开始使用Handlebars

开始

Handlebars模版看起来像嵌入了handlebars表达式的普通的HTML。

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

一个handlebars表达式由一个 `{{` ，一些内容，后面跟着 `}}` 组成。

你可以通过 `<script>` 标签把模版加载到浏览器中。

```
<scirpt id="entry-template" type="text/x-handlebars-template">
    <div class="entry">
        <h1>{{title}}</h1>
        <div class="body">
            {{body}}
        </div>
    </div>
</script>
```

在Javascript中使用 `Handlebars.compile` 编译一个模版

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

同样有可能预编译你的模版。这将导致对运行库的更小需求和不需要在浏览器上面编译，使得性能更到更好的体验。这在移动设备上面开发时特别重要。

传递一个上下文context运行Handlebars模版，得到经过模版渲染后的HTML结果。

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

渲染的结果的HTML为

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

HTML转义

Handlebars通过 `{{expression}}` 表达式返回HTML-escape转义的HTML编码。如果你不想Handlebars对这些内容进行转义，那么就使用“triple-stash”， `{{{` .

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

使用的上下文如下：

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

模版渲染结果为：

```
<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不会对 `Handlebars.SafeString` 进行转义（编码）。如果你写一个helper产生HTML，那么你通常希望返回一个 `new Handlebars.SafeString(result)` ，在这些奇怪的情况下，你会希望手动去转义参数。

```
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);
});
```

这将会对传入对参数进行转义，但是会标记返回值为安全响应，因此Handlebars不会尝试去转义它，甚至“triple-stash”也不起作用。

块表达式(Block Expression)

块表达式允许你自己定义的模版，并使用一个不同于当前的上下文来调用模版的一部分。这些block helpers由 `#` 紧接着helper的名称声明定义，并需要一个闭合的Mustache `/` 加上相同的名称

现在考虑一个产生HTML列表的helper：

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

如果我们有如下上下文：

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

我们可以创建一个命名为 `list` 的helper去产生我们的HTML列表。helper会将 `people` 作为它的第一个参数，而一个options对象（hash哈希）作为第二个参数。options对象包含一个名为 `fn` 的属性，这个属性让你可以像调用一个正常的Handlebars模版一样调用一个上下文。

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

当程序运行时，模版会渲染为：

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

Block helpers有更多的特征，比如说创建一个 `else` 块（例如，通过内置的if helper）。

因为当你调用 `options.fn(context)` 的时候，block helper把内容都转义了，Handlebars将不会对block helper的结果进行转义。如果转义了的话，里面的内容就会被两次转义。

Handlebars的路径（Paths）

Handlebars支持简单的路径，就好像Mustache。

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

Handlebars同样支持嵌套的路径，这使得它可以在当前的上下文中查找内部嵌套的属性。

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

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

这个模版通过下面的上下文渲染：

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

这使得有可能去使用Handlebars模版，这些模版中用到的上下文可以包含更多的JSON对象。

嵌套的handlebars路径可以通过包含 `../` 来访问他们的父级上下文的路径。

```
<h1>Comments</h1>

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

尽管链接是以Comment为上下文而输出，它仍然会返回主要的上下文（the post）从而取回它的永久链接（permalink）。

可能使用了如下的上下文：

```
var context = {
    post: {
        body: 'This is content',
        permalink: 'futurefeeling.github.io',
        comments: [{
            title: 'This is a Title'
        }, {
            title: 'Anther title'
        }]
    }
}
```

`../` 标志符表示对模版的父级作用域的引用，并不是表示在上下文数据中的上一层。这是因为block helpers可以以任何上下文来调用一个块级表达式，所以“one level up”的概念并不是特别有意义的，除了作为父级模版作用域的引用。

Handlebars同样允许经由 `this` 的引用解决helpers和data块之间的名称冲突：

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

上面的情况会造成 `name` 字段引用到当前的上下文上，而不是helper中相同的名称。

用 {{!– –}} 和 {{! }} 进行模版注释（Comment）

你可以在handlebars代码中使用注释就好像在你自己的代码中一样。通常由于这有一定的逻辑，所以这是一个好的事件。

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

注释不会作为结果输出。如果你想注释被看到，只要使用html的注释就好了，他们就会被看到。

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

任何的注释都要使用 `}}` 或者多行注释应该使用 `{{!-- --}}` 语法。

Helpers(实在是想不出这个该翻译成啥)

Handlebars helpers可以在模版中访问任何上下文。你可以通过 `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>
```

当使用下面的上下文和helpers时：

```
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;
});
```

渲染的结果为：

```
<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得到的当前多上下文是 `this` 指向的上下文。

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

当使用下面的上下文和helper：

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

渲染结果为：

```
<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>
```

如果你的helper返回的HTML是你不想要转义的，请确定返回的 `Handlebars.SafeString`

Partials(泛音？)

Handlebars partials 允许通过创建分享的模版去减少代码。渲染这个模版：

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

    <h1>Comments</h1>

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

当使用下面的partial和上下文：

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

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

渲染结果为：

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

    <h2>Comments</h2>
    <div class="body">Me Too!</div>
</div>
```

内置Helpers (Built-In Helpers)

Handlebars提供了很多内置的helpers，比如 `if` 条件语句和 `each` 迭代。