表达式（More Expression）

基本使用（Basic Usage）

最简单的Handlebars表达式是一个简单的标识符.

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

这个表达式的意思是：在当前的上下文中查找 `title` 属性。Block helpers可能会操作当前的上下文，但是它们不会改变表达式的基本意义。

实际上，它意味着：查找一个名为 `title` 的helper，然后执行上述操作。但是我们很快就会到达的。

Handlebars表达式也可以成为圆点分隔（dot-separated）的路径

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

这个表达式意思是：在当前的上下文查找 `article` 属性，然后在查找的结果中查找 `title` 属性。

Handlebars同样支持一个不推荐使用的 `/` 语法，所以你也可以这样写你的模版：

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

标志符可以是任何unicode字符，除了如下字符外：

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

为了引用一个不是有效的标识符，你可以使用部分文字符号（segment-literal notation）， `[` **：**

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

在上面的例子中，模版会将每个 `each` 参数大致等效于这个javascript：
`articles[10]['#commments']`

你可能不包含闭合 `]` 在你的路径文本（path-literal），但是其他的字符是fair game（靠，什么鬼）。

Handlebars通过{{expresssion}}返回HTML编码（HTML-escape）。如果你不希望Handlebars去转义一个值，那么可以使用“triple-stash”，即 `{{{`

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

Helpers(依旧不知道怎么翻译这个东东)

一个Handlebars helper的调用是一个简单的标识符，后面跟着零个或者多个参数（用空格隔开）。每个参数是一个Handlebars表达式。

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

在这种情况下， `link` 作为Handlebars helper的名字，而 `story` 是helper的参数。Handlebars实际上通过和“Basic Usage”描述的方式去访问参数。

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

当经过helper返回HTML编码，你应该返回一个Handlebars SafeString，如果你不希望再默认情况下被转码到话。当使用所有未知的和不安全的数据的SafeString应该通过 `escapeExpression` 方法手动编码。

你同样可以使用简单的字符，数字或者布尔值作为Handlebars helpers的参数。

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

在这种情况下，Handlebars会为link helper传递两个参数：字符串 `"See more..."` 和在当前上下文访问 `story.url` 的结果。

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

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

你可以使用实际相同的带有动态文本的helper，这些动态文本是 `story.text` 的值。

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

Handlebars helper也可以接受一系列可选的键值对（key-value）作为它们最后的参数（与文档中的哈希参数（hash arguments）比较）

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

哈希参数中的值每个都要是简单的标识符，并且每个值都要是Handlebars表达式。这就意味着这些值可以是简单的标识符，路径或者是字符串。

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

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

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

Handlebars提供了额外的元数据，比如哈希参数（Hash arguments），作为helpers的最后一个参数。

Handlebars同样提供了一个机制，这个机制可以引用模版中一个block的helper。Block helpers接着可以引用它在任何上下文中选择的block零次或者多次。

子表达式（Subexpressions）

Handlebars提供子表达式的支持，子表达式允许你在一个single mustache中引用多个helpers，并且传递核心helper请求的结果作为参数给非核心的helpers。子表达式被插入语（parentheses）界定

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

在这种情形下，`inner-helper` 将被字符串参数 `abc` 引用，无论 `inner-helper` 函数返回任何值都将作为 `outer-helper` 的第一个参数。（而 `def` 将作为 `out-helper` 的第二个参数）

空白控制（Whitespace Control）

模版空白会被任何mustache语句的任何一边通过在大括号中增加一个 `~` 字符进行省略。应用在该一方的所有空白将删除直到那边的第一个Handlebars表达式或非空格字符

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

上下文如下：

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

渲染结果在一行上并且格式化了所有的空白：

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

这扩展了剥离一行的默认行为，这就是所谓的“standalone”helpers（只有一个block helper，comment，或者partial和whitespace）

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

渲染结果为：

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

Id 跟踪（Tracking）

可选的，**helpers**可以被用来查找给定值的参数的路径。这种模式可以通过 `trackIds` 编译器标识启动

```
{{foo bar.baz}}
```

将会通过 `bar.baz` 的值调用helper `foo` ，但是同样将会包含在 `options` 参数中的 `ids` 块中的文字字符串 `"bar.baz"` 。