

邦彦

Mastering `{{Mustache}}`

Mastering == 从入门到精通

2011.12.19

substitute

模版：

```
<a href="{url}">{title}</a>
```

数据：

```
{url:"http://www.taobao.com",title:"淘宝网"}
```

替换：

```
substitute(template, data)
```

```
var template = '*****{a}*****{b}*****',
    data = {a:"&&&&&",b:"@@@@@"};

template.replace(/\{([^\}]+)\}/g, function
(match, key) {

    return (data[key] !== undefined) ?
data[key] : '';

});
```

这里 data 表示传入的 JSON 对象

原理

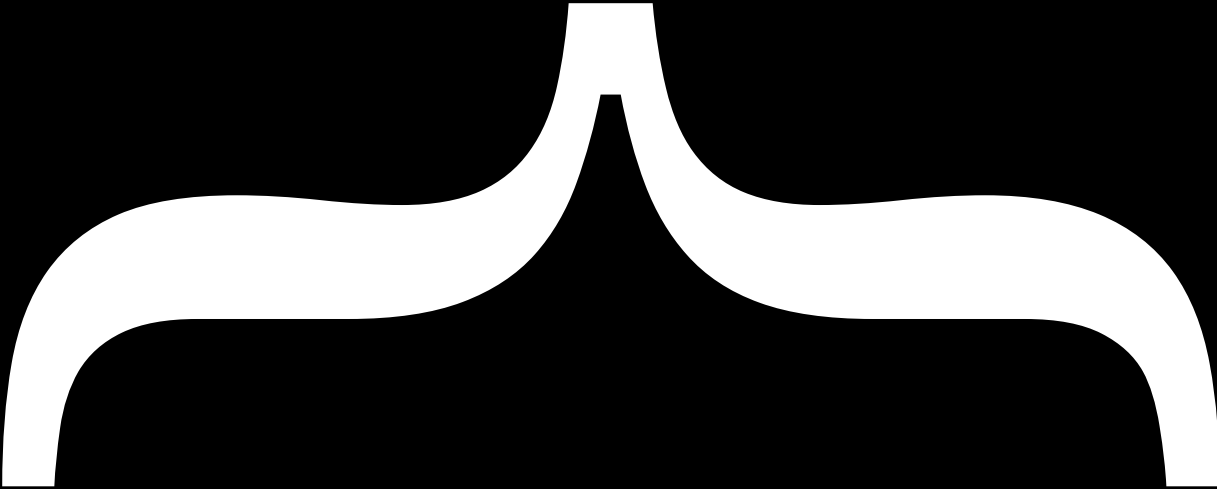
`replace` 函数结合正则匹配，对字符串模版执行搜索替换

YUI3:

`Y.substitute(s, o, f, recurse)`

KISSY:

`KISSY.substitute(str, o, regexp)`



开始入门

少逻辑

Logic-less template.

多语言支持

Support JavaScript, Ruby, Python, PHP, Java,
node.js...

编辑器插件

Support TextMate, Vim, Emacs, Coda.

Demo

```
<h1>{{header}}</h1>
{{#bug}}
{{/bug}}
{{#items}}
    {{#first}}
        <li><strong>{{name}}</strong></li>
    {{/first}}
    {{#link}}
        <li><a href="{{url}}">{{name}}</a></li>
    {{/link}}
{{/items}}
{{#empty}}
    <p>The list is empty.</p>
{{/empty}}
```

Demo

```
{
  "header": "Colors",
  "items": [
    {"name": "red", "first": true, "url":
"#Red"},
    {"name": "green", "link": true, "url":
"#Green"},
    {"name": "blue", "link": true, "url":
"#Blue"}
  ],
  "empty": false
}
```

Demo

```
<h1>Colors</h1>  
<li><strong>red</strong></li>  
<li><a href="#Green">green</a></li>  
<li><a href="#Blue">blue</a></li>
```

通过标记字段为 `true` 或 `false` 实现仅有的逻辑操作功能

权衡

缺少逻辑能力 =>

模版 => 简洁

数据 => 复杂、冗余

开始精通

`{}` => `{{}}`

标签形式和 KISSY Template 一致

`if, else, 循环` =>

`{{#tag}}{{/tag}}`

通过纯粹标签实现

标签说明

Mustache Tag Types

变量

Variables

模版：

- * {{name}}
- * {{age}}
- * {{company}}
- * {{{company}}}

数据：

```
{"name": "Chris", "company": "<b>GitHub</b>"}
```

+

结果：

- * Chris
- *
- * GitHub
- * GitHub

区块

Sections

情况一

值为 `false` 或空列表，标签对内的信息不被展现

模版：

Shown.

```
{{#nothin}}
```

Never shown!

```
{{/nothin}}
```

数据：

```
{"person": true,}
```



结果：

Shown.

情况二

- 1) 如果键名存在并且值为非false，执行输出
- 2) 如果值为非空列表，以循环形式逐一输出

模版：

```
{{#repo}}  
  <b>{{name}}</b>  
{{/repo}}
```

+

数据：

```
{  
  "repo": [  
    { "name": "resque" },  
    { "name": "hub" },  
    { "name": "rip" }  
  ]  
}
```

结果：

```
<b>resque</b>  
<b>hub</b>  
<b>rip</b>
```

情况三

值为可调用对象(`callable object`)时（通常是匿名函数），该对象将被调用，并同时当前取值作为参数传入

模版：

```
{{#wrapped}}  
{{name}} is awesome.  
{{/wrapped}}
```



数据：

```
{  
  "name":  
  "Willy", "wrapped":  
  function() {  
    return  
      function(text) {  
        return "<b>"  
+ render(text) + "</b>"  
      }  
  }  
}
```

结果：

```
<b>Willy is awesome.</b>
```

情况四

值为非 `false` 且非列表，则
进行单一条目渲染

模版：

```
{{#person}}  
Hi {{name}}!  
{{/person?}}
```

数据：

```
{"person": { "name":  
"Jon" }}
```



结果：

Hi Jon!

反向区块

Inverted Sections

```
{{^person}}  
{{/person}}
```

输出反向情况

换句话说，就是当值不存在、
false、空列表时进行输出

模版：

```
{{#repo}}  
  <b>{{name}}</b>  
{{/repo}}  
{{^repo}}  
  No repos :(  
{{/repo}}
```

数据：

```
{"repo": []}
```



结果：

```
No repos :(
```

注释

Comments

模版：

```
<h1>Today{{! ignore me }}.</h1>
```

结果：

```
<h1>Today.</h1>
```

局部模版

Comments

```
var view = {
  name: "Joe",
  winnings: {
    value: 1000
  }
};
```

```
var template = "Welcome, {{name}}! {{>winnings}}"
var partials = {
  winnings: "You just won ${{value}}"
};
```

```
var output = Mustache.to_html(template, view,
partials)
```

output will be:

Welcome, Joe! You just won \$1000

模版结构

结构（建议）：

```
<script id="example-tpl" type="text/template">  
  <div>{{mustache}} template here...</div>  
</script>
```

MIME

浏览器不懂 `text/template`

=> 忽略该 `script` 标签对

=> 放置任何代码片段

MIME

<script><script>

<script type="text/javascript"><script>

<script type="application/javascript"><script>

<script type="application/x-javascript"><script>

jquery:

<script type="text/x-jquery-tmpl"><script>

backbone.js:

<script type="text/template"><script>

请求数据

XHR or JSONP:

```
success : function (data) {  
  
    if (data.status === 'ok') {  
        //do sth a...  
    } else {  
        //do sth b...  
    }  
  
}
```

渲染过程

`data.status === 'ok' :`

- a. 数据预处理 `data.xxx = function () {...}`
- b. 执行渲染 `Mustache.to_html(template, data);`
- c. 构建 DOM 树
- d. 事件绑定等后续操作

完

Game over