Handlebars 为你提供了一个可以毫无挫折感的高效率书写 **语义化的模板** 所必需的一切。

Mustache 模板和 Handlebars 是兼容的，所以你可以把Mustache模板拿来导入到Handlebars中，并开始使用Handlebars所提供的更丰富的功能。

开始

Handlebars模板看起来就像是正常的Html，并使用了嵌入的 handlebars 表达式。

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

handlebars表达式，是以 `{{` 开始，跟一些内容，然后以 `}}` 结束。
更多资料：表达式 (http://handlebarsjs.com/expressions.html)

你可以通过 `<script>` 标签把一段模板加载到浏览器中。

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

在 JavaScript 中使用 `Handlebars.compile` 来编译模板。

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

还可以预编译模板。这样的话，就只需要一个更小的运行时库文件，并且对性能来说是一个极大的节约，因为这样就不必在浏览器中编译模板了。这点在移动版的开发中就更显的非常重要了。
更多资料：预编译 (http://handlebarsjs.com/precompilation.html)

只需传递一个上下文context执行模板，即可得到返回的 HTML 的值
（译者注：通常来说在 js 中上下文就决定了当前函数的this的指向）

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

得到下面的HTML

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

更多资料：执行 (http://handlebarsjs.com/execution.html)

Handlebars的 `{{expression}}` 表达式会返回一个 HTML编码 `HTML-escape` 过的值。如果不希望Handlebars来编码这些值，使用三个大括号即可： `{{{` 。

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

使用这段上下文（数据）：

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

会得到如下结果：

```
<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 不会再对 `Handlebars.SafeString` 安全字符串进行编码。如果你写的 helper 用来生成 HTML，就经常需要返回一个 `new Handlebars.SafeString(result)` 。在这种情况下，你就需要手动的来编码参数了。

```
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);
});
```

这样来编码传递进来的参数，并把返回的值标记为 安全，这样的话，即便不是哟给你“三个大括号”，Handlebars 就不会再次编码它了。

块级表达式

块级表达式允许你定义一个helpers，并使用一个不同于当前的上下文（context）来调用你模板的一部分。现在考虑下这种情况，你需要一个helper来生成一段 HTML 列表：

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

并使用下面的上下文（数据）：

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

此时需要创建一个 名为 `list` 的 helper 来生成这段 HTML 列表。这个 helper 使用 `people` 作为第一个参数，还有一个 `options` 对象（hash哈希）作为第二个参数。这个 options 对象有一个叫 `fn` 的属性，你可以传递一个上下文给它（fn），就跟执行一个普通的 Handlebars 模板一样：

```
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>";
});
```

执行之后，这个模板就会渲染出：

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

块级的 helpers 还有很多其他的特性，比如可以创建一个 `else` 区块（例如，内置的 if helper 就是用 else）。

注意，因为在你执行 `options.fn(context)` 的时候，这个 helper 已经把内容编码一次了，所以 Handlebars 不会再对这个 helper 输出的值进行编码了。如果编码了，这些内容就会被编码两 次！
更多资料：块级Helpers (http://handlebarsjs.com/block_helpers.html)
译文在此 (http://blog.segmentfault.com/chao2/1190000000347965)

Handlebars 路径

Handlebars 支持简单的路径，就像Mustache那样。

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

Handlebars 同样也支持嵌套的路径，这样的话就可以在当前的上下文中查找内部嵌套的属性了。

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

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

上面的模板使用下面这段上下文：

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

这样一来 Handlebars 就可以直接把JSON数据拿来用了。

巢状嵌套的 handlebars 路径也可以使用 `../` ， 这样会把路径指向父级（上层）上下文。

```
<h1>Comments</h1>

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

尽管 a 链接在输出时是以 comment 评论为上下文的，但它仍然可以退回上一层的上下文（post上下文）并取出permalink（固定链接）值。
（译者注）上下文数据应该如下所示（源文档并没有给出）

```
var context = {
  post: {
    body:'这是文章内容',
    permalink: 'http://xx.com/xx',
    comments:[{
      title:'这篇文章不错，赞一个'
    },{
      title:'好文要顶！'
    }]
  }
}
```

`../` 标识符表示对**模板的父级作用域的引用**，**并不表示在上下文数据中的上一层。**这是因为块级 helpers 可以以任何上下文来调用一个块级表达式，所以这个【上一层】的概念用来指模板作用域的父级更有意义些。

Handlebars也允许通过一个 `this` 的引用来解决 helpers 和 数据字段间的名字冲突：

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

上面的这一种方式都会将 `name` 字段引用到当前上下文上，而不是 helper 上的同名属性。

模板注释：{{! }} 或 {{!-- --}}

你可以在 handlebars 代码中加注释，就跟在代码中写注释一样。对于有一定程度的逻辑的部分来说，这倒是一个很好的实践。

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

注释是不会最终输出到返回结果中的。如果你希望把注释展示出来，就使用 HTML 的注释就行了。

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

所有注释都必须有 `}}` ，一些多行注释可以使用 `{{!-- --}}` 语法。

Helpers

Handlebars 的 helpers 在模板中可以访问任何的上下文。可以通过 `Handlebars.registerHelper` 方法注册一个 helper。

```
<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>
```

当时用下面的上下文数据和 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;
});
```

会得到如下结果：

```
<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 会把当前的上下文作为函数中的 `this` 上下文。

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

当使用下面的 this上下文 和 helpers：

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

Handlebars.registerHelper('agree_button', function() {
  return new Handlebars.SafeString(
    "<button>I agree. I " + this.emotion + " " + this.name + "</button>"
  );
});
```

会得到如下结果：

```
<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>
```

如果你不希望你的 helper 返回的 HTML 值被编码，就请务必返回一个 `new Handlebars.SafeString`

内置的 Helpers

`with` helper

一般情况下，Handlebars 模板在计算值时，会把传递给模板的参数作为上下文。

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

结果如下：

```
<p>Johnson, Alan</p>
```

不过也可以在模板的某个区域切换上下文，使用内置的 `with` helper即可。

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

  {{#with author}}
  <h2>By {{firstName}} {{lastName}}</h2>
  {{/with}}
</div>
```

在使用下面数据作为上下文时：

```
{
  title: "My first post!",
  author: {
    firstName: "Charles",
    lastName: "Jolley"
  }
}
```

会得到如下结果：

```
<div class="entry">
  <h1>My first post!</h1>

  <h2>By Charles Jolley</h2>
</div>
```

each helper

你可以使用内置的 `each` helper 来循环一个列表，循环中可以使用 `this` 来代表当前被循环的列表项。

```
<ul class="people_list">
  {{#each people}}
  <li>{{this}}</li>
  {{/each}}
</ul>
```

使用这个上下文：

```
{
  people: [
    "Yehuda Katz",
    "Alan Johnson",
    "Charles Jolley"
  ]
}
```

会得到：

```
<ul class="people_list">
  <li>Yehuda Katz</li>
  <li>Alan Johnson</li>
  <li>Charles Jolley</li>
</ul>
```

事实上，可以使用 `this` 表达式在任何上下文中表示对当前的上下文的引用。

还可以选择性的使用 `else` ，当被循环的是一个空列表的时候会显示其中的内容。

```
{{#each paragraphs}}
  <p>{{this}}</p>
{{else}}
  <p class="empty">No content</p>
{{/each}}
```

在使用 `each` 来循环列表的时候，可以使用 `{{@index}}` 来表示当前循环的索引值。

```
{{#each array}}
  {{@index}}: {{this}}
{{/each}}
```

对于 object 类型的循环，可以使用 `{{@key}}` 来表示：

```
{{#each object}}
  {{@key}}: {{this}}
{{/each}}
```

`if` helper

`if` 表达式可以选择性的渲染一些区块。如果它的参数返回 `false` ， `undefined` ， `null` ， `""` 或 `[]` （**译注：还有 `0` **）（都是JS中的“假”值），Handlebars 就不会渲染这一块内容：

```
<div class="entry">
  {{#if author}}
  <h1>{{firstName}} {{lastName}}</h1>
  {{/if}}
</div>
```

当时用一个空对象（ `{}` ）作为上下文时，会得到：

```
<div class="entry">
</div>
```

在使用 `if` 表达式的时候，可以配合 `{{else}}` 来使用，这样当参数返回 假 值时，可以渲染 else 区块：

```
<div class="entry">
  {{#if author}}
    <h1>{{firstName}} {{lastName}}</h1>
  {{else}}
    <h1>Unknown Author</h1>
  {{/if}}
</div>
```

`unless` helper

`unless` helper 和 `if` helper 是正好相反的，当表达式返回假值时就会渲染其内容：

```
<div class="entry">
  {{#unless license}}
  <h3 class="warning">WARNING: This entry does not have a license!</h3>
  {{/unless}}
</div>
```

如果在当前上下文中查找 `license` 返回假值，Handlebars 就会渲染这段警告信息。反之，就什么也不输出。

`log` helper

`log` helper 可以在执行模板的时候输出当前上下文的状态。

```
{{log "Look at me!"}}
```

这样会把委托信息发送给 `Handlebars.logger.log` ，而且这个函数可以重写来实现自定义的log。