Django内置模板标签

1. autoescape

- 控制自动转义是否可用，这种标签带有任何on和off作为参数的话，他将决定转义块内效果。
- 该标签会以一个endautoescape作为结束标签。
- 当自动转义生效时，所有变量内容会被转义成HTML输出（在所有过滤器生效后）这等同与

手动将escape筛选器应用于每个变量。

- 唯一一个例外是，变量或者通过渲染变量的代码，或者因为他已经应用了safe或escape过

滤器，已经被标记为“safe”

例如

```
{% autoescape on %}
    {{ body }}
{% endautoescape %}
```

2. block

- block 标签可以被子模板覆盖

3. comment

- 在{% comment %} 和 {% endcomment %}，之间的内容会被忽略，作为注释。在第一个

  标签可以插入一个可选的记录。比如，当要注释掉一些代码时，可以用此来记录代码被注

  释掉的原因。

例如:

```
<p>Rendered text with {{ pub_date|date:"c" }}</p>

{% comment "Optional note" %}
    <p>Commented out text with {{ create_date|date:"c" }}</p>
{% endcomment %}
```

comment标签不能嵌套使用.

4. Csrf_token

- 这个标签用于跨站请求伪造保护

5. cycle

- 每当这个标签被访问，则传出一个它的可迭代参数的元素。第一次访问返回第一个元素，第二

  次访问返回第二个参数，依次类推，一旦所有的变量都被访问过了，就会回到最开始的地方，

  重复下去。

  这个标签在循环中特别好用

```
{% for o in some_list %}
    <tr class="{% cycle 'row1' 'row2' %}">
        ...
    </tr>
{% endfor %}
```

第一次迭代产生的HTML引用了row1类，第二次则是row2类，如此类推。

- 也可以使用变量，例如，如果有两个模板变量，rowvalue1和rowvalue2,可以让他们的值像这

  样一样替换

```
{% for o in some_list %}
    <tr class="{% cycle rowvalue1 rowvalue2 %}">
        ...
    </tr>
{% endfor %}
```

- 被包含在cycle中的变量将会被转义，可以禁止自动转义

```
{% for o in some_list %}
    <tr class="{% autoescape off %}{% cycle rowvalue1 rowvalue2 %}{% endau
toescape %}">
        ...
    </tr>
{% endfor %}
```

- 能够混合使用变量和字符串

```
{% for o in some_list %}
    <tr class="{% cycle 'row1' rowvalue2 'row3' %}">
        ...
    </tr>
{% endfor %}
```

- 在某些情况下，可能需要连续引用一个当前循环的值，而不前进到下一个循环值。要达到这个

  目的，只需要使用“as”来给{% cycle %}一个别名，

```
{% cycle 'row1' 'row2' as rowcolors %}
```

- 设置别名后，可以通过将别名作为一个模板变量进行引用，从而随意在模板中插入当前循环的

  值。如果要将循环值移到独立于原始cycle标记的下一个值，可以使用另一个cycle标记并指定

  变量的名称。

模板：

```
<tr>
    <td class="{% cycle 'row1' 'row2' as rowcolors %}">...</td>
    <td class="{{ rowcolors }}">...</td>
</tr>
<tr>
    <td class="{% cycle rowcolors %}">...</td>
    <td class="{{ rowcolors }}">...</td>
</tr>
```

将输出：

```
<tr>
    <td class="row1">...</td>
    <td class="row1">...</td>
</tr>
<tr>
    <td class="row2">...</td>
    <td class="row2">...</td>
</tr>
```

- 默认情况下，当你在cycle标签中使用 `as` 关键字时，关于 `{% cycle %}` 的使用，会启动cycle并且直接产生第一个值。 如果你想要在嵌套循环中或者included模版中使用这个值，那么将会遇到困难。 如果你只是想要声明cycle，但是不产生第一个值，你可以添加一个 `silent` 关键字来作为cycle标签的最后一个关键字。 像这样：

```
{% for obj in some_list %}
    {% cycle 'row1' 'row2' as rowcolors silent %}
    <tr class="{{ rowcolors }}">{% include "subtemplate.html" %}</tr>
{% endfor %}
```

这将输出row1元素的列表，其中class在row1和row2之间交替。 子模板将在其上下文中访问rowcolors，并且该值将匹配包围它的<tr>的类。 如果省略row2关键字，则row1和silent将作为正常文本发出，

- The following template would output *nothing*, even though the second call to `{% cycle %}` doesn't specify `silent`:

```
{% cycle 'row1' 'row2' as rowcolors silent %}
{% cycle rowcolors %}
```

可以使用resetcycle标签制作{% 循环 %}标签从下一次遇到的第一个值重新启动。

6. debug

- 输出整个调试信息，包括当前上下文和导入的模块。

7. extends

- 表示当前模板继承自一个父模板

  这个标签可以有以下两种用法

  - `{% extends "base.html" %}` (要有引号).继承名为 `"base.html"` 的父模板
  - `{% extends variable %}` 使用 `variable` 如果变量被计算成一个字符串，Django将会把它看成是父模版的名字。 如果变量被计算到一个 `Template` 对象，Django将会使用那个对象作为一个父模版。

通常模板名称是相对于模板加载器的根目录。 字符串参数也可以是以./或../开头的相对路

径。 例如，假设以下目录结构：

```
dir1/
    template.html
    base2.html
    my/
        base3.html
base1.html
```

在template.html中，以下路径将有效：

```
{% extends "./base2.html" %}
{% extends "../base1.html" %}
{% extends "./my/base3.html" %}
```

8. filter

- 通过一个或多个过滤器对内容过滤。 作为灵活可变的语法，多个过滤器被管道符号相连接，且过滤器可以有参数。

例如:

```
{% filter force_escape|lower %}
    This text will be HTML-escaped, and will appear in all lowercase.
{% endfilter %}
```

9. firstof

- 输出第一个不为False参数。如果传入的所有变量都为 `False` ， 就什么也不输出。

例如:

```
{% firstof var1 var2 var3 %}
```

它等价于:

```
{% if var1 %}
    {{ var1 }}
{% elif var2 %}
    {{ var2 }}
{% elif var3 %}
    {{ var3 }}
{% endif %}
```

- 当然你也可以用一个默认字符串作为输出以防传入的所有变量都是False:

```
{% firstof var1 var2 var3 "fallback value" %}
```

- 标签auto-escapes是开启的， 你可以禁止自动转义:

```
{% autoescape off %}
```

```
    {% firstof var1 var2 var3 "<strong>fallback value</strong>" %}
{% endautoescape %}
```

- 如果只想要部分变量被规避，可以这样使用：

```
{% firstof var1 var2|safe var3 "fallback value"|safe %}

You can use the syntax {% firstof var1 var2 var3 as value %} to store the
 output inside a variable.
```

10. for

for

- 循环组中的每一个项目，并让这些项目在上下文可用。 举个例子，展示 `athlete_list` 中的每个成员：

```
<ul>
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
{% endfor %}
</ul>
```

- 可以利用 `{% for obj in list reversed %}` 反向完成循环。

  如果你需要循环一个包含列表的列表，可以通过拆分每一个二级列表为一个独立变量来达到目的。 举个例子，如果你的内容包括一个叫做 `points` 的(x,y) 列表，你可以像以下例子一样输出points列表：

```
{% for x, y in points %}
    There is a point at {{ x }},{{ y }}
{% endfor %}
```

- 如果你想访问一个字典中的项目，这个方法同样有用。 举个例子：如果你的内容包含一个叫做 `data` 的字典，下面的方式可以输出这个字典的键和值：

```
{% for key, value in data.items %}
    {{ key }}: {{ value }}
{% endfor %}
```

- 请记住，对于点运算符，字典键查找优先于方法查找。
- Therefore if the `data` dictionary contains a key named `'items'`, `data.items` will return `data['items']` instead of `data.items()`.

- 如果要在模板中使用这些方法（`items`， `values`， `keys` 等），请避免添加名为字典方法的键。
- The current iteration of the loop (1-indexed)

变量	描述
`forloop.counter`	循环的当前迭代（1索引）
`forloop.counter0`	循环的当前迭代（0索引）
`forloop.revcounter`	循环结束的迭代次数（1索引）
`forloop.revcounter0`	循环结束的迭代次数（0索引）
`forloop.first`	如果这是第一次通过循环，则为真
`forloop.last`	如果这是最后一次循环，则为真
`forloop.parentloop`	对于嵌套循环，这是围绕当前循环的循环

for ... empty

`for` 标签带有一个可选的 `{% empty %}` 从句，以便在给出的组是空的或者没有被找到时，可以有所操作。

```
<ul>
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
{% empty %}
    <li>Sorry, no athletes in this list.</li>
{% endfor %}
</ul>
```

它和下面的例子作用相等，但是更简洁、更清晰甚至可能运行起来更快：

```
<ul>
  {% if athlete_list %}
    {% for athlete in athlete_list %}
      <li>{{ athlete.name }}</li>
    {% endfor %}
  {% else %}
    <li>Sorry, no athletes in this list.</li>
  {% endif %}
</ul>
```

11. if

- `{% if %}` 会对一个变量求值，如果它的值是“True”（存在、不为空、且不是boolean类型的false值），这个内容块会输出：

```
{% if athlete_list %}
    Number of athletes: {{ athlete_list|length }}
{% elif athlete_in_locker_room_list %}
    Athletes should be out of the locker room soon!
{% else %}
    No athletes.
{% endif %}
```

- 上述例子中，如果 `athlete_list` 不为空，就会通过使用 `{{ athlete_list|length }}` 过滤器展示出athletes的数量。
- `if` 标签之后可以带有一个或者多个 `{% elif %}` 从句，也可以带有一个 `{% else %}` 从句以便在之前的所有条件不成立的情况下完成执行。 这些从句都是可选的。

布尔运算符

`if` 标签可以使用 `not`， `and` 或 `or` 来测试多个变量或取消给定变量:

```
{% if athlete_list and coach_list %}
    Both athletes and coaches are available.
{% endif %}

{% if not athlete_list %}
    There are no athletes.
{% endif %}

{% if athlete_list or coach_list %}
    There are some athletes or some coaches.
{% endif %}

{% if not athlete_list or coach_list %}
    There are no athletes or there are some coaches.
{% endif %}

{% if athlete_list and not coach_list %}
    There are some athletes and absolutely no coaches.
{% endif %}
```

允许同时使用 `and` 和 `or` 子句， `and` 的优先级高于 `or` ：

```
{% if athlete_list and coach_list or cheerleader_list %}
```

解释如下:

```
if (athlete_list and coach_list) or cheerleader_list
```

在 `if` 标记中使用实际括号是无效的语法。 如果您需要它们指示优先级，则应使用嵌套的 `if` 标记。

- `if` tags may also use the operators `==`, `!=`, `<`, `>`, `<=`, `>=`, `in`, `not in`, `is`, and `is not` which work as follows:

`==` operator

相等。 例如:

```
{% if somevar == "x" %}
  This appears if variable somevar equals the string "x"
{% endif %}
```

`!=` operator

不相等。 例如:

```
{% if somevar != "x" %}
  This appears if variable somevar does not equal the string "x",
  or if somevar is not found in the context
{% endif %}
```

`<` operator

小于。 例如:

```
{% if somevar < 100 %}
  This appears if variable somevar is less than 100.
{% endif %}
```

`>` operator

大于。 例如:

```
{% if somevar > 0 %}
  This appears if variable somevar is greater than 0.
{% endif %}
```

`<=` operator

小于或等于。 例如:

```
{% if somevar <= 100 %}
  This appears if variable somevar is less than 100 or equal to 100.
{% endif %}
```

`>=` operator

大于或等于。 例如：

```
{% if somevar >= 1 %}
  This appears if variable somevar is greater than 1 or equal to 1.
{% endif %}
```

`in` 操作符

包含在内。 许多Python容器支持此运算符，以测试给定值是否在容器中。 以下是 中的 `x` 的一些示例将被解释：

```
{% if "bc" in "abcdef" %}
  This appears since "bc" is a substring of "abcdef"
{% endif %}

{% if "hello" in greetings %}
  If greetings is a list or set, one element of which is the string
  "hello", this will appear.
{% endif %}

{% if user in users %}
  If users is a QuerySet, this will appear if user is an
  instance that belongs to the QuerySet.
{% endif %}
```

`not in` 操作符

不包含在内。 这是 `in` 运算符的否定操作。

`is` operator

Django中的新功能1.10。

对象身份。 测试两个值是否相同。 例如：

```
{% if somevar is True %}
  This appears if and only if somevar is True.
{% endif %}

{% if somevar is None %}
  This appears if somevar is None, or if somevar is not found in the conte
xt.
{% endif %}
```

`is not` 操作符

Django中的新功能1.10。

否定对象身份 测试两个值是否不一样。 这是 `is` 运算符的否定。 例如：

```
{% if somevar is not True %}
  This appears if somevar is not True, or if somevar is not found in the
  context.
{% endif %}

{% if somevar is not None %}
  This appears if and only if somevar is not None.
{% endif %}
```

过滤器

你也可以在 `if` 表达式中使用过滤器。 像这样：

```
{% if messages|length >= 100 %}
   You have lots of messages today!
{% endif %}
```

复合表达式

所有上述操作符可以组合以形成复杂表达式。 对于这样的表达式，重要的是要知道在表达式求值时如何对运算符进行分组 - 即优先级规则。 操作符的优先级从低至高如下：

- `or`
- `and`
- `not`
- `in`
- `==`， `!=`， `<`， `>`， `<=` ， `>=`

（这完全依据Python）。 所以，例如，下面的复杂 `if` 标签：

```
{% if a == b or c == d and e %}
```

...将被解释为：

```
(a == b) or ((c == d) and e)
```

如果你想要不同的优先级，那么你需要使用嵌套的 `if` 标签。 有时，为了清楚起见，更好的是为了那些不知道优先规则的人。

比较运算符不能像Python或数学符号中那样“链接”。 例如，不能使用：

```
{% if a > b > c %}  (WRONG)
```

你应该使用：

```
{% if a > b and b > c %}
```

12. ifchanged

- 检查一个值是否在上一次的迭代中改变。

  `{% ifchanged %}` 块标签用在循环里。 它可能有两个用处：

  检查它已经渲染过的内容中的先前状态。并且只会显示发生改变的内容。 例如， 以下的代码是

  输出days的列表项，不过它只会输出被修改过月份的项:

```
<h1>Archive for {{ year }}</h1>

{% for date in days %}
    {% ifchanged %}<h3>{{ date|date:"F" }}</h3>{% endifchanged %}
    <a href="{{ date|date:"M/d"|lower }}/">{{ date|date:"j" }}</a>
{% endfor %}
```

- 如果标签内被给予多个值时,则会比较每一个值是否与上一次不同。 例如，以下显示每次更改时

的日期，如果小时或日期已更改，则显示小时：

```
{% for date in days %}
    {% ifchanged date.date %} {{ date.date }} {% endifchanged %}
    {% ifchanged date.hour date.date %}
        {{ date.hour }}
    {% endifchanged %}
{% endfor %}
```

- `ifchanged` 标记也可以采用可选的 `{% else %}` 将显示如果值没有改变：

```
{% for match in matches %}
    <div style="background-color:
        {% ifchanged match.ballot_id %}
            {% cycle "red" "blue" %}
        {% else %}
            gray
        {% endifchanged %}
    ">{{ match }}</div>
{% endfor %}
```

13. include

- 加载模板并以标签内的参数渲染。 这是一种可以引入别的模板的方法。

  模板名可以是变量或者是硬编码的字符串，可以用单引号也可以是双引号.

  下面这个示例包括模板 `"foo/bar.html"` 的内容：

```
{% include "foo/bar.html" %}
```

- 通常模板名称是相对于模板加载器的根目录。 字符串参数也可以是以 `./` 或 `../` 开头的相对路径，如 `extends` 标签中所述。

  Django中的新功能1.10：添加了使用相对路径的能力。

  此示例包括其名称包含在变量 `template_name` 中的模板的内容：

```
{% include template_name %}
```

- 变量也可以是任何实现了 `render()` 方法接口的对象，这个对象要可以接收上下文（context）。 这就允许你在context中引用一个已经被编译过的 `Template` 。

  被包含的模板在包含它的模板的上下文中渲染。

  下面这个示例生成输出 `“Hello, John!”`：

  上下文：变量 `greeting` 设置为 `"John"` ， 变量 `person` 设置为 `"Hello"` 。

  模板：

```
{% include "name_snippet.html" %}
```

name_snippet.html模板：

```
{{ greeting }}, {{ person|default:"friend" }}!
```

- 你可以使用关键字参数将额外的上下文传递到模板：

```
{% include "name_snippet.html" with person="Jane" greeting="Hello" %}
```

- 如果要仅使用提供的变量（或根本不使用变量）来渲染上下文，请使用 `only` 选项。 所包含的模板没有其他变量可用：

```
{% include "name_snippet.html" with greeting="Hi" only %}
```

- 如果包含的模板在渲染时导致异常（包括缺少或具有语法错误），行为会因 `template engine's` 而异。 `debug` 选项（如果未设置，此选项默认为 `DEBUG` 的值）。 当调试模式打开时，将出现 `TemplateDoesNotExist` 或 `TemplateSyntaxError` 之类的异常。当调试模式关闭时， `{% 包括 %}` 向 `django.template` 记录器，除了在渲染所包含的模板并返回一个空字符串时发生的异常。
- 自1.11版以来已弃用 渲染 `{% 包含 %}` 模板标记的沉默异常已弃用。 在Django 2.1中，将会提出异常。
- 注

  `include` 标签应该被理解为是一种”将子模版渲染并嵌入HTML中”的变种方法,而不是认为是”解析子模版并在被父模版包含的情况下展现其被父模版定义的内容”.这意味着在不同的被包含的子模版之间并不共享父模版的状态,每一个子包含都是完全独立的渲染过程.

  Block模块在被包含 *之前* 就已经被执行. 这意味着模版在被包含之前就已经从另一个block扩展并 *已经被执行并完成渲染* - 没有block模块会被include引入并执行,即使父模版中的扩展模版.

14. load

- 加载自定义模板标签集。

  举个例子, 下面这模板将会从 `package` 包中载入所有 `otherlibrary` 和 `somelibrary` 中已经注册的标签和过滤器:

```
{% load somelibrary package.otherlibrary %}
```

- 还可以使用 `from` 参数从库中选择性加载单个过滤器或标记。 在下面这个示例中，名为 `somelibrary` 和 `bar` 的模板标签/过滤器将从 `foo` 加载：

```
{% load foo bar from somelibrary %}
```

15. now

- 显示最近的日期或事件,可以通过给定的字符串格式显示。 此类字符串可以包含格式说明符字符，如 `date` 过滤器部分中所述。

  例如：

```
It is {% now "jS F Y H:i" %}
```

- 注意！，如果你想要使用“raw”值，你能够反斜杠转义一个格式化字符串。 在这个例子中，“o”和“f”都是反斜杠转义，因为如果不这样，会分别显示年和时间：

```
It is the {% now "jS \o\f F" %}
```

这将显示为“这是9月4日”。

- 注

传递的格式也可以是预定义的 `DATE_FORMAT` ， `DATETIME_FORMAT` ， `SHORT_DATE_FORMAT` 或 `SHORT_DATETIME_FORMAT` 之一。 预定义的格式可能会因当前语言环境和Format localization的启用而有所不同，

例如：

```
It is {% now "SHORT_DATETIME_FORMAT" %}
```

- 也可以使用语法 `{% now “Y” as current_year %}` 将输出（作为字符串）存储在变量中。 This is useful if you want to use `{% now %}` inside a template tag like `blocktrans` for example:

```
{% now "Y" as current_year %}
{% blocktrans %}Copyright {{ current_year }}{% endblocktrans %}
```

16. regroup

- 用相似对象间共有的属性重组列表.
- This complex tag is best illustrated by way of an example: say that `cities` is a list of cities represented by dictionaries containing `"name"` , `"population"` , and `"country"` keys:

```
cities = [
    {'name': 'Mumbai', 'population': '19,000,000', 'country': 'India'},
    {'name': 'Calcutta', 'population': '15,000,000', 'country': 'India'},
    {'name': 'New York', 'population': '20,000,000', 'country': 'USA'},
    {'name': 'Chicago', 'population': '7,000,000', 'country': 'USA'},
    {'name': 'Tokyo', 'population': '33,000,000', 'country': 'Japan'},
]
```

并且想显示按国家/地区排序的分层列表，如下所示：

- 印度
  - 孟买：19,000,000
  - 加尔各答：15,000,000
- 美国
  - 纽约：20,000,000
  - 芝加哥：7,000,000
- 日本
  - 东京：33,000,000

你可以使用 {% regroup %} 标签来给每个国家的城市分组。 以下模板代码片段将实现这一点：

```
{% regroup cities by country as country_list %}

<ul>
{% for country in country_list %}
    <li>{{ country.grouper }}
    <ul>
        {% for city in country.list %}
          <li>{{ city.name }}: {{ city.population }}</li>
        {% endfor %}
    </ul>
    </li>
{% endfor %}
</ul>
```

让我们来看看这个例子。 {% regroup %} 有三个参数： 你想要重组的列表, 被分组的属性, 还有结果列表的名字. 在这里，我们通过 country_list 属性重新分组 country 列表，并调用结果 cities 。

{% regroup %} 产生一个清单（在本例中为 country_list 的**组对象**。 组对象是具有两个字段的 namedtuple() 的实例：

- grouper - 按分组的项目（例如，字符串“India”或“Japan”）。
- list - 此群组中所有项目的列表（例如，所有城市的列表，其中country =’India’）。

在Django更改1.11：组对象已从字典更改为 namedtuple() 。

Because {% regroup %} produces namedtuple() objects, you can also write the previous example as:

```
{% regroup cities by country as country_list %}

<ul>
{% for country, local_cities in country_list %}
    <li>{{ country }}
    <ul>
        {% for city in local_cities %}
          <li>{{ city.name }}: {{ city.population }}</li>
        {% endfor %}
    </ul>
    </li>
{% endfor %}
</ul>
```

请注意， {% regroup %} 不会对其输入进行排序！ 我们的例子依赖于事实： cities 列表首先由 country 排序。 如果 country 列表不通过 cities 对其成员进行排序，则重新分组将天真显示单个国家/地区的多个组。 例如，假设 cities 列表已设置为此（请注意，国家/地区未分组在一起）：

```
cities = [
    {'name': 'Mumbai', 'population': '19,000,000', 'country': 'India'},
    {'name': 'New York', 'population': '20,000,000', 'country': 'USA'},
    {'name': 'Calcutta', 'population': '15,000,000', 'country': 'India'},
    {'name': 'Chicago', 'population': '7,000,000', 'country': 'USA'},
    {'name': 'Tokyo', 'population': '33,000,000', 'country': 'Japan'},
]
```

对于 `cities` 的输入，示例 `{% regroup %}` 以上将导致以下输出：

- 印度
  - 孟买：19,000,000
- 美国
  - 纽约：20,000,000
- 印度
  - 加尔各答：15,000,000
- 美国
  - 芝加哥：7,000,000
- 日本
  - 东京：33,000,000

这个问题的最简单的解决方案是确保在你的视图代码中，数据是根据你想要显示的顺序排序。

另一个解决方案是使用 `dictsort` 过滤器对模板中的数据进行排序，如果您的数据在字典列表中：

```
{% regroup cities|dictsort:"country" by country as country_list %}
```

分组其他属性

- 一个有效的模版查找是一个regroup标签的合法的分组属性。包括方法，属性，字典健和列表项。 例如，如果“country”字段是具有属性“description”的类的外键，则可以使用：

```
{% regroup cities by country.description as country_list %}
```

- 或者，如果 `choices` 是具有 `choices` 的字段，则它将具有作为属性的 `get_FOO_display()` 方法，显示字符串而不是 `country` 键：

```
{% regroup cities by get_country_display as country_list %}
{{ country.grouper }}`现在会显示`choices
```

17. resetcycle

Django中的新功能1.11。

- 重置先前的循环，以便在下一次遇到时从其第一个项目重新启动。 没有参数， `{% resetcycle %}` 会重置模板中定义的最后一个 `{% cycle %}`

用法示例：

```
{% for coach in coach_list %}
    <h1>{{ coach.name }}</h1>
    {% for athlete in coach.athlete_set.all %}
        <p class="{% cycle 'odd' 'even' %}">{{ athlete.name }}</p>
    {% endfor %}
    {% resetcycle %}
{% endfor %}
```

这个示例将返回下面的HTML：

```
<h1>José Mourinho</h1>
<p class="odd">Thibaut Courtois</p>
<p class="even">John Terry</p>
<p class="odd">Eden Hazard</p>

<h1>Carlo Ancelotti</h1>
<p class="odd">Manuel Neuer</p>
<p class="even">Thomas Müller</p>
```

注意第一个块以 `class="odd"` 结束，新的以 `class="odd"` 开头。 没有 `{% resetcycle %}` 标签，第二个块将以 `class="even"`

- 还可以重置命名循环标签：

```
{% for item in list %}
    <p class="{% cycle 'odd' 'even' as stripe %} {% cycle 'major' 'minor'
 'minor' 'minor' 'minor' as tick %}">
        {{ item.data }}
    </p>
    {% ifchanged item.category %}
        <h1>{{ item.category }}</h1>
        {% if not forloop.first %}{% resetcycle tick %}{% endif %}
    {% endifchanged %}
{% endfor %}
```

在这个例子中，我们有交替的奇数/偶数行和第五行的“主要”行。 当类别更改时，只有五行周期被重置。

18. spaceless

- 删除HTML标签之间的空白格. 包括制表符和换行.

用法示例：

```
{% spaceless %}
    <p>
        <a href="foo/">Foo</a>
    </p>
{% endspaceless %}
```

这个示例将返回下面的HTML：

```
<p><a href="foo/">Foo</a></p>
```

- 仅删除 *tags* 之间的空格 – 而不是标签和文本之间的。 在此示例中， `Hello` 周围的空格不会被删除：

```
{% spaceless %}
    <strong>
        Hello
    </strong>
{% endspaceless %}
```

19. templatetag

- 输出用于构成模板标记的语法字符之一。
- 由于模板系统没有“转义”的概念，为了显示模板标签中使用的一个位，必须使用 `{% templatetag %}` 标记。
- 参数指定要输出哪个模板位：

论据	输出
openblock	{%
closeblock	%}
openvariable	{{
closevariable	}}
openbrace	{
closebrace	}
opencomment	{#
closecomment	#}

例如：

```
{% templatetag openblock %} url 'entry_list' {% templatetag closeblock %}
```

20. url

- 返回与给定视图和可选参数匹配的绝对路径引用（不带域名的URL）。 在解析后返回的结果路径字符串中，每个特殊字符将使用 `iri_to_uri()` 编码。
- 这是一种不违反DRY原则的输出链接的方式，它可以避免在模板中硬编码链接路径。

```
{% url 'some-url-name' v1 v2 %}
```

- 第一个参数是 `url()  name` 。 它可以是一个被引号引起来的字符串或者其他的上下文变量. 其他参数是可选的并且应该以空格隔开，这些值会在URL中以参数的形式传递. 上面的例子展示了如何传递位置参数. 当然你也可以使用关键字参数.

```
{% url 'some-url-name' arg1=v1 arg2=v2 %}
```

- 不要把位置参数和关键字参数混在一起使用。 URLconf所需的所有参数都应该存在。

  例如，假设您有一个视图 `app_views.py` ，其URLconf接受客户端ID（此处 `client()` 是视图文件 `app_views.client` ）。 URLconf行可能如下所示：

```
('^client/([0-9]+)/$', app_views.client, name='app-views-client')
```

如果你的应用中的URLconf 已经被包含到项目 URLconf 中，比如下面这样

```
('^clients/', include('project_name.app_name.urls'))
```

然后，在模板中，您可以创建一个此视图的链接，如下所示：

```
{% url 'app-views-client' client.id %}
```

模板标签会输出如下的字符串 `/clients/client/123/` .

请注意，如果您要反查的网址不存在，您会收到 `NoReverseMatch` 异常，这会导致您的网站显示错误网页。

如果希望在不显示网址的情况下检索网址，则可以使用略有不同的调用：

```
{% url 'some-url-name' arg arg2 as the_url %}

<a href="{{ the_url }}">I'm linking to {{ the_url }}</a>
```

- `as var` 语法创建的变量的范围是 `{% 块 %}` 其中 `{% url %}` 标签出现。

  此标签 `{% url ... as VAR %}` 语法将不导致错误，如果视图丢失。 实际上，您将使用此链接来链接到可选的视图：

```
{% url 'some-url-name' as the_url %}
{% if the_url %}
  <a href="{{ the_url }}">Link to optional stuff</a>
{% endif %}
```

- 如果要检索名称空间网址，请指定完全限定名称：

```
{% url 'myapp:view-name' %}
```

这将遵循正常的namespaced URL resolution strategy，包括使用上下文对当前应用程序提供的任何提示。

- 警告

不要忘记在 `url()` `name` 之间放置引号，否则该值将被解释为上下文变量！

21. verbatim

- 停止模版引擎在该标签中的渲染/
- 常见的用法是允许与Django语法冲突的JavaScript模板图层。 像这样：

```
{% verbatim %}
    {{if dying}}Still alive.{{/if}}
{% endverbatim %}
```

- You can also designate a specific closing tag, allowing the use of `{% endverbatim %}` as part of the unrendered contents:

```
{% verbatim myblock %}
    Avoid template rendering via the {% verbatim %}{% endverbatim %} bloc
k.
{% endverbatim myblock %}
```

22. widthratio

- 为了创建条形图等，此标签计算给定值与最大值的比率，然后将该比率应用于常量。

  像这样：

```
<img src="bar.png" alt="Bar"
     height="10" width="{% widthratio this_value max_value max_width %}" /
>
```

- 如果 `max_width` 是175， `max_value` 是200，并且 `this_value` 是100，则上述示例中的图像将是88像素宽（因为175 / 200 = .875； .875 * 100 = 87.5，上舍入为88）。
- 在某些情况下，您可能想要捕获变量中的 `widthratio` 的结果。 它可以是有用的，例如，在 `blocktrans` 像这样：

```
{% widthratio this_value max_value max_width as width %}
{% blocktrans %}The width is: {{ width }}{% endblocktrans %}
```

23. with

- 使用一个简单地名字缓存一个复杂的变量， 当你需要使用一个“昂贵的”方法（比如访问数据库）很多次的时候是非常有用的

  像这样：

```
{% with total=business.employees.count %}
    {{ total }} employee{{ total|pluralize }}
{% endwith %}
```

- 填充变量（以上示例 `total` ）仅适用于 `{% with %} t5>` 和 `{% endwith %}` 标签。

  可以分配多个上下文变量：

```
{% with alpha=1 beta=2 %}
    ...
{% endwith %}
```