模板

如果使用react实现前端页面，其实Django就没有必须使用模板，它其实就是一个后台服务程序，接收请求，响应数据。接口设计就可以是纯粹的Restful风格。

模板的目的就是为了可视化，将数据按照一定布局格式输出，而不是为了数据处理，所以一般不会有复杂的处理逻辑。模板的引入实现了业务逻辑和显示格式的分离，这样，在开发中，就可以分工协作，页面开发完成页面布局设计，后台开发完成数据处理逻辑的实现。

Python的模板引擎默认使用Django template language (DTL)构建

模板配置

在settings.py中，设置模板项目的路径

```
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__))) # 这一句取项目根目录

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

DIRS 列表，定义模板文件的搜索路径顺序
APP_DIRS 是否运行在每个已经安装的应用中查找模板。应用自己目录下有templates目录，例如django/contrib/admin/templates。如果应用需要可分离、可重用，建议把模板放到应用目录下

BASE_DIR 是 项目根目录，os.path.join(BASE_DIR, 'templates')就是在manage.py这一层建立一个目录templates。这个路径就是以后默认找模板的地方。

模板渲染

模板处理2个步骤

1、加载模板
模板是一个文件，需要从磁盘读取并加载。要将模板放置在
2、渲染
模板需要使用内容数据来渲染，生成HTML文件内容

```
from django.template import loader, RequestContext

def index(request:HttpRequest):
    """视图函数：请求进来返回响应"""
    template = loader.get_template('index.html') # 加载器模块搜索模板并加载它
    context = RequestContext(request, {'content':'www.magedu.com'})
    return HttpResponse(template.render(context))
```

render快捷渲染函数

上面2个步骤代码编写繁琐，Django提供了对其的封装——快捷函数render。
render(request, template_name, context=None)
返回HttpResponse对象
template_name 模板名称
context 数据字典
render_to_string()

```
from django.shortcuts import render

def index(request:HttpRequest):
    """视图函数：请求进来返回响应"""
    return render(request, 'index.html',{'content': 'www.magedu.com'})
```

模板页

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>马哥教育Django首页</title>
</head>
<body>
我是模板，数据是{{content}}
</body>
</html>
```

将模板index.html放入到templates目录下。使用浏览器访问首页，可以正常显示

DTL语法

- 变量
- 标签
- 注释

- 过滤器

1 变量

`语法 {{ variable }}`

变量名由字母、数字、下划线、点号组成。

点号使用的时候，例如foo.bar，遵循以下顺序：

1. 字典查找，例如foo["bar"]，把foo当做字典，bar当做key
2. 属性或方法的查找，例如foo.bar，把foo当做对象，bar当做属性或方法
3. 数字索引查找，例如foo[bar]，把foo当做列表一样，使用索引访问

如果变量未能找到，则缺省插入空字符串''
在模板中调用方法，不能加小括号，自然也不能传递参数。
{{ my_dict.keys }} 这样是对的，不能写成{{ my_dict.keys() }}

2 模板标签

if/else 标签
基本语法格式如下：

```
{% if condition %}
     ... display
{% endif %}
```

或者：

```
{% if condition1 %}
   ... display 1
{% elif condition2 %}
   ... display 2
{% else %}
   ... display 3
{% endif %}
```

条件也支持and、or、not
注意，因为这些标签是断开的，所以不能像Python一样使用缩进就可以表示出来，必须有个结束标签，例如endif、endfor。

for 标签
https://docs.djangoproject.com/en/2.0/ref/templates/builtins/#for

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

{%  for person  in  person_list %}
<li>    {{  person.name }}  </li>
{%  endfor  %}
```

变量	说明
forloop.counter	当前循环从1开始的计数
forloop.counter0	当前循环从0开始的计数
forloop.revcounter	从循环的末尾开始倒计数到1
forloop.revcounter0	从循环的末尾开始到计数到0
forloop.first	第一次进入循环
forloop.last	最后一次进入循环
forloop.parentloop	循环嵌套时，内层当前循环的外层循环

给标签增加一个 reversed 使得该列表被反向迭代：

```
{% for athlete in athlete_list reversed %}
...
{% empty %}
... 如果被迭代的列表是空的或者不存在，执行empty
{% endfor %}
```

可以嵌套使用 {% for %} 标签：

```
{% for athlete in athlete_list %}
    <h1>{{ athlete.name }}</h1>
    <ul>
    {% for sport in athlete.sports_played %}
        <li>{{ sport }}</li>
    {% endfor %}
    </ul>
{% endfor %}
```

testfor.html模板

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <title>测试for</title>
</head>
<body>
字典是dict(zip('abced', range(1,6)))
<ul>
{% for k,v in dct.items %}
    <li>{{forloop.counter}} {{k}} {{v}}</li>
{% endfor %}
</ul>
<hr>

<ul>
{% for k,v in dct.items %}
    <li>{{forloop.counter0}} {{k}} {{v}}</li>
{% endfor %}
</ul>
<hr>

<ul>
{% for k,v in dct.items %}
    {{ forloop.first }}
    {{ forloop.last }}
    <li>{{forloop.revcounter0}} {{k}} {{v}}</li>
{% endfor %}
</ul>
<hr>

<ul>
{% for k,v in dct.items %}
    <li>{{forloop.revcounter}} {{k}} {{v}}</li>
{% endfor %}
</ul>
<hr>

</body>
</html>
```

ifequal/ifnotequal 标签
{% ifequal %} 标签比较两个值，当他们相等时，显示在 {% ifequal %} 和 {% endifequal %} 之中所有的值。
下面的例子比较两个模板变量 user 和 currentuser :

```
{% ifequal user currentuser %}
    <h1>Welcome!</h1>
{% endifequal %}
```

和 {% if %} 类似， {% ifequal %} 支持可选的 {% else%} 标签：

```
{% ifequal section 'sitenews' %}
    <h1>Site News</h1>
{% else %}
    <h1>No News Here</h1>
{% endifequal %}
```

其他标签

csrf_token 用于跨站请求伪造保护，防止跨站攻击的。
{% csrf_token %}

3 注释标签

单行注释 {# #}。
多行注释 {% comment %} ... {% endcomment %}.

```
{# 这是一个注释 #}

{% comment %}
这是多行注释
{% endcomment %}.
```

4 过滤器

模板过滤器可以在变量被显示前修改它。

语法 {{ 变量|过滤器 }}
过滤器使用管道字符 | ，例如{{ name|lower }}，{{ name }} 变量被过滤器 lower 处理后，文档大写转换文本为小写。

过滤管道可以被**套接** ，一个过滤器管道的输出又可以作为下一个管道的输入，例如{{ my_list|first|upper }}，将列表第一个元素并将其转化为大写。

过滤器传参
有些过滤器可以传递参数，过滤器的参数跟随冒号之后并且总是以双引号包含。
例如：{{ bio|truncatewords:"30" }}，截取显示变量 bio 的前30个词。
{{ my_list|join:"," }}，将my_list的所有元素使用,逗号连接起来

其他过滤器

过滤器	说明	举例
first	取列表第一个元素	
last	取列表最后元素	
yesno	变量可以是True、False、None，yesno的参数给定逗号分隔的三个值，返回3个值中的一个。 True对应第一个 False对应第二个 None对应第三个 如果参数只有2个，None等效False处理	{{ value | yesno:"yeah,no,maybe" }}
add	加法。参数是负数就是减法	数字加{{ value | add:"100"}} 列表合并{{mylist | add:newlist}}
divisibleby	能否被整除	{{ value | divisibleby:"3" }}能被3整除返回True
addslashes	在反斜杠、单引号或者双引号前面加上反斜杠	{{ value | addslashes }}
length	返回变量的长度	{% if my_list | length > 1 %}
default	变量等价False则使用缺省值	{{ value | default:"nothing" }}
default_if_none	变量为None使用缺省值	{{ value | default_if_none:"nothing" }}

date : 按指定的格式字符串参数格式化 date 或者 datetime 对象，实例：
{{ pub_date|date:"n j, Y" }}
n 1~12 月
j 1~31 日
Y 2000 年
时间的格式字符查看 https://docs.djangoproject.com/en/2.0/ref/templates/builtins/#date

练习：要求模板中列表输出多行数据，要求奇偶行颜色不同

```
<ul>
{% for k,v in dct.items %}
    <li style='color:{{forloop.revcounter0|divisibleby:"2"|yesno:"red,blue"}}'>
{{forloop.revcounter}} {{k}} {{v | add:"100"}}</li>
{% endfor %}
</ul>
```