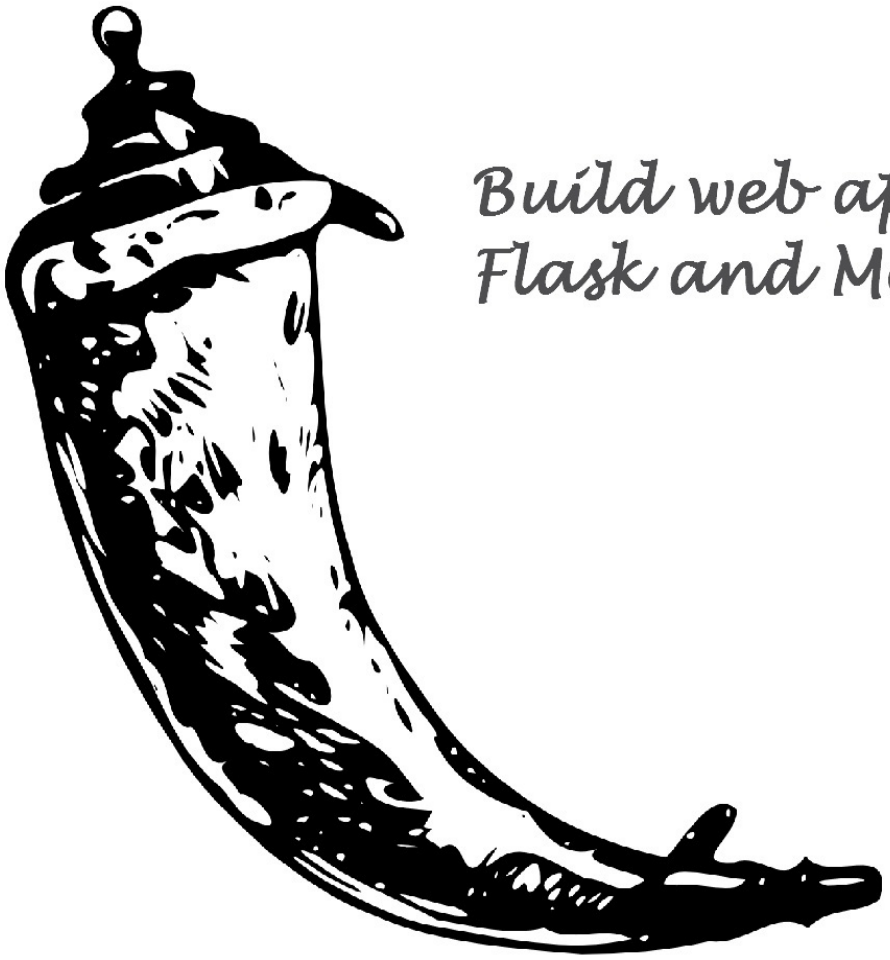
Flask Tutorial



Build web app based on
Flask and MongoEngine

defshine

Table of Contents

Flask Tutorial

欢迎来到Flask教程
本教程主要面向初学者，老鸟可以直接飞过。
Python中有很多Web开发框架，而Flask可以说是其中的佼佼者。
那为什么选择Flask呢？

1. 文档。Flask的官方文档齐全，初学者可以直接从它入手学习。
2. 微内核。这点与Django这种全栈式框架形成鲜明的对比，Flask尽量保持内核的精简，同时提供良好的扩展机制，给开发者更大的选择空间。易用，灵活，不失强大。
3. Flask源码。Flask本身就是很好的Python学习资料，熟练使用后，深入学习源码，是一条很好的提高途径。

既然，Flask的文档很好，为何还要写这样一个教程？
为了更快，更好的入门~
如何做到更快更好呢？

1. 小项目驱动。麻雀虽小五脏俱全，覆盖开发中的基础知识。
2. 小节讲解，由浅入深，循序渐进。
3. 不懂运维的开发不是好开发，所以还会教你如何部署项目。

小项目就是一个简单的todo应用，后台数据库使用MongoDB，名字就叫awesome-flask-todo，必须高大上。
再开始本教程前，需要一点点学前知识：

1. Python基础语法。看看官网文档就好。
2. 基本的前端知识。CSS，HTML，了解bootstrap更好。
3. 了解点MongoDB。去官网看看，安装一下，简单感受一下。
4. Git。项目使用git管理，放在github上。每一小节的开发都会打上Tag，记录学习过程。

总之，千言万语不如现在就手动，与其纠结选择哪个框架，不如直接开始本教程，因为本教程的学习时间远远小于你纠结的时间。
所以，现在就开始吧~

About Flask

Flask是一个微内核的Web开发框架.
Flask主要依赖两个库：

1. Jinja2。模板引擎。
2. Werkzeug。一个WSGI套件。

如何理解微内核呢？文档里是这样说的：

> The “micro” in microframework means Flask aims to keep the core simple but extensible.

内核精简，易于扩展，功能不失强大。
比如，一般的Web应用都需要和数据库打交道。在某些框架中，比如Django已经集成了ORM框架，而Flask不会束缚你，你可以使用SQLAlchemy，也可以使用MongoEngine，当然你也可以不用ORM，直接基于底层驱动进行开发，选择权完全在你的手中。 开源社区中，有大量的优秀的Flask扩展，所以，Don't repeat youself!大胆的去使用他们，提高开发效率。本教程中也有使用一些扩展，主要有：

1. Flask-WTF。基于WTForms的表单验证扩展。
2. flask-mongoengine。MongoEngine(Document-Object Mapper,类似关系数据库中的ORM)的Flask扩展，方便我们操作MongoDB。同时集成WTForm，统一数据库和表单的验证。
3. Flask-Script。为Flask应用提供编写脚本的功能。可以用来运行一个开发服务器，也可以与数据库交互，方便开发。

Hello World

从Hello World开始Flask之旅
在写代码之前，先在github上创建仓库，名字就叫awesome-flask-todo，之后所有的代码都提交到这里。
将仓库克隆到本地

```
git clone https://github.com/defshine/awesome-flask-todo.git
```

创建app.py文件

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def index():
    return "Hello World"

if __name__ == "__main__":
    app.run()
```

这就是一个最简单的Flask应用。我们尝试运行它，然后通过浏览器进行访问

```
$ python app.py
```

此时，只要访问http://127.0.0.1:5000/,就可以看到Hello World
那这些代码都做了什么呢？

1. 导入Flask类，使用这个类我们可以创建应用。
2. 创建app，也就是一个Flask应用实例。
3. 创建index方法，使用app.route装饰器绑定路由规则，当用户访问http://127.0.0.1:5000/时，运行该方法，这样我们在浏览器上看到了返回结果Hello World。
4. 运行Flask应用。

相信，做过Java开发的同学对些代码会有很熟悉的感觉，这几行代码和SpringMVC的Controller中的写法很相似，SpringMVC中使用RequestMapping注解，这里是使用route装饰器定义路由规则，随着学习的深入，可以自行进行对比学习。
完成Hello World后，给项目打上Tag,并推送到远程仓库。记录自己由浅入深的学习过程，方便回头查看。

```
git tag -a v0.1 -m 'Hello World'
git push origin v0.1
```

Project

工程结构

一个项目往往有很多文件，为了更好的对项目进行管理，同时也为了后期的扩展，需要好好设计一下工程的骨架结构。
删除之前的app.py文件，创建如下的目录结构

```
├── app --app包目录
│  ├── __init__.py --初始化app包
│  ├── models.py --数据模型
│  ├── static --静态文件目录
│  │  ├── bootstrap.css --样式文件
│  │  └── index.css --样式文件
│  ├── templates --模板目录
│  │  └── index.html --模板文件
│  └── views.py --视图
├── config.py --应用配置信息，比如数据库配置
├── manage.py --脚本，比如启动服务器，与数据库交互
├── README.md --项目的说明
├── requirenments.txt --项目依赖
└── run.py --项目运行脚本
```

使用app包来组织应用，前端使用Bootstrap，也加入进来。

安装依赖

首先，在requirenments.txt添加项目的依赖。

> Flask
> flask-mongoengine
> Flask-Script

然后使用pip安装这些依赖。

```
$ sudo pip install -r requirenments.txt
```

重新修改项目

在这个项目中，实现Hello World。在__init__.py中

```
from flask import Flask

app = Flask(__name__)
app.config.from_object("config")

from app import views, models
```

此时，在app包中实例化Flask的应用，从config.py中加载配置。
在view.py中

```
from app import app
from flask import render_template

@app.route('/')
def index():
    return render_template("index.html", text="Hello World")
```

此时，不是直接返回数据，而是通过模板渲染数据。
在index.hmtl模板文件中，最终会通过{{}}将Hello World展示出来。更多的展现方式参考 Jinja2文档。

```
{{ text }}
```

然后，可以在Flask-Script中，运行项目。往manage.py中，加入代码

```
# -×- coding: utf-8 -*-

from flask.ext.script import Manager, Server
from app import app

manager = Manager(app)

manager.add_command("runserver",
         Server(host='0.0.0.0',port=5000, use_debugger=True))
if __name__ == '__main__':
    manager.run()
```

此时，这样运行这个项目

```
$ python manage.py runserver
```

此时，又可以在浏览器中看到亲切的Hello World
完成本次调整后，提交代码并且打上标签v0.2

PS：

对于开发工具来说，推荐Vim和Pycharm。 至于到底如何选择，看个人习惯。对于编写少量的脚本用Vim更便捷一些，配置几个插件很顺手，对于新手会折腾一点。如果之前用过IDEAD同学，直接使用pycharm更快一些。

Datebase

本教程中使用MongoDB存储数据，Flask应用通过MongoEngine与MongoDB交互。

安装

在Ubuntu中，只要

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
$ echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' | sudo tee /etc/apt/sources.list.d/mongodb
$ sudo apt-get update
$ sudo apt-get install -y mongodb-org
```

其他系统，可以参考官方文档进行安装。

运行MongoDB

安装完成后，可以通过以下两个命令，启动或者关闭MongoDB

```
$ sudo service mongod start
$ sudo service mongod stop
```

在终端中使用mongo命令打开MongoDB的控制台进行操作。为了方便操作数据库，可以安装第三方的控制台工具。推荐使用Robomongo,开源免费的跨平台工具。

配置MongoDB

在工程中配置MongoDB，首先在config.py中进行添加

```
MONGODB_SETTINGS = {'DB': 'todo_db'}
```

开发

在__init__.py中导入MongoEngine，并且实例化

```
from flask import Flask
from flask.ext.mongoengine import MongoEngine

app = Flask(__name__)
app.config.from_object("config")

db = MongoEngine(app)

from app import views, models
```

然后创建数据模型，用于映射MongoDB中的Document对象，在model.py中

```
from app import db
import datetime

class Todo(db.Document):
    content = db.StringField(required=True, max_length=20)
    time = db.DateTimeField(default=datetime.datetime.now())
    status = db.IntField(default=0)
```

比较简单，只有三个字段，分别表示：todo的内容，todo的发布时间，todo的完成状态。
在manage.py中加入数据库的命令

```
@manager.command
def save_todo():
    todo = Todo(content="my first todo")
    todo.save()
```

然后，在终端中运行

```
$ python manage.py save_todo
```

此时，使用Robomongo，就可以查看到插入的的数据。
完成后，提交代码，打上Tag，v0.3

Get Todos

使用manage.py插入几条todo，然后，我们实现在前端展示所有todo的功能。

前端

在模板中，使用表格展示所有的todo

```
<table class="table">
    <thead>
        <tr>
             <td>Content</td>
             <td>Time</td>
             <td>Status</td>
       </tr>
    </thead>
    <tbody>
    {% for t in todos %}
        <tr>
            <td>{{ t.content }}</td>
            <td>{{ t.time }}</td>
            <td>{{ t.status }}</td>
       </tr>
    {% endfor %}
    </tbody>
</table>
```

使用一个循环，输出了所有的todo。

后台

在view.py中，要从后台数据库获取所有的todo

```
@app.route('/')
def index():
    todos = Todo.objects.all()
    return render_template("index.html", todos=todos)
```

在index方法中

1. 获取所有的todo
2. 返回模板和所有的todo

运行项目，在浏览器中就可以看到所有的todo。
完成后，提交代码，打上Tag，v0.4

Save Todo

之前我们都是用manage.py脚本保存的Todo，那如何在前端浏览器上添加呢？

1. 使用表单接受输入，并传给Flask后台。
2. Flask获取到前端来的数据后，将数据保存到MongoDB中。

前端

现在index.html模板中，加入一个表单,可以输入todo的内容。

```
<form class="input-group" action="/add" method="post">
    <input class="form-control" id="content" name="content" type="text" value="">
    <span class="input-group-btn">
         <button class="btn btn-primary" type="submit">Add</button>
    </span>
</form>
```

后台

然后在view.py中

```
@app.route('/add', methods=['POST', ])
def add():
    content = request.form.get("content")
    todo = Todo(content=content)
    todo.save()
    todos = Todo.objects.all()
    return render_template("index.html", todos=todos)
```

获取到数据，将数据保存。
运行项目，在浏览器中，我们添加一个todo，是不是很简单？
记得提交代码，并打上新的Tag，v0.5

Update Todo

实现了保存与展示Todo的功能，那如何控制Todo的完成与未完成的状态呢？这时候就可以使用status标志位，0就是未完成，1就是完成，所以，更新它的值就可以了。

前端

先在前端每条todo后，加入do和undo两个按钮：

1. 如果todo的status未完成，则显示do按钮。
2. 如果todo的status是完成，则显示undo按钮。

```
<table class="table">
 <thead>
     <tr>
         <td>Content</td>
         <td>Time</td>
     <td>Operation</td>
 </tr>
 </thead>
 <tbody>
 {% for t in todos %}
 <tr>
     <td>{{ t.content }}</td>
     <td>{{ t.time }}</td>
     <td>
     {% if t.status == 0 %}
      <a href="/done/{{ t.id }}" class="btn btn-primary">Done</a>
     {% else %}
      <a href="/undone/{{ t.id }}" class="btn btn-primary">Undone</a>
     {% endif %}
     </td>
 </tr>
 {% endfor %}
 </tbody>
</table>
```

修改了之前的展示表格，修改最后一列用于显示按钮，按钮要根据todo的状态进行显示。

后台

前端写好后，开始添加后台的支持，在view.py中

```
@app.route('/done/<string:todo_id>')
def done(todo_id):
    todo = Todo.objects.get_or_404(id=todo_id)
    todo.status = 1
    todo.save()
    todos = Todo.objects.all()
    return render_template("index.html", todos=todos)


@app.route('/undone/<string:todo_id>')
def undone(todo_id):
    todo = Todo.objects.get_or_404(id=todo_id)
    todo.status = 0
    todo.save()
    todos = Todo.objects.all()
    return render_template("index.html", todos=todos)
```

两个方法思路是一样的，从前端传来todo的id，然后，根据id获取todo后更新status。

运行项目，就可以对todo的状态就可以进行修改了，这个版本，打上Tag，v0.6

Delete Todo

当完成一个todo，想删除它，那该如何实现呢？

1. 前段每个todo后面都加一个删除按钮。
2. 后端响应前端，从数据库中删除todo。

前端

上一小结已经加过按钮，删除按钮直接加在后面即可。

```
 <tbody>
{% for t in todos %}
    <tr>
        <td>{{ t.content }}</td>
        <td>{{ t.time }}</td>
        <td>
            {% if t.status == 0 %}
                 <a href="/done/{{ t.id }}" class="btn btn-primary">Done</a>
            {% else %}
                 <a href="/undone/{{ t.id }}" class="btn btn-primary">Undone</a>
            {% endif %}
            <a href="/delete/{{ t.id }}" class="btn btn-danger">Delete</a>
        </td>
    </tr>
{% endfor %}
</tbody>
```

后台

后台，添加删除方法，view.py中

```
@app.route('/delete/<string:todo_id>')
def delete(todo_id):
    todo = Todo.objects.get_or_404(id=todo_id)
    todo.delete()
    todos = Todo.objects.all()
    return render_template("index.html", todos=todos)
```

delete方法会根据传来的id，找到todo，然后进行删除。
有了之前的学习，这个是不是感觉很简单？
好了，打上Tag，v0.7

Validate

基本的CRUD我们已经完成了，但是我们想想，当前程序有什么问题？

1. 如果我们不输入内容，直接点击添加todo的按钮。
2. 如果我们输入的内容很长很长。

这两种情况，我们的程序会有问题，所以我们需要进行表单验证。
WTForm就是专门来做表单验证的，而且和Flask-MongoEngine可以完美结合，而无需再添加Form相关的类，用过SqlAlchemy的同学都知道。
直接在model.py中，

```
from app import db
import datetime
from flask.ext.mongoengine.wtf import model_form


class Todo(db.Document):
    content = db.StringField(required=True, max_length=20)
    time = db.DateTimeField(default=datetime.datetime.now())
    status = db.IntField(default=0)

TodoForm = model_form(Todo)
```

只要通过Todo类生成TodoForm类即可，这样，数据库中的验证条件就和表单验证条件达成一致。

然后我们需要修改view.py，在add方法中

```
@app.route('/add', methods=['POST', ])
def add():
    form = TodoForm(request.form)
    if form.validate():
        content = form.content.data
        todo = Todo(content=content)
        todo.save()
    todos = Todo.objects.all()
    return render_template("index.html", todos=todos, form=form)
```

其他方法也要添加对form的支持。
最好，需要修改前端。

```
 <form class="input-group" action="/add" method="post">
    {{ form.hidden_tag() }}
    {{ form.content(class="form-control") }}
    <span class="input-group-btn">
        <button class="btn btn-primary" type="submit">Add</button>
    </span>
</form>
{% for error in form.errors.content %}
<div class="flash alert-danger"><span>{{ error }}</span></div>
{% endfor %}
```

修改了表单，在下面添加了错误提示。此时，再按照开始说的那两种情况进行操作，就会分别出现提示：

1. This field is required.
2. Field cannot be longer than 20 characters.

有了表单验证，我们的程序就更加健壮，让我们赶紧打上新的Tag，v08

More

这一节，我们对应用的体验一步思考。
1.如果没有todo，应该提示用户
2.用不同的颜色表示todo的状态。
3.对时间进行格式化，显示更友好。
4.对todo排序，让最新的Todo排在最前面。

所以在模板页面上进行修改

```
<div>
<h2>Todo List</h2>
{% if todos %}
    <table class="table">
    <thead>
        <tr>
            <th>content</td>
            <th>time</td>
            <th>operation</td>
        </tr>
    </thead>
    <tbody>
     {% for t in todos %}
        {% if t.status == 0 %}
            <tr class="info">
                <td>{{ t.content }}</td>
                <td>{{ t.time.strftime('%H:%M %d-%m-%Y') }}</td>
                <td>
                    <a href="/done/{{ t.id }}" class="btn btn-primary">Done</a>
                    <a href="/delete/{{ t.id }}" class="btn btn-danger">Delete</a>
                </td>
            </tr>
         {% else %}
              <tr class="danger">
                <td id="center" >{{ t.content }}</td>
                <td>{{ t.time.strftime('%H:%M %d-%m-%Y') }}</td>
                <td>
                    <a href="/undone/{{ t.id }}" class="btn btn-primary">Undone</a>
                    <a href="/delete/{{ t.id }}" class="btn btn-danger">Delete</a>
                </td>
            </tr>

         {% endif %}
        {% endfor %}
    </tbody>
    </table>
{% else %}
    <h3 class="text-info">No todos,pelease add</h3>
{% endif %}
</div>
```

如果没有todo，我们会提示用户添加，如果有todo才进行显示，而且用不同的颜色显示todo的状态。 那如何进行排序呢？
需要在后台进行修改，例如在view.py的index方法中

```
@app.route('/')
def index():
    form = TodoForm()
    todos = Todo.objects.order_by('-time')
    return render_template("index.html", todos=todos, form=form)
```

按照time字段进行降序，如果想用升序呢？把减号改成加号即可，so eazy ~
这样我们的程序就基本完成了，最终它时这个样子的。

所以此时，给项目打上Tag，v0.9, 别着急，还有工作要做。

Virtualenv

什么是Virtualenv？

从字面就可以看出，它是虚拟环境，官方是这样描述的：

> virtualenv is a tool to create isolated Python environments.

为什么需要Virtualenv

开发中，版本的不同是一个很头疼的问题，所以，有了很多版本管理工具。
在nodejs中，每个项目的依赖可以全局安装，也可以局部安装，但是python只能全局安装。
Virtualenv可以为应用创建一个独立的依赖运行环境，所以，它很好的解决了开发以及部署中的版本依赖问题。

使用Virtualenv

可以通过pip直接安装

```
$ sudo pip install virtualenv
```

然后，我们在终端命令中，进入项目根目录

```
$ virtualenv venv
$ source env/bin/activate
(venv)$ pip install -r requirenments.txt
(venv)$ python manage.py runserver
```

上面的命令都做了什么呢？

1. 创建虚拟环境venv（你也可以起其他的名字），此时就会有一个venv文件夹生成。
2. 启动虚拟环境venv
3. 在虚拟环境中安装项目依赖
4. 运行项目

再次打开浏览器，就可以访问todo应用了。
如果想退出虚拟环境呢？只要执行

```
(venv)$ deactivate
```

Deploy

如何在生产环境中部署Flask应用呢？
我们选择这样的部署方案：

> Virtualenv+Gunicorn+Nginx+Supervisor

注意

1. 使用Ubuntu14.04（64位）系统，使用defshine用户进行部署。
2. 部署目录是（/home/defshine/www/awesome-flask-todo），所以请注意配置文件中的目录。

环境

1. 系统:Ubuntu 14.04 64
2. Web Server: Nginx
3. 虚拟环境: Virtualenv
4. WSGI Server: Gunicorn
5. 数据库: MongoDB
6. Monitor: Supervisor

使用supervisor主要是监控gunicorn的运行，保证服务器的可以持续运行。

安装

安装软件

```
$ sudo apt-get install nginx
$ sudo apt-get install supervisor
```

MongoDB以及Virtualenv安装之前已经讲过了

创建虚拟环境

项目放到到（/home/defshine/www/）下
启动虚拟环境，安装工程依赖

```
$ virtualenv venv
$ source venv/bin/activate
(venv)$ pip install -r requirements.txt
```

如何退出虚拟环境

```
(venv)$ deactivate
```

开启监控

根据自己的情况，编辑工程下的supervisor配置文件（awesome-flask-todo.conf），然后复制到系统目录中

```
$ sudo cp etc/awesome-flask-todo.conf /etc/supervisor/conf.d/
```

重新载入配置文件，并启动awesome-flask-todo

```
$ sudo supervisorctl reload
$ sudo supervisorctl start awesome-flask-todo
```

查看运行状态

```
$ sudo supervisorctl status
```

Nginx

修改nginx的配置文件（awesome-flask-todo），然后复制到系统目录中去，并创建软链接。重启nignx。

```
$ sudo cp etc/awesome-flask-todo /etc/nginx/site-available/
$ cd  /etc/nginx/site-enabled
$ sudo ln -s /etc/nginx/site-avalaible/awesome-flask-todo .
$ sudo service nginx reload
$ sudo service nginx restart
```

查看nginx状态

```
$ sudo service nginx status
```

然后，就可以通过127.0.0.1地址访问了。 通过部署awesome-flask-todo这个小项目，也学习到了很多东西，麻雀虽小，五脏俱全啊。

Summary

到此为止，一个Flask应用，从开发到部署就完成了。
小项目结束了，但是Flask的学习才刚刚开始。
回头看一下所有的Tag从v0.1最终到了v1.0，会不会感觉好充实？你可以回头，checkout不同的Tag版本，再次回顾学习的过程。
相信这个简单的小教程会开启你的Flask学习之路 ~
当然，我也继续学习Flask，所以，这个小教程还会继续升级 ~

关于本教程有任何疑问或者任何建议，都可以联系我
Email：crazyxin1988@gmail.com
Blog:defshine
Github:defshine