[](http://python.jobbole.com/)

* [**首页**](http://python.jobbole.com/)
* [**所有文章**](http://python.jobbole.com/all-posts/)
* [**观点与动态**](http://python.jobbole.com/category/news/)
* [**基础知识**](http://python.jobbole.com/category/basic/)
* [**系列教程**](http://python.jobbole.com/category/guide/)
* [**实践项目**](http://python.jobbole.com/category/project/)
* [**工具与框架应用**](http://python.jobbole.com/category/tools/)
* [**工具资源**](http://hao.jobbole.com/?catid=144)



[伯乐在线](http://www.jobbole.com/) > [Python - 伯乐在线](http://python.jobbole.com/) > [所有文章](http://python.jobbole.com/all-posts/) > [工具与框架](http://python.jobbole.com/category/tools/) > Flask、Django、Pyramid三个框架的对比

# Flask、Django、Pyramid三个框架的对比

2015/05/09 · [工具与框架](http://python.jobbole.com/category/tools/), [框架](http://python.jobbole.com/category/framework/) · [Django](http://python.jobbole.com/tag/django/), [Flask](http://python.jobbole.com/tag/flask/), [Pyramid](http://python.jobbole.com/tag/pyramid/)

分享到：**13**

本文由 [伯乐在线](http://python.jobbole.com/) - [贱圣OMG](http://www.jobbole.com/members/fwg1989) 翻译，[Daetalus](http://www.jobbole.com/members/daetalus) 校稿。未经许可，禁止转载！  
英文出处：[www.airpair.com](https://www.airpair.com/python/posts/django-flask-pyramid)。欢迎加入[翻译组](http://group.jobbole.com/category/feedback/trans-team/)。

**1.简介**

在python web框架的世界里充满了选择。有Django，Flask，Pyramid，Tornado，Bottle，Diesel，Pecan，Falcon等等的来吸引开发者的注意。作为一个开发者，你想要从中选择一个框架来帮你完成项目，并且能继续做大事情。我们将关注Flask、Pyramid和Django。它们是微框架和商业级web服务的典范。

为了让你在这三个中做选择的时候能够更容易，我们会用每个框架来构建相同的应用，然后比较各自的代码，突出它们的优势和弱点。如果你想要代码，可以直接跳到 框架实战 章节或者在[Github](https://github.com/ryansb/wut4lunch_demos)上下载代码。

Flask是一个微框架，主要面向需求简单的小应用。Pyramid和Django都是面向大的应用，但是在扩展性和灵活性上走了不同的路。Pyramid关注灵活性，让开发者选择合适的工具来开发项目。这意味着开发者可以选择数据库，URL结构，模板风格等等。Django的目标是提供web应用开发的一站式解决方案，所以相应的模块也就比较多。

Django包含了一个ORM模块，而Pyramid和Flask是让开发者来选择如何存储数据。针对非Django框架的最流行的ORM目前是SQLAlchemy，也有很多其他的选择，比如DynamoDB和MongoDB，亦或是像LevelDB和SQLite这样的简单本地持久化。Pyramid被设计可以使用任何持久层，甚至是还没做好的。

## 2  关于框架

Django的一站式解决的思路能让开发者不用在开发之前就在选择应用的基础设施上花费大量时间。Django有模板，表单，路由，认证，基本的数据库管理等等内建功能。与之相反，Pyramid包含路由和验证，但是模板和数据库管理需要第三方库。

用Flask和Pyramid来构建应用之前，选择组件的时候会给开发者带来更多的灵活性 ，可能有的应用场景不适合使用一个标准的ORM，或者需要与不同的工作流和模板系统交互。

Flask，这三个框架李最年轻的一个，创始与2010年年中。Pyramid框架来源于Pylons项目，在2010年末更名为Pyramid，它最早发布与2005年。Django发布于2006年，就在Pylons项目之后。Pyramid和Django是非常成熟的框架，积累了大量的插件和扩展来满足不同需要。

尽管Flask的历史较短，但它能够从以前的框架学到一些东西并且将它的目标设定在了小型项目上。它在一些仅有一两个功能的小型项目上得到了大量应用。比如httpbin这样的项目，简单但非常强大，是一个帮助debug和测试HTTP的库。

## 3 社区

Django的社区是最活跃的，在StackOverflow上有80000个相关问题和大量的博客和强大的用户。Flask和Pyramid的社区就没有这么大了，但是它们的社区在邮件列表和IRC里还是挺活跃的。在StackOverflow上只有5000个相关问题，Flask比Django的关注度小15倍。在Github上，它们的stars数相近，Django有11300个，Flask有10900个。

这三个框架都是处于BSD衍生的开源许可证书之下。Flask和Django的证书都是3条款BSD，而Pyramid的是RPL，是4条款BSD证书的衍生版。

## 4 入门引导

Django和Pyramid都有内建的引导工具。Flask没有，因为Flask的主要受众不是要构建大型MVC应用的。

### 4.1 Flask

Flask的Hello World应用的代码是最简单的，只用在一个Python文件里码7行代码就够了。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | # from http://flask.pocoo.org/ tutorial  from flask import Flask  app = Flask(\_\_name\_\_)    @app.route("/") # take note of this decorator syntax, it's a common pattern  def hello():      return "Hello World!"    if \_\_name\_\_ == "\_\_main\_\_":      app.run() |

这就是为什么Flask没有引导工具：因为它根本不需要。从上面的Hello World应用的特点来看，一个没什么Python web开发经验的人就可以很快的上手开始撸代码。

对于需要把组件分离开的项目，Flask有[blueprints](http://flask.pocoo.org/docs/0.10/blueprints/)。例如，你可以这样构建你的应用，将与用户有关的功能放在user.py里，把与销售相关的功能放在ecommerce.py里，然后在site.py里引用并添加到你的应用里。我们暂时不会体验这个功能了，它超出了我们的实例应用的需要。

### 4.2 Pyramid

Pyramid的引导工具叫pcreate，它是Pyramid的一部分。之前有一个Paste的工具，不过后来被Pyramid指定的工具链替代了。

Python



|  |  |
| --- | --- |
| 1 | $ pcreate -s starter hello\_pyramid # Just make a Pyramid project |

Pyramid希望能够做比Flask更大和复杂的应用。因此，它的引导工具会创建一个更大的项目框架。它里面包含一个配置文件，一个例子模板，还有文件能打包你的应用并上传到Python Package Index（PYPI）。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | hello\_pyramid  ├── CHANGES.txt  ├── development.ini  ├── MANIFEST.in  ├── production.ini  ├── hello\_pyramid  │ ├── \_\_init\_\_.py  │ ├── static  │ │ ├── pyramid-16x16.png  │ │ ├── pyramid.png  │ │ ├── theme.css  │ │ └── theme.min.css  │ ├── templates  │ │ └── mytemplate.pt  │ ├── tests.py  │ └── views.py  ├── README.txt  └── setup.py |

跟别的框架相比，Pyramid的引导工具特别的灵活。它没有被限制在一个默认应用里;pcreate可以使用任何数量的项目模板。包括我们在上面使用starter模板创建出来的，包含SQLAlchemy和ZODB支撑的项目。在PyPi上，可以找到依赖于Google App Engine，jQuery Mobile，Jinja2 templating，modern frontend frameworks等等的模板。

### 4.3 Django

Django也有自己的引导工具，它是django-admin的一部分。

Python



|  |  |
| --- | --- |
| 1  2 | django-admin startproject hello\_django  django-admin startapp howdy # make an application within our project |

我们已经可以看到Django和Pyramid的一些区别。Django把一个项目分成各自独立的应用，而Pyramid和Flask认为一个项目应该是一个包含一些视图和模型的单个应用。也可以在Flask和Pyramid里复制出像Django那样的项目结构，但那不是默认的。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | hello\_django  ├── hello\_django  │ ├── \_\_init\_\_.py  │ ├── settings.py  │ ├── urls.py  │ └── wsgi.py  ├── howdy  │ ├── admin.py  │ ├── \_\_init\_\_.py  │ ├── migrations  │ │ └── \_\_init\_\_.py  │ ├── models.py  │ ├── tests.py  │ └── views.py  └── manage.py |

默认情况下，Django只包含空的模型和模板文件，一个新的用户可以看一点例子代码就开始工作。它也能让开发者选择如何分配Django的应用。

这个引导工具的弱点是没有引导用户去打包他们的应用，新手往往会忽视这一点。如果一个开发者从未打包过应用，他们会发现在第一次部署的时候会有点乱糟糟的。一些有较大社区的项目，如django-oscar是被打包好的，并且能在PyPi上下载，但是在Github上，一些小的项目往往缺乏统一的打包。

## 5 模板

有一个能回复HTTP请求的Python应用是个很好的开始，但是你的用户不会用curl来与你的应用交互。幸运的是，这三个框架都提供了简单的方法能让你向HTML里填充自定义信息，可以让大家领略到你酷炫的前端技术。

模板能让你直接向页面中嵌入动态信息，而不用发送AJAX请求。这对于用户体验是有利的，你能一次加载整个页面和它的动态数据。这对于移动端网站是非常重要的，能够节省数秒钟时间。

所有的我们使用的模板都依赖上下文提供动态信息，并将其通过模板渲染进HTML。模板的最简单使用场景是很流行的一个例子，给已登录的用户的名字打招呼。也可以使用AJAX来获取动态信息，但需要一个调用来获取用户名信息，这样可能有点费事了，毕竟用模板来实现这么简单。

### 5.1 Django

我们这里的使用场景非常简单，假定我们有一个user对象，它有一个fullname的属性，里面包含一个用户的名字。在Python里，我们会这样把当前用户信息传递给模板：

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | def a\_view(request):      # get the logged in user      # ... do more things      return render\_to\_response(          "view.html",          {"user": cur\_user}      ) |

构成模板上下文非常简单，只需要将一个有Python对象和数据结构的字典传进模板即可。现在，我们需要通过名称将其渲染进页面，以免我们忘了它们是什么。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | &lt;!-- view.html --&gt;  &lt;div class=&quot;top-bar row&quot;&gt;    &lt;div class=&quot;col-md-10&quot;&gt;    &lt;!-- more top bar things go here --&gt;    &lt;/div&gt;    {% if user %}    &lt;div class=&quot;col-md-2 whoami&quot;&gt;      You are logged in as {{ user.fullname }}    &lt;/div&gt;    {% endif %}  &lt;/div&gt; |

首先，我们看到{% if user %}这样的结构。在Django模板里，{% 是用来一些控制语句的比如循环和条件式。if use 的声明是用来针对如果没有user的情况。匿名用户不应该在页面上方看到“you are logged in as”的字样。

在if块里，你可以看到包括名字是被简单的封装在了{{}}里。{{是用来放实际中嵌入模板的值，比如{{ user.fullname }}。

模板的另一个常见用法是展示一组东西，比如一个商业网站的存货页面。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | def browse\_shop(request):      # get items      return render\_to\_response(          "browse.html",          {"inventory": all\_items}      ) |

模板里，我们可以使用相同的 {% 来循环出存货数据里的所有项目，并且将其填充到URL和各自页面。

Python



|  |  |
| --- | --- |
| 1  2  3 | {% for widget in inventory %}      &lt;li&gt;&lt;a href=&quot;/widget/{{ widget.slug }}/&quot;&gt;{{ widget.displayname }}&lt;/a&gt;&lt;/li&gt;  {% endfor %} |

对于大多数普通的模板任务来说，Django能够轻松实现，非常容易上手。

### 5.2 Flask

Flask默认使用一个受Django启发而发展起来的名为Jinja2的模板，但也可以通过配置来使用其他的语言。一个码农可能会将Django和Jinja模板弄混。事实上，所有上面的Django模板的例子在Jinja2里也是好使的。我们就不重复上面的例子了，我们来看看Jinja2比Django模板的一些更有表现力的特点。

Jinja和Django模板都提供一个叫过滤的特性，一个列表可以在被展示前传给一个函数。一个博客如果包含分类的特性，就可以使用过滤来将一个分类下的文章筛选出来。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5 | &lt;!-- Django --&gt;  &lt;div class=&quot;categories&quot;&gt;Categories: {{ post.categories|join:&quot;, &quot; }}&lt;/div&gt;    &lt;!-- now in Jinja --&gt;  &lt;div class=&quot;categories&quot;&gt;Categories: {{ post.categories|join(&quot;, &quot;) }}&lt;/div&gt; |

在Jinja的模板语言里，可以把任何数量的参数传给过滤器，因为Jinja像调用一个Python函数的方式来看待它，用圆括号来封装参数。Django使用冒号来分隔过滤器名和参数，这样就只能传递一个参数了。

Jinja和Django的 for 循环很相似。我们来看看它们的区别。在Jinja2, for-else-endfor 结构让你能对一个列表进行迭代，也能处理列表为空的情况。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | {% for item in inventory %}  &lt;div class=&quot;display-item&quot;&gt;{{ item.render() }}&lt;/div&gt;  {% else %}  &lt;div class=&quot;display-warn&quot;&gt;  &lt;h3&gt;No items found&lt;/h3&gt;  &lt;p&gt;Try another search, maybe?&lt;/p&gt;  &lt;/div&gt;  {% endfor %} |

在Django版本的功能是一样的，只是使用了 for-empty-endfor 这样的结构替换了 for-else-endfor 的结构。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | {% for item in inventory %}  &lt;div class=&quot;display-item&quot;&gt;{{ item.render }}&lt;/div&gt;  {% empty %}  &lt;div class=&quot;display-warn&quot;&gt;  &lt;h3&gt;No items found&lt;/h3&gt;  &lt;p&gt;Try another search, maybe?&lt;/p&gt;  &lt;/div&gt;  {% endfor %} |

除了上面的语法差别，Jinja2提供更多的执行环境的控制和高级特性。例如，它可以关闭潜在的危险特性来运行不受信任的模板，或者提前编译模板以确保它们有效。

### 5.3 Pyramid

跟Flask类似，Pyramid支持很多模板语言（包括Jinja2和Mako），但它有一个默认的模板。Pyramid使用Chameleon，一个ZPT（the Zope Page Template）语言的实现。我们来看一个例子，把用户的名字添加到网站顶端。相应的Python代码有点类似但更加明确，不用调用render\_template函数。

Python



|  |  |
| --- | --- |
| 1  2  3  4 | @view\_config(renderer='templates/home.pt')  def my\_view(request):      # do stuff...      return {'user': user} |

但是我们的模板看起来就差别挺大了。ZPT是一个基于XML的模板标准，所以我们使用XSLT类似的声明来管理数据。



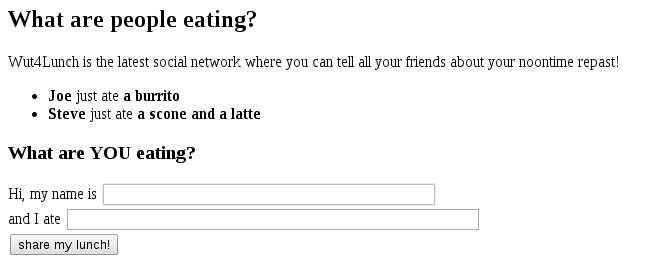
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <div class="top-bar row">    <div class="col-md-10">    <!-- more top bar things go here -->    </div>    <div tal:condition="user"         tal:content="string:You are logged in as ${user.fullname}"         class="col-md-2 whoami">    </div>  </div> |

Chameleon 有三种不同的模板动作命名空间。TAL（template attribute language）提供基本的条件语句，基本的字符串格式化，根据标记过滤内容。上面的例子只使用了TAL来完成相应工作。对于更加高级的工作，就需要TALES和METAL了。TALES（Template Attribute Language Expression Syntax）提供一些高级的字符串格式化，评估Python表达式，和引入表达式和模板。

METAL（Macro Expansion Template Attribute Language）是Chameleon模板里最强大（和最复杂）的部分。Macros是可扩展的，可以被定义得像槽一样，在macro被调用的时候填充上。

## 6. 框架实战

对每个框架，我们来看看做一个叫wut4lunch的应用，这是一个社交网络，可以告诉整个英特网这你晚饭吃了什么。就用这个想法启动，是个游戏规则改变者。这个应用会是一个简单的接口，能让用户PO出他们午饭吃了什么，还可以看到别人吃了什么。首页看起来会是这个样子：



### 6.1 用Flask做的应用

这个最短的实现，仅需34行Python代码和一个22行的Jinja模板。首先，我们有一些例行公事的任务，比如初始化我们的应用和引入ORM。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | from flask import Flask    # For this example we'll use SQLAlchemy, a popular ORM that supports a  # variety of backends including SQLite, MySQL, and PostgreSQL  from flask.ext.sqlalchemy import SQLAlchemy    app = Flask(\_\_name\_\_)  # We'll just use SQLite here so we don't need an external database  app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///test.db'    db = SQLAlchemy(app) |

首先，我们来看我们的模型，跟其他两个例子里的会差不多。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5 | class Lunch(db.Model):      """A single lunch"""      id = db.Column(db.Integer, primary\_key=True)      submitter = db.Column(db.String(63))      food = db.Column(db.String(255)) |

Wow，就是这么简单。最难的地方是找到合适的SQLAlchemy 数据类型并为数据库的里的字段选择一个合适的长度。使用我们的模型是相当简单的，要感谢SQLAlchemy 的查询语法，我们在接下来会看到。

建立我们的提交表单灰常简单。引入Flask-WTForms 并且弄好字段类型，你可以看到表单看起来就像我们的模型。主要的区别是新的提交按钮和关于食品的提示和提交者名字的字段。

SECRET\_KEY 字段在应用的配置里是被WTForms使用的，用来创建CSRF token。它也被 isdangerous 使用，用来创建cookie和其他数据。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | from flask.ext.wtf import Form  from wtforms.fields import StringField, SubmitField    app.config['SECRET\_KEY'] = 'please, tell nobody'    class LunchForm(Form):      submitter = StringField(u'Hi, my name is')      food = StringField(u'and I ate')      # submit button will read "share my lunch!"      submit = SubmitField(u'share my lunch!') |

让表单在浏览器里显示出来意味着模板里需要有它。我们会在下面做这件事。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | from flask import render\_template    @app.route("/")  def root():      lunches = Lunch.query.all()      form = LunchForm()      return render\_template('index.html', form=form, lunches=lunches) |

好的，发生了什么？我们有了用Lunch.query.all() 查询出来的一些已经PO出来的午餐，实例化一个表单，让用户PO出他们自己的美食冒险。为了简洁，变量用相同名称传递给模板，但这不是必须的。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | &lt;html&gt;  &lt;title&gt;Wut 4 Lunch&lt;/title&gt;  &lt;b&gt;What are people eating?&lt;/b&gt;    &lt;p&gt;Wut4Lunch is the latest social network where you can tell all your friends  about your noontime repast!&lt;/p&gt; |

这是真正的模板了，我们循环出所有展示和吃掉的午餐，并将它们展示在<ul>标签里。这与之前看到的例子差不多。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | &lt;ul&gt;  {% for lunch in lunches %}  &lt;li&gt;&lt;strong&gt;{{ lunch.submitter|safe }}&lt;/strong&gt; just ate &lt;strong&gt;{{ lunch.food|safe }}&lt;/strong&gt;  {% else %}  &lt;li&gt;&lt;em&gt;Nobody has eaten lunch, you must all be starving!&lt;/em&gt;&lt;/li&gt;  {% endfor %}  &lt;/ul&gt;    &lt;b&gt;What are YOU eating?&lt;/b&gt;    &lt;form method=&quot;POST&quot; action=&quot;/new&quot;&gt;      {{ form.hidden\_tag() }}      {{ form.submitter.label }} {{ form.submitter(size=40) }}      &lt;br/&gt;      {{ form.food.label }} {{ form.food(size=50) }}      &lt;br/&gt;      {{ form.submit }}  &lt;/form&gt;  &lt;/html&gt; |

模板里<form> 的部分会渲染表单，将WTForm对象的输入在root() 视图传入模板。当表单被提交的时候，它会发送一个POST请求给 /new 终端，它会用下面的函数来处理。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | from flask import url\_for, redirect    @app.route(u'/new', methods=[u'POST'])  def newlunch():      form = LunchForm()      if form.validate\_on\_submit():          lunch = Lunch()          form.populate\_obj(lunch)          db.session.add(lunch)          db.session.commit()      return redirect(url\_for('root')) |

在验证表单数据后，我们把内容放入我们的模型对象，然后将其提交进数据库。一旦我们在数据库里存储午餐信息后，它会把别人吃的午餐展示出来。

Python



|  |  |
| --- | --- |
| 1  2  3 | if \_\_name\_\_ == "\_\_main\_\_":      db.create\_all()  # make our sqlalchemy tables      app.run() |

最后，我们只做了一点工作就完成了应用。我们用SQLAlchemy创建数据表来存储午餐信息，然后运行我们写的路由处理器。

### 6.2 Django的示例应用

Django版本的wut4lunch跟Flask版本的比较类似，但它分成了Django项目里的几个文件。首先，看最相近的部分：数据模型。唯一的区别就是SQLAlchemy 在声明一个数据库时的细微差别。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | # from wut4lunch/models.py  from django.db import models    class Lunch(models.Model):      submitter = models.CharField(max\_length=63)      food = models.CharField(max\_length=255) |

在表单系统上。与Flask不同，Django有内建的表单系统来让我们使用。它看起来很像我们在Flask里使用的WTForm模块，只是语法稍有不同。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | from django import forms  from django.http import HttpResponse  from django.shortcuts import render, redirect    from .models import Lunch    # Create your views here.    class LunchForm(forms.Form):      """Form object. Looks a lot like the WTForms Flask example"""      submitter = forms.CharField(label='Your name')      food = forms.CharField(label='What did you eat?') |

现在我们需要建立一个LunchForm的实例，并将其传入模板。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | lunch\_form = LunchForm(auto\_id=False)    def index(request):      lunches = Lunch.objects.all()      return render(          request,          'wut4lunch/index.html',          {              'lunches': lunches,              'form': lunch\_form,          }      ) |

render函数在Django里是一个捷径，用来接受请求，模板路径和上下文的字典。与Flask里的 render\_template 类似，但它也能接受进来的请求。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | def newlunch(request):      l = Lunch()      l.submitter = request.POST['submitter']      l.food = request.POST['food']      l.save()      return redirect('home') |

保存表单，应答数据库是不同的，没有用一个全局的数据库session，Django让我们调用模型的 save() 方法 并且能清楚的处理session，很巧妙！

Django提供了一些很棒的特性，能让我们管理用户提交的午餐，因而我们可以删除掉那些看起来不适合我们网站的午餐。Flask和Pyramid没有自动提供这个功能，在Django里不用为你的应用另写一个管理页面。毕竟码农的时间也是时间！你需要做的，仅仅是在wut4lunch/admin.py 文件里加两行代码就行了。

Python



|  |  |
| --- | --- |
| 1  2 | from wut4lunch.models import Lunch  admin.site.register(Lunch) |

现在，我们就可以添加和删除数据，而不用做额外的工作。

最后，我们来看看主页模板里面的区别：

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | &lt;ul&gt;  {% for lunch in lunches %}  &lt;li&gt;&lt;strong&gt;{{ lunch.submitter }}&lt;/strong&gt; just ate &lt;strong&gt;{{ lunch.food }}&lt;/strong&gt;&lt;/li&gt;  {% empty %}  &lt;em&gt;Nobody has eaten lunch, you must all be starving!&lt;/em&gt;  {% endfor %}  &lt;/ul&gt; |

Django里有很方便的方法把其他的视图引入你的页面。url标签使得调整你应用里的URL而不用破坏视图提供了可能。这能够好使是因为url标签能够向上寻找URL所指向的视图。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5 | &lt;form action=&quot;{% url &#039;newlunch&#039; %}&quot; method=&quot;post&quot;&gt;    {% csrf\_token %}    {{ form.as\_ul }}    &lt;input type=&quot;submit&quot; value=&quot;I ate this!&quot; /&gt;  &lt;/form&gt; |

表单被用不同语法渲染， 我们需要在表单里包含一个CSRF token ，这些不同主要是修饰性的。

6.3 Pyramid 的例子应用

终于，我们来看一下用Pyramid写的例子应用。与Django和Flask最大的区别就是模板。稍微改动Jinja2模板就能解决Django的问题。Pyramid的Chameleon模板的语法比XSLT显得更加怀旧。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | &lt;!-- pyramid\_wut4lunch/templates/index.pt --&gt;  &lt;div tal:condition=&quot;lunches&quot;&gt;      &lt;ul&gt;          &lt;div tal:repeat=&quot;lunch lunches&quot; tal:omit-tag=&quot;&quot;&gt;              &lt;li tal:content=&quot;string:${lunch.submitter} just ate ${lunch.food}&quot;/&gt;          &lt;/div&gt;      &lt;/ul&gt;  &lt;/div&gt;  &lt;div tal:condition=&quot;not:lunches&quot;&gt;      &lt;em&gt;Nobody has eaten lunch, you must all be starving!&lt;/em&gt;  &lt;/div&gt; |

就像Django的模板，缺少 for-else-endfor 的结构让逻辑看起来有点冗长。在这个例子里，我们用 if-for 和 if-not-for 语句块来提供相同的功能。模板使用XHTML标签可能看起来与Django的不太一样，AngularJS风格的模板使用 {{ 或 {% 来进行结构和条件控制。

Chameleon模板的一大优点是你选择的编辑器可以正确的将语法高亮，因为模板是基于XHTML的。对Django和Flask的模板来说，你的编辑器需要能够支持相应的模板语言并且将其正确高亮。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | &lt;b&gt;What are YOU eating?&lt;/b&gt;  &lt;form method=&quot;POST&quot; action=&quot;/newlunch&quot;&gt;      Name: ${form.text(&quot;submitter&quot;, size=40)}      &lt;br/&gt;      What did you eat? ${form.text(&quot;food&quot;, size=40)}      &lt;br/&gt;      &lt;input type=&quot;submit&quot; value=&quot;I ate this!&quot; /&gt;  &lt;/form&gt;  &lt;/html&gt; |

Pyramid里渲染表单的方法可能有点冗长，因为 pyramid\_simpleform 没有等同于Django的表单函数form.as\_ul 的能够自动渲染表单字段的功能。

现在，我们来看这个应用。首先，我们需要定义表单和生成首页。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | # pyramid\_wut4lunch/views.py  class LunchSchema(Schema):      submitter = validators.UnicodeString()      food = validators.UnicodeString()    @view\_config(route\_name='home', renderer='templates/index.pt')  def home(request):      lunches = DBSession.query(Lunch).all()      form = Form(request, schema=LunchSchema())      return {'lunches': lunches, 'form': FormRenderer(form)} |

获取所有午餐信息的查询语句与Flask是类似的，因为两个应用都使用了流行的SQLAlchemy ORM 来提供持久化功能。Pyramid里，你可以直接返回你的模板上下文字典，而不用特别调用render函数。@view\_config装饰器会自动把你返回的内容渲染到模板。能够不调用render函数，使得在Pyramid里写视图能够更加容易测试，因为返回的数据没有被湮没在一个模板渲染对象里。

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | @view\_config(route\_name='newlunch', renderer='templates/index.pt', request\_method='POST')  def newlunch(request):      l = Lunch( submitter=request.POST.get('submitter', 'nobody'),              food=request.POST.get('food', 'nothing'), )        with transaction.manager:          DBSession.add(l)      raise exc.HTTPSeeOther('/') |

在Pyramid里能更容易的从请求对象里获取数据，它会自动把表单发送POST请求里的数据解析，变成一个我们可以访问的字典。为了防止在同一时间，多个并发的请求访问数据库引发问题，ZopeTransaction模块为数据库组提供了上下文管理器，将它写进逻辑可以防止你的应用出现某个线程覆盖其他线程数据的问题（线程安全），这在你的应用有很大流量并且使用一个全局session访问数据库的时候是个严重的问题。

## 7 总结

Pyramid在这三个框架里是最灵活的。它可以用来写小的应用，它也能来支持像Dropbox这样大名鼎鼎的网站。像Fedora这样的开源社区选择它来做一些应用， 比如他们的社区badges\_system，它会获取关于很多项目工具的事件来给用户颁发成就奖。对Pyramid最多的抱怨是它有如此多的选择，在开始新项目的时候可能会有点纠结。

目前最流行的框架是Django，有一堆网站用它。有Bitbucket、Pinterest、Instagram、The Onion来完成网站功能的全部或一部分。对于一些有普遍需求的网站，选Django是非常理智的，因为它对于中到大型的web应用是个非常流行的选择。

Flask适合开发者用最快的速度做一个简单的，Python做后端的网站。它适合一些一次性的工具，或者一些基于现有API的简单web应用。需要一个简单的web接口的后端项目可以开发的很快，一些需要少量配置的应用可以在Flask的前端上受益，比如jitviewer ，它就可以提供web接口来查看 PyPi的即时编译日志。

这三个框架都对一些需求提供了解决方案，我们可以来看看它们的区别。有些区别不仅仅是表面的，它会影响你如何设计产，多快能实现特性并且修复问题。因为我们的例子都很小，我们可以看到在小规模项目的时候，Flask非常棒，而Django就有点笨重了。Pyramid的灵活性没有成为一个要素，因为我们的需求是一样的，但是现实delig中的需求都是非常随机的。

### 7.1 结尾致谢

标题图片里的logo来自Flask Django Pyramid 项目的官网。

能完成这篇文章要感谢它的审稿人，Remy DeCausemaker、Ross Delinger 和 Liam Middlebrook，对初稿提出了大量意见。

这边文章收到很多评论和指正，感谢 Adam Chainz、bendwarn、Sergei Maertens、Tom Leo、wichert。