找下一级的属性

1. **<div** **class**="entry"**>**
2. **<h1>**{{title}}**</h1>**
3. **<h2>**By {{author.name}}**</h2>**
5. **<div** **class**="body"**>**
6. {{body}}
7. **</div>**
8. **</div>**

此模板使用下面的数据：

1. **var** context = {
2. title: "My First Blog Post!",
3. author: {
4. id: 47,
5. name: "Yehuda Katz"
6. },
7. body: "My first post. Wheeeee!"
8. };

嵌套路径同样支持../，

1. **<h1>**Comments**</h1>**
3. **<div** **id**="comments"**>**
4. {{#each comments}}
5. **<h2><a** **href**="/posts/{{../permalink}}#{{id}}"**>**{{title}}**</a></h2>**
6. **<div>**{{body}}**</div>**
7. {{/each}}
8. **</div>**

尽管链接在打印出的时候，是以comments为上下文的，但是它可以访问到上一级的上下文（context）找到permalink。

../引用上一级的作用域，直接看一下上面模板对应的数据就明白了

1. **var** data = {
2. permalink:'http://keenwon.com',
3. comments: [
4. {id:1,title:'链接1',body:'链接1'},
5. {id:2,title:'链接2',body:'链接2'}
6. ]
7. };

Handlebars可以通过this引用解决helpers和数据命名冲突的问题。

1. <p>{{./name}} or {{**this**/name}} or {{**this**.name}}</p>

**模板注释**{{! }}**or**{{!-- --}}

你可以在 handlebars 里加注释：

1. **<div** **class**="entry"**>**
2. {{! only output this author names if an author exists }}
3. {{#if author}}
4. **<h1>**{{firstName}} {{lastName}}**</h1>**
5. {{/if}}
6. **</div>**

注释不会出现在输出结果里，如果想要显示出来，可以使用html的注释（会被执行，然后以注释的形式显示，所以如果html注释内有错，还是会报错）

1. **<div** **class**="entry"**>**
2. {{! 这个注释不会显示在输出结果中 }}
3. <!-- 会显示 -->
4. **</div>**

所有注释必须包含结束标签}}，多行注释可以使用{{!-- --}}

**Helpers**

Handlebars Helpers可以读取到模板中的任何数据上下文，你可以使用Handlebars.registerHelper注册一个helpers。

1. **<div** **class**="post"**>**
2. **<h1>**By {{fullName author}}**</h1>**
3. **<div** **class**="body"**>**{{body}}**</div>**
5. **<h1>**Comments**</h1>**
7. {{#each comments}}
8. **<h2>**By {{fullName author}}**</h2>**
9. **<div** **class**="body"**>**{{body}}**</div>**
10. {{/each}}
11. **</div>**

然后使用如下的数据上下文和Helpers：

1. **var** context = {
2. author: {firstName: "Alan", lastName: "Johnson"},
3. body: "I Love Handlebars",
4. comments: [{
5. author: {firstName: "Yehuda", lastName: "Katz"},
6. body: "Me too!"
7. }]
8. };
10. Handlebars.registerHelper('fullName', **function**(person) {
11. **return** person.firstName + " " + person.lastName;
12. });

结果是：

1. **<div** **class**="post"**>**
2. **<h1>**By Alan Johnson**</h1>**
3. **<div** **class**="body"**>**I Love Handlebars**</div>**
5. **<h1>**Comments**</h1>**
7. **<h2>**By Yehuda Katz**</h2>**
8. **<div** **class**="body"**>**Me Too!**</div>**
9. **</div>**

使用this可以访问到当前的上下文

1. **<ul>**
2. {{#each items}}
3. **<li>**{{agree\_button}}**</li>**
4. {{/each}}
5. **</ul>**

使用如下的Helpers和数据上下文

1. **var** context = {
2. items: [
3. {name: "Handlebars", emotion: "love"},
4. {name: "Mustache", emotion: "enjoy"},
5. {name: "Ember", emotion: "want to learn"}
6. ]
7. };
9. Handlebars.registerHelper('agree\_button', **function**() {
10. **return** **new** Handlebars.SafeString(
11. "<button>I agree. I " + **this**.emotion + " " + **this**.name + "</button>"
12. );
13. });

结果是：

1. **<ul>**
2. **<li><button>**I agree. I love Handlebars**</button></li>**
3. **<li><button>**I agree. I enjoy Mustache**</button></li>**
4. **<li><button>**I agree. I want to learn Ember**</button></li>**
5. **</ul>**

如果你的helpers返回一个html片段，不想被编码。必须new一个Handlebars.SafeString返回出来。

**内置的Helpers**

**The**with**Block Helper**

通常，Handlebars会将数据上下文传入编译方法：

1. **var** source = "<p>{{lastName}}, {{firstName}}</p>";
2. **var** template = Handlebars.compile(source);
3. template({firstName: "Alan", lastName: "Johnson"});

结果：

1. **<p>**Johnson, Alan**</p>**

使用with可以改变当前的上下文

1. **<div** **class**="entry"**>**
2. **<h1>**{{title}}**</h1>**
4. {{#with author}}
5. **<h2>**By {{firstName}} {{lastName}}**</h2>**
6. {{/with}}
7. **</div>**

数据上下文如下：

1. {
2. title: "My first post!",
3. author: {
4. firstName: "Charles",
5. lastName: "Jolley"
6. }
7. }

结果：

1. **<div** **class**="entry"**>**
2. **<h1>**My first post!**</h1>**
4. **<h2>**By Charles Jolley**</h2>**
5. **</div>**

**The**each**block helper**

你可以使用内置的each helper生成列表，可以使用this访问当前项。

1. **<ul** **class**="people\_list"**>**
2. {{#each people}}
3. **<li>**{{this}}**</li>**
4. {{/each}}
5. **</ul>**

数据上下文如下：

1. {
2. people: [
3. "Yehuda Katz",
4. "Alan Johnson",
5. "Charles Jolley"
6. ]
7. }

结果：

1. **<ul** **class**="people\_list"**>**
2. **<li>**Yehuda Katz**</li>**
3. **<li>**Alan Johnson**</li>**
4. **<li>**Charles Jolley**</li>**
5. **</ul>**

你可以在任何上下文里，使用this引用当前的上下文

另外，还可以使用{{else}}块，当列表内容为空的时候会显示{{else}}的内容

1. {{#each paragraphs}}
2. <p>{{**this**}}</p>
3. {{**else**}}
4. <p **class**="empty">暂无内容</p>
5. {{/each}}

在each中循环每一项的时候，可以使用{{@index}}获取当前的序号。

1. {{#each array}}
2. {{@index}}: {{**this**}}
3. {{/each}}

对于object，可以使用{{key}}获取当前的key。

1. {{#each object}}
2. {{@key}}: {{**this**}}
3. {{/each}}

在迭代的过程中，可以使用@first和@last判断当前的第一步和最后一步，对于object，只有@first可用。

**The**if**block helper**

可以使用if helper有条件的渲染block，如果是false, undefined, null, "" 或者 []（a “falsy” value），Handlebars不会渲染此block.

1. **<div** **class**="entry"**>**
2. {{#if author}}
3. **<h1>**{{firstName}} {{lastName}}**</h1>**
4. {{/if}}
5. **</div>**

如果使用的是空的数据上下文（例如{}），author会返回undefined，结果是：

1. **<div** **class**="entry"**>**
2. **</div>**

当使用块表达式，可以使用{{else}}来指定一个“片段”，当结果是 falsy value 的时候呈现

1. **<div** **class**="entry"**>**
2. {{#if author}}
3. **<h1>**{{firstName}} {{lastName}}**</h1>**
4. {{else}}
5. **<h1>**Unknown Author**</h1>**
6. {{/if}}
7. **</div>**

**The**unless**block helper**

unless的作用和if刚好相反，但表达式返回falsy value的时候渲染block

1. **<div** **class**="entry"**>**
2. {{#unless license}}
3. **<h3** **class**="warning"**>**WARNING: This entry does not have a license!**</h3>**
4. {{/unless}}
5. **</div>**

如果当前上下文的license返回一个falsy value，Handlebars会输出警告信息，否则什么都不输出。

**The**log**block helper**

log helper允许执行模板的时候输出当前上下文的状态

1. {{log "Look at me!"}}

信息传给Handlebars.logger.log，重写这个函数可以实现自定义的log。

**内置的工具**

Handlebars提供各种工具方法，在Handlebars.Util命名空间下。