

# Hello Flask

---

源代码: <https://github.com/Itoddy/flask-tutorial>

技术交流群:630398887(欢迎一起吹牛)

写在前面的话：这里我假设你电脑已经安装好了Python3，本项目基于Python3开发。(没有pip没关系)

## 什么是pip?

pip就是一个软件包管理，因为有各种人士开发了python的第三方库，但是这些库是不在标准库中的，这些库发布在PyPi上。所以可以使用pip这个工具来自动下载。

## 环境搭建:

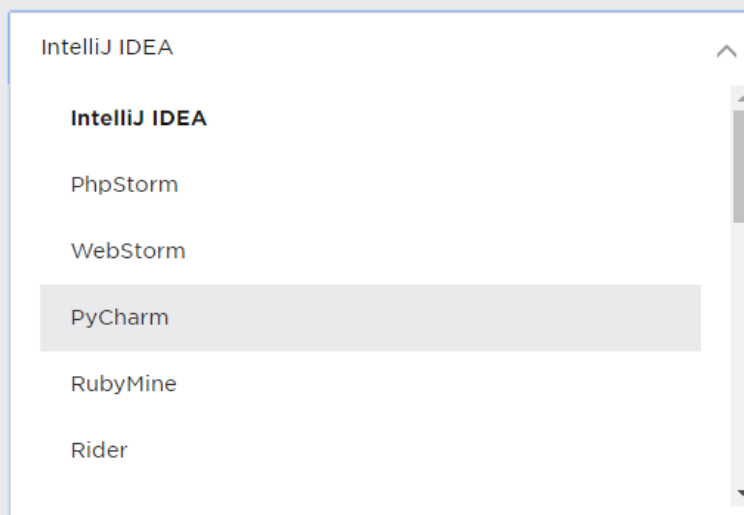
---

所选用的集成开发环境(也就是开发软件)是：Pycharm Professional

去jetbrains官网 [click me](#)

### Download the tool you need

Make sure you're using the most up-to-date version of your favorite JetBrains tool.





Version: 2017.2  
Build: 172.3317.103  
Released: July 26, 2017

[System requirements](#)  
[Installation instructions](#)  
[Previous versions](#)

## Download PyCharm

Windows

macOS

Linux

### Professional

Full-featured IDE  
for Python & Web  
development

DOWNLOAD

Free trial

### Community

Lightweight IDE  
for Python & Scientific  
development

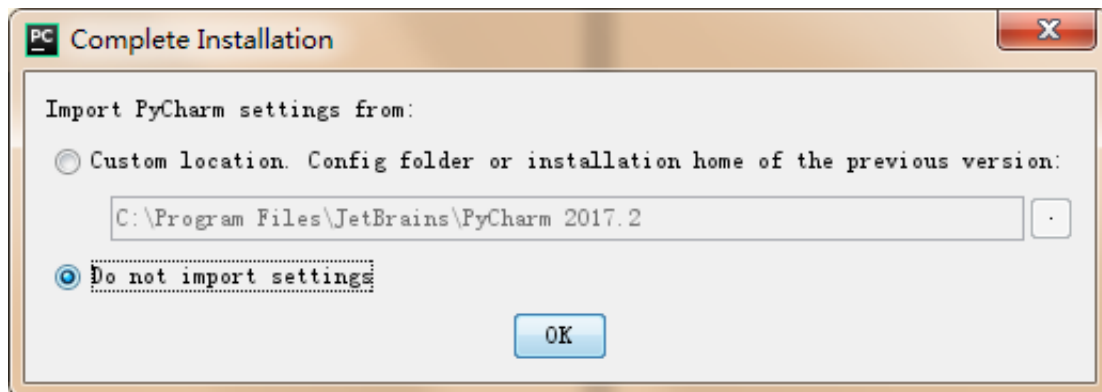
DOWNLOAD

Free, open-source

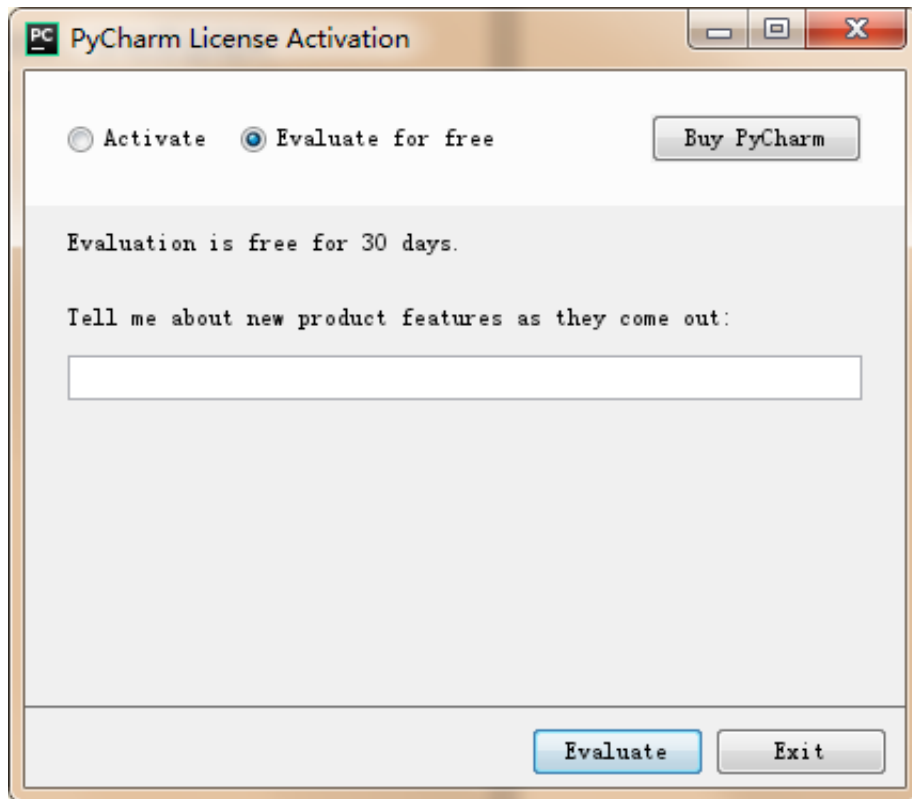
有两个版本一个是Professional(专业版)，一个是Community(社区版)。这里我们下载Professional版。

顺便提一句，这个软件内部已经集成好了pip，所以不需要自己手动去安装pip。

然后下载好Pycharm Professional之后，去安装，没什么问题就下一步就好。

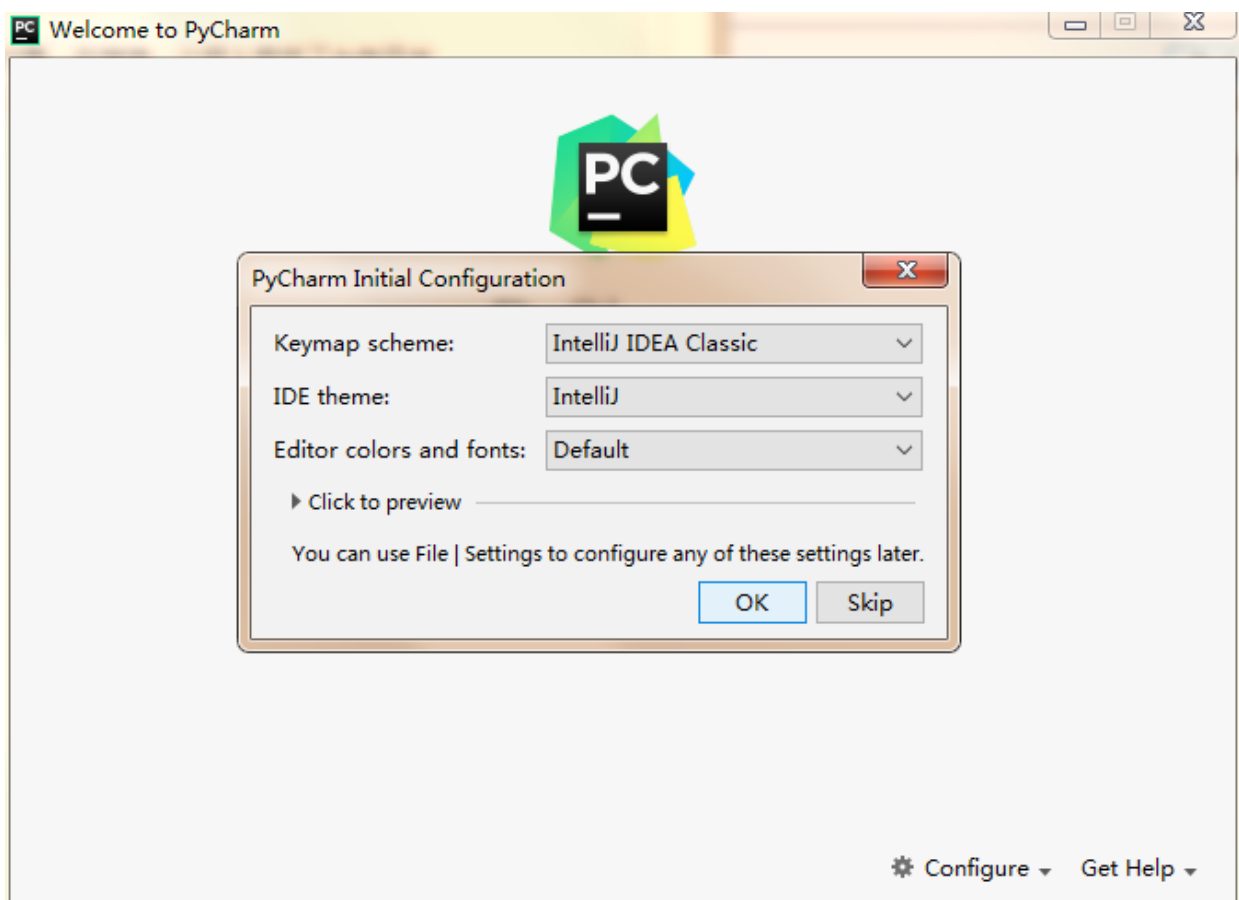


如果你是第一次下载使用，点击Do not import settings,这里的意思就是说，如果你之前用过这个软件，本地是由相关的配置文件的，比如保存着软件的主题，插件之类的东西。



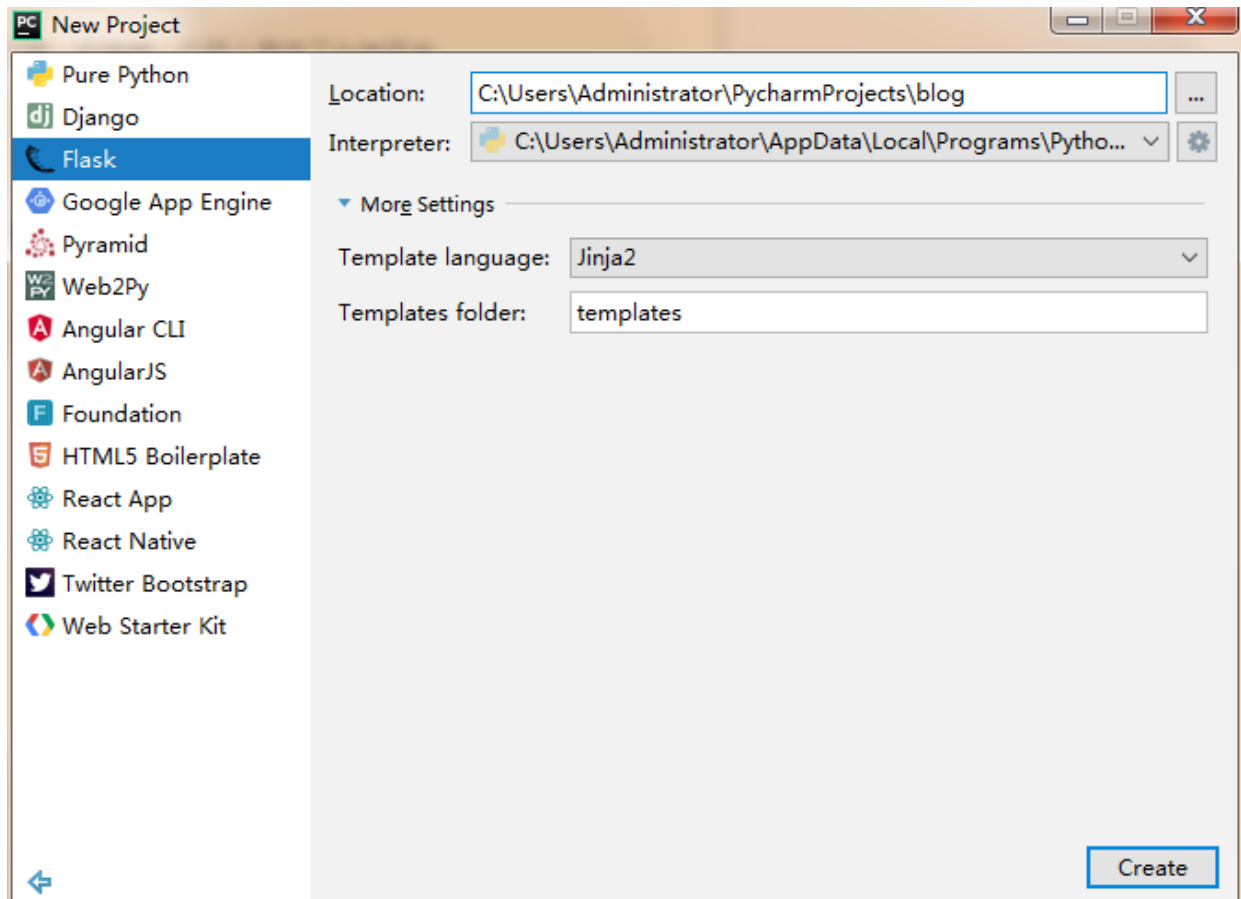
因为这是个付费软件，但是可以免费使用三十天。说句公道话，不要埋怨什么软件付费，你想想，公司人员花了力气开发出来东西，凭什么就让你白白使用。换做是谁都不愿意啊。

然后点击 Evaluate，然后点击Accept。



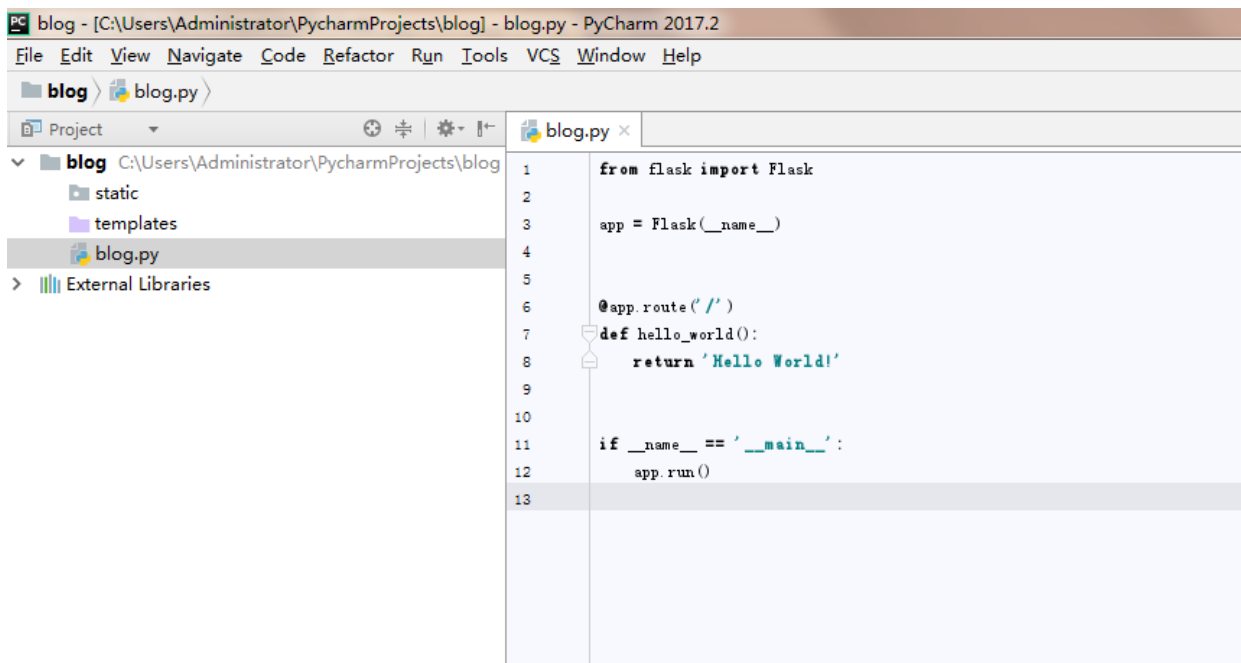
这里是让你去选择你软件的快捷键，主题，字体，颜色。选好之后点OK就好。

然后点击create new project

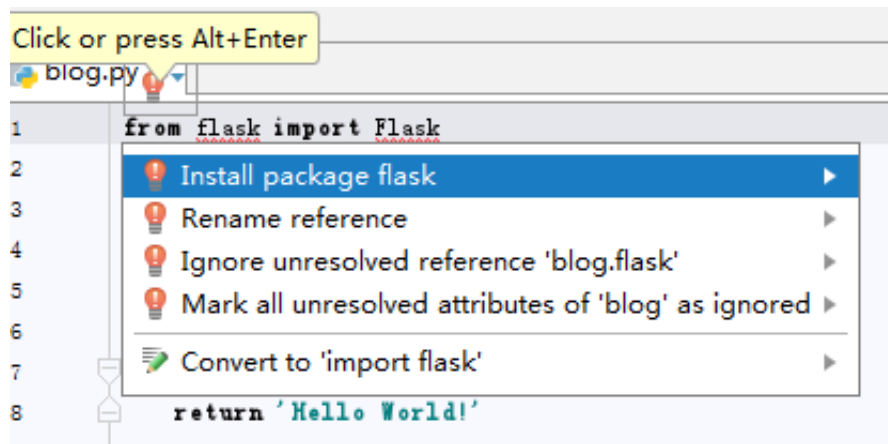


这里我们项目取名字为blog。(仅仅是把上面的Location中最后untitled改成了blog)。然后点击create。

然后等一小会，软件把Python3的标准库和第三方库导入进去。

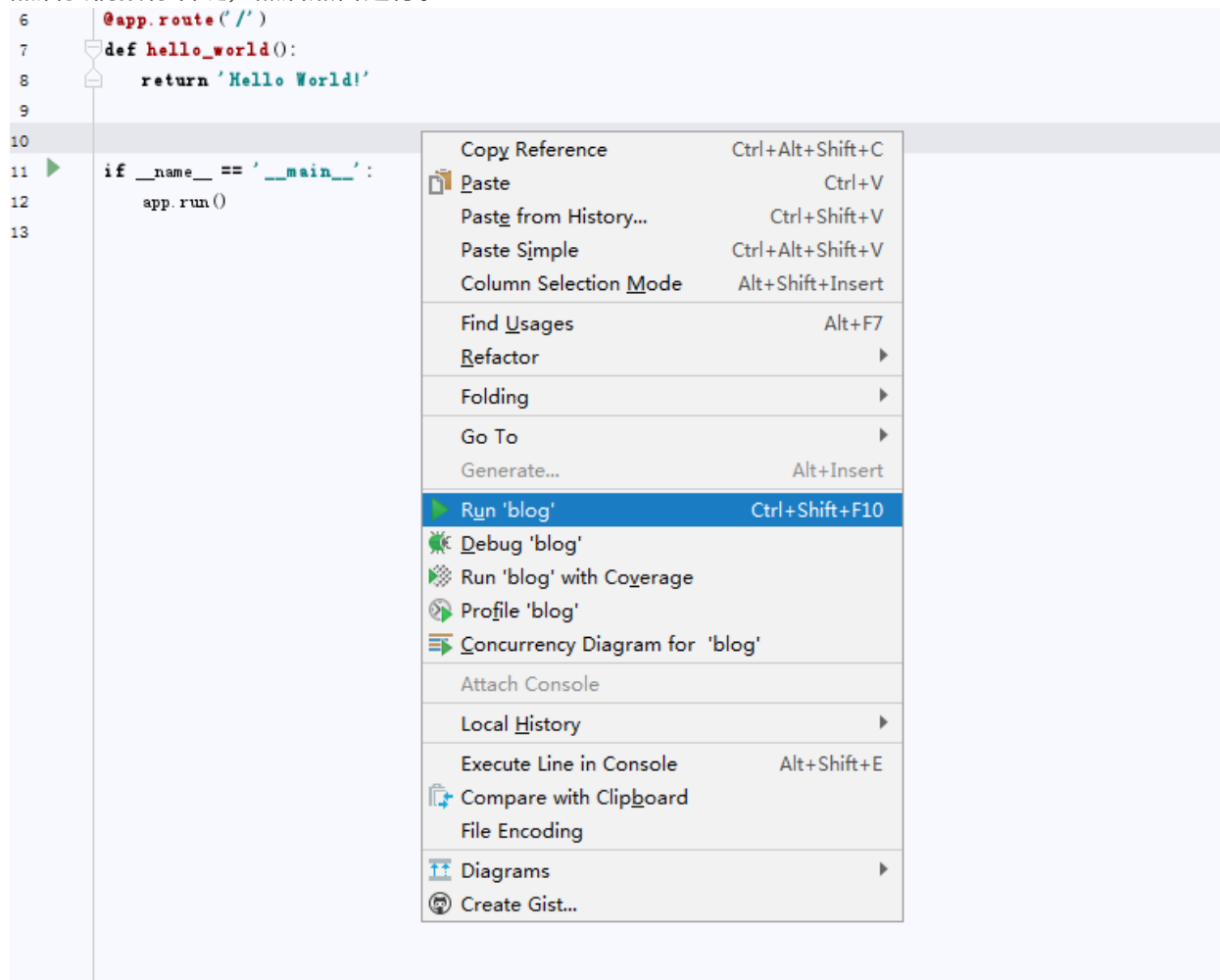


进去之后是如上图的样子。

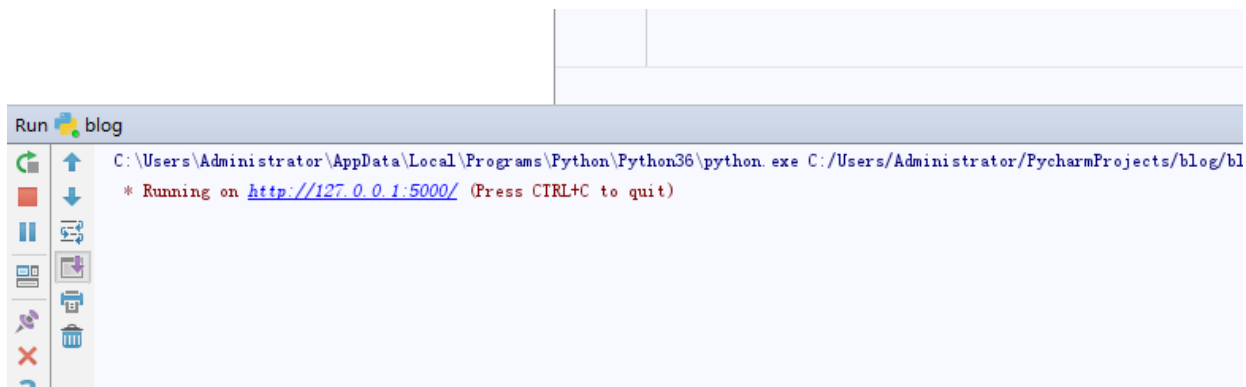


如果你电脑没有相关的第三方库(这里指的flask库, 和相关的扩展库)的话, 把鼠标一到有红色下划线的地方, 然后会有一个橙红色的灯泡, 点击一下, 然后点击第一个选项(Install package flask),让软件自身集成的pip给你安装好。然后等一小会, 就安装好了, 然后你就会看到红色下划线消失了。

然后我们鼠标右键, 然后点击运行。



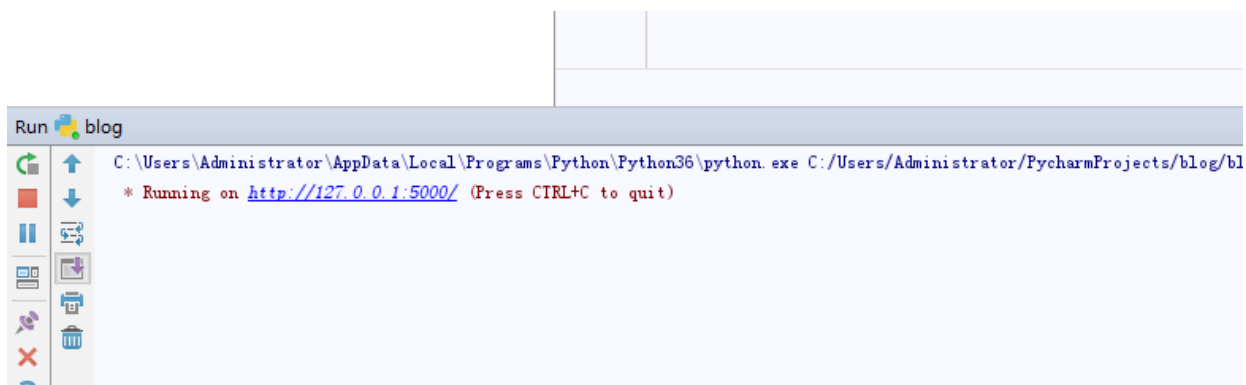
下方出现如下效果, 然后点击那个网址。



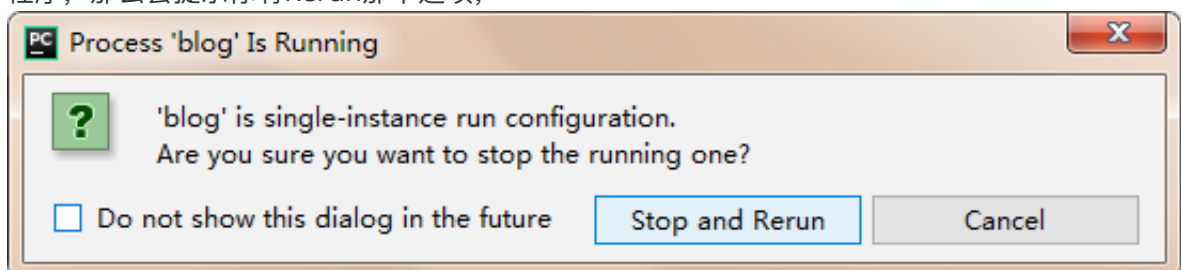
Hello World!

他会自动打开电脑默认的浏览器去浏览。

这里写给外行，非计算机人士



这在这里的最左侧，有这么几个功能按钮，Rerun(重新启动这个程序),Stop(停止这个程序)。让你不浏览程序的结果的时候，尽量就Stop这个程序。因为这个程序会占用电脑的5000端口号，然后每次运行就会占用这个端口号，当你程序已经运行之后，再更新了代码看效果的时候，如果你还想再去run这个程序，那么会提示你有Rerun那个选项，



点击

那个Stop and Rerun就好。不过这样子总感觉怪怪的。

再次回归正题。

感受一下动态路由。

代码如下:

blog/blog.py

```
from flask import Flask

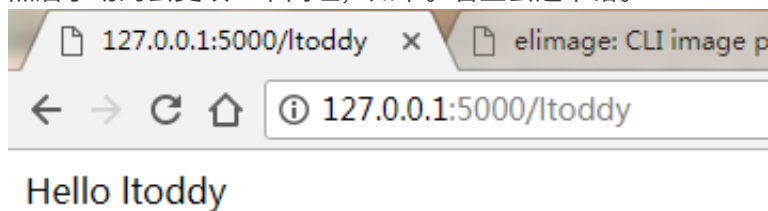
app = Flask(__name__)

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

@app.route('/<username>')
def user(username):
    return "Hello {}".format(username)

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

然后手动的去更改一下网址, 如下。看上去还不错。



## OK, 下面讲解一下Flask框架。

Flask主要有两个依赖:路由, Web服务器网关接口子系统(由Werkzeug提供); 和模板系统(由Jinja2提供, 下一节将会用到)。

简单说一下路由:

```
app = Flask(__name__)

@app.route('/')
def index():
    return 'Hello World!'
```

简单说一下这个部分代码，`app = Flask(name)`，用来创建一个Flask应用，一般Flask类构造函数只有一个必须指定的参数，即程序主模块或包的名字，在大多数程序中，Python的`name`变量就是所需要的值。`@app.route('/')`这个东西叫路由，程序实例需要知道对每个URL(网址)请求运用那些代码，所以保存了一个URL到Python函数的映射关系。处理URL和函数之间的关系的程序称为路由。而下面所修饰的`index()`函数被叫做视图函数，他来展示你的web页面的样子。

最后说一点，我是linux用户，之后的讲解我就在我的ubuntu上写了，如果windows上和linux上有区别的地方，我会以windows为主，毕竟我有两台电脑。

---

## Jinja2模板引擎, 使用Twitter Bootstrap

---

源代码: <https://github.com/ltoddy/flask-tutorial>

技术交流群:630398887(欢迎一起吹牛)

有些地方没看懂没关系，坚持往下看，下面会有演示代码来说明。

上一篇中如下代码

```
@app.route('/')
def hello_world():
    return 'Hello World!'
```

这个样子返回一个字符串，很不爽，能不能直接返回一个HTML页面，当然可以。

在templates文件夹下新建一个index.html

templates/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Just for fun</title>
</head>
<body>

<h1>Hello World</h1>

</body>
</html>
```

blog.py



```
from flask import render_template
#...
@app.route('/')
def hello_world():
    return render_template('index.html')
```

这个样子就直接把那个index.html页面显示了出来.

## Jinja2模板引擎

### 渲染模板

在blog.py的def user(username) 这个函数中,如果我们想把那个username传入html页面怎么办呢?  
在templates文件夹下新建user.html

templates/user.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Just for fun</title>
</head>
<body>

<h1>Hello {{ name }}.</h1>

</body>
</html>
```

视图函数user()返回的响应中包含一个使用变量表示的动态部分. blog.py

```
@app.route('/<username>')
def user(username):
    return render_template('user.html', name=username)
```

注意看这里的对应关系,在return那里,render\_template接受了如下两个部分:

- 'user.html' 这个是要展现的页面名字,页面要放在templates文件夹下,jinja2引擎会自动去templates文件夹寻找此文件.
- 第二部分就是user.html中所需要的参数,这里可以有很多个参数,下文将会看到.在这里,name=username,左边的name表示参数名,就是模板中使用的占位符,右边的username是当前作用域中的变量.

### 变量

刚才使用的`{{ name }}`结构表示一个变量,它是一个特殊的占位符,告诉模板引擎这个位置的值从渲染模板时使用的数据中获取. Jinja2能识别所有类型的变量,甚至是一些特殊的类型:例如列表,字典和对象.  
e.g.

```
A value from a dictionary: {{ mydict['key'] }}.
A value from a list: {{ mylist[3] }}.
A value from a list, with a variable index: {{ mylist[myintvar] }}
A value from a object's method: {{ myobj.somemethod() }}.
```

Jinja2另外也提供了叫 *过滤器* 的东西,但是不怎么用,这里就不再说明了。

[Jinja2过滤器完整文档](#)

下面是一些知识的介绍,了解一下就好.

## 控制结构

条件控制结构语句:

```
{% if user %}
    Hello, {{ user }}!
{% else %}
    Hello, Stranger!
{% endif %}
```

## 循环控制结构

```
<ul>
    {% for comment in comments %}
        <li>comment</li>
    {% endfor %}
</ul>
```

需要在多出重复使用的模板代码片段可以写入单独的文件,再包含在所有模板中,以避免重复:

```
{% include 'common.html' %}
```

另一种重复使用代码的强大方式是模板继承. 首先创建一个名为base.html的模板

```

<html lang="en">
<head>
    {% block head %}
        <title>{% block title %}{% endblock %} - My Application</title>
    {% endblock %}
</head>
<body>
    {% block body %}
    {% endblock %}
</body>
</html>

```

block标签定义的元素可在衍生模板中修改.例如

```

{% extends 'base.html' %}
{% block title %}Index{% endblock %}
{% block head %}
    {{ super() }}
    <style>
    </style>
{% endblock %}
{% block body %}
<h1>Hello, World!</h1>
{% endblock %}

```

block标签定义的元素可以在衍生的模板中修改. extends指令声明这个模板衍生自base.html.在extends指令之后,基模板中的3个块被重新定义,模板引擎会将其插入适当的位置.

## 使用Flask-Bootstrap集成Twitter Bootstrap

如果电脑上有pip的可以:

```
pip install flask-bootstrap
```

没有的就如上篇文章那样,通过Pycharm自动帮你安装. Bootstrap CSS样式库,简单点理解就是你获得了一大堆化妆品(定制好的,而非按照自己想法定制的样子),可以有选择的去使用.

初始化Flask-Bootstrap:

```

from flask_bootstrap import Bootstrap
# ...
bootstrap = Bootstrap(app)

```

待会要通过继承Bootstrap的base.html页面来实现模板的复用. 先看看bootstrap的base.html为我们提供了什么:

```
{% block doc -%}
    <!DOCTYPE html>
    <html{% block html_attribs %}{% endblock html_attribs %}>
    {% - block html %}
        <head>
            {% - block head %}
                <title>{% block title %}{% endblock title %}</title>

                {% - block metas %}
                    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
                {% - endblock metas %}

                {% - block styles %}
                    <!-- Bootstrap -->
                    <link href="{{
bootstrap_find_resource('css/bootstrap.css', cdn='bootstrap') }}"
rel="stylesheet">
                {% - endblock styles %}
            {% - endblock head %}
        </head>
        <body{% block body_attribs %}{% endblock body_attribs %}>
        {% block body -%}
            {% block navbar %}
            {% - endblock navbar %}
            {% block content -%}
            {% - endblock content %}

            {% block scripts %}
                <script src="{{ bootstrap_find_resource('jquery.js',
cdn='jquery') }}"></script>
                <script src="{{ bootstrap_find_resource('js/bootstrap.js',
cdn='bootstrap') }}"></script>
            {% - endblock scripts %}
        {% - endblock body %}
    </body>
    {% - endblock html %}
</html>
{% endblock doc -%}
```

主要用到两个块: navbar(导航栏)和content(页面主要内容).

在templates文件夹下新建base.html文件

templates/base.html

```

{% extends 'bootstrap/base.html' %}

{% block title %}Just for fun{% endblock %}

{% block navbar %}
    <nav class="navbar navbar-inverse">
        <div class="container-fluid">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-
toggle="collapse" data-target=".navbar-collapse">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a class="navbar-brand" href="/">Just for fun</a>
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li><a href="/">Home</a></li>
                </ul>
            </div>
        </div>
    </nav>
{% endblock %}

{% block content %}
    <div class="container">
        {% block page_content %}{% endblock %}
    </div>
{% endblock %}

```

然后我们更改一下index.html和user.html templates/index.html

```

{% extends 'base.html' %}

{% block page_content %}
    <div class="page-header">
        <h1>Hello World!</h1>
    </div>
{% endblock %}

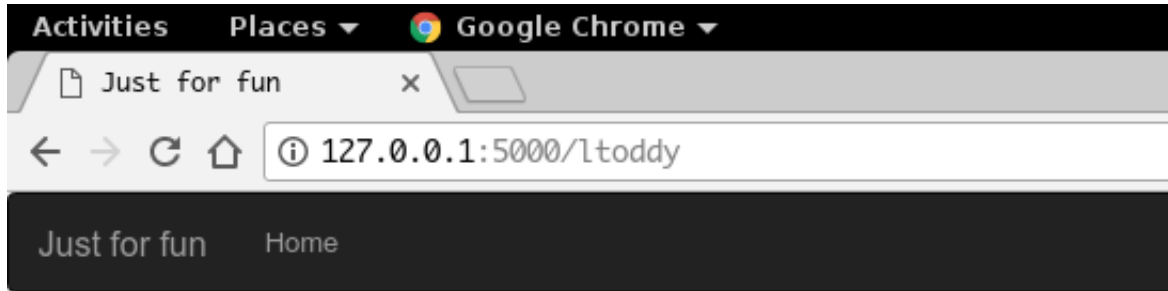
```

templates/user.html

```
{% extends 'base.html' %}

{% block page_content %}
    <div class="page-header">
        <h1>Hello {{ name }}.</h1>
    </div>
{% endblock %}
```

OK,我们运行一下看看效果:



# Hello ltoddy.

还不错.

## 自定义错误页面.

有时候输入一个错误的网址的时候,会返回一个404 page not found错误. 分别在templates文件夹下创建404.html和500.html

templates/404.html

```
{% extends 'base.html' %}

{% block title %}404 - Page Not Found{% endblock %}

{% block page_content %}
    <div class="page-header">
        <h1>Not Found</h1>
    </div>
{% endblock %}
```

templates/500.html

```
{% extends 'base.html' %}

{% block title %}500 - Internal Server Error{% endblock %}

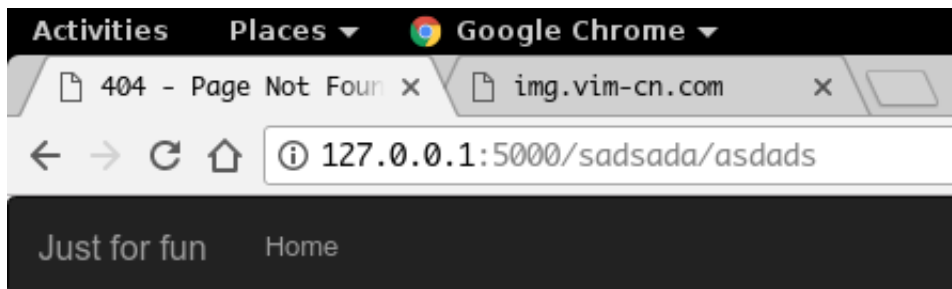
{% block page_content %}
    <div class="page-header">
        <h1>Server Error</h1>
    </div>
{% endblock %}
```

以及我们要在blog.py中添加相应的路由:

blog.py

```
@app.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404

@app.errorhandler(500)
def internal_server_error(e):
    return render_template('500.html'), 500
```



# Not Found

看一下blog.py的完整代码: blog.py

```
from flask import Flask
from flask import render_template
from flask_bootstrap import Bootstrap

app = Flask(__name__)
bootstrap = Bootstrap(app)
```

```

@app.route('/')
def hello_world():
    return render_template('index.html')

@app.route('/<username>')
def user(username):
    return render_template('user.html', name=username)

@app.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404

@app.errorhandler(500)
def internal_server_error(e):
    return render_template('500.html'), 500

if __name__ == '__main__':
    app.run(debug=True)

```

在最后一行,我加入了:debug=True,目的是当你对程序做出了改变之后,不需要手动重启项目,项目会自动帮你做出重启。

最后说一点内容,下一篇将会用到

## 使用Flask-Script支持命令行选项

安装Flask-Script:

```
pip install flask-script
```

Flask的开发web服务器支持很多启动设置选项,传递设置选项的理想方式是使用命令行参数。

e.g.:

```

from flask_script import Manager
manager = Manager(app)

# ...

if __name__ == '__main__':
    manager.run()

```

此时的完整代码: blog.py



```

from flask import Flask
from flask import render_template
from flask_bootstrap import Bootstrap
from flask_script import Manager

app = Flask(__name__)
bootstrap = Bootstrap(app)
manager = Manager(app)

@app.route('/')
def hello_world():
    return render_template('index.html')

@app.route('/<username>')
def user(username):
    return render_template('user.html', name=username)

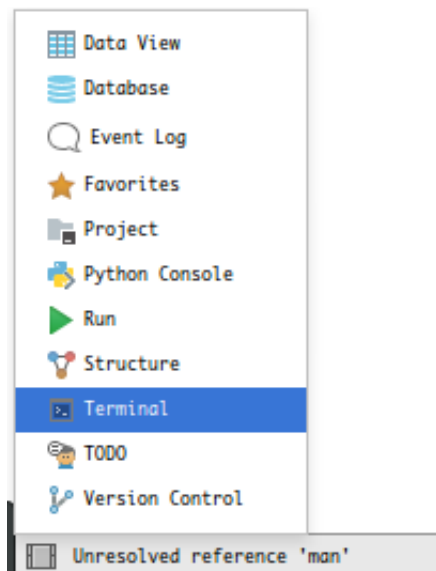
@app.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404

@app.errorhandler(500)
def internal_server_error(e):
    return render_template('500.html'), 500

if __name__ == '__main__':
    manager.run()

```

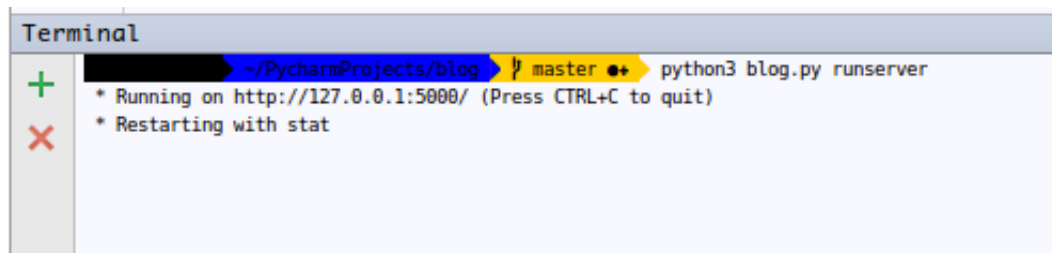
现在如何启动程序？



在软件的左下角,选择点击Terminal, 输入:

```
python3 blog.py runserver
```

如下:



这里要注意一下,我目前使用的是我的Linux系统,Python2和Python3都有,所以这里写明是Python3,如果你电脑上只有Python3,那么你把这里的python3替换成python就可以了.按照自己电脑为主.

然后打开浏览器,在地址栏填入: <http://localhost:5000/> 回车就可以了.

## Web表单

源代码: <https://github.com/ltoddy/flask-tutorial>

技术交流群:630398887(欢迎一起吹牛)

```
pip install flask-wtf
```

先看看一个普通的HTML页面的表单的样子:

```
<form action="">
  <label>你叫什么名字: <input type="text"></label><br>
  <input type="button" value="提交">
</form>
```

你叫什么名字:

提交

也就是说阿,在你要填写的框框前有一个提示的标语 (label) , 然后有一个提交的按钮,按钮上写着提交俩字。

```

from flask_wtf import FlaskForm
from wtforms import StringField
from wtforms import SubmitField

class NameForm(FlaskForm):
    name = StringField('你叫什么名字?') # 这里的'你叫什么名字?'就是对应的那个提示语
    # StringField() 就是那个输入的框框
    submit = SubmitField('提交') # 这里的'提交'对应着按钮的文字

```

先大体浏览一下下面两个表格，然后我在具体讲解怎么使用

## WTForms支持的HTML标准字段

字段类型	说明
StringField	文本字段
TextAreaField	多行文本字段
PasswordField	密码文本字段
HiddenField	隐藏文本字段
DateField	文本字段，值为datetime.date格式
DateTimeField	文本字段，值为datetime.datetime格式
IntegerField	文本字段，值为整数
DecimalField	文本字段，值为decimal.Decimal
FloatField	文本字段，值为浮点数
BooleanField	复选框，值为True和False
RadioField	一组单选框
SelectField	下拉列表
SelectMultipleField	下拉列表，可选择多个值
FileField	文件上传字段
SubmitField	表单提交按钮
FormField	把表单作为字段嵌入另一个表单
FieldList	一组指定类型的字段

## WTForms验证函数

验证函数	说明
Email	验证电子邮件地址
EqualTo	比较两个字段的值，常用于要求输入两次密码进行确认的情况
IPAddress	验证IPv4网络地址
Length	验证输入字符串的长度
NumberRange	验证输入的值在数字范围内
Optional	无输入值时跳过其他验证函数
Required	确保字段中有数据
Regexp	使用正则表达式验证输入值
URL	验证URL
AnyOf	确保输入值在可选值列表中
NoneOf	确保输入值不在可选列表中

## 把表单加入到页面中去:

顺便提一句，之前的那个hello\_world函数，我把它改名为index了。先看我们的html页面：

templates/index.html

```
{% extends 'base.html' %}
{% import 'bootstrap/wtf.html' as wtf %}
{% block page_content %}
    <div class="page-header">
        <h1>Hello {% if name %}{{ name }}{% else %}stranger!{% endif %}
    </h1>
    </div>
    {{ wtf.quick_form(form) }}
{% endblock %}
```

第二行：

```
{% import 'bootstrap/wtf.html' as wtf %}
```

bootstrap为我们集成好了一个宏（理解成函数就好了），它可以很方便的把我们的表单类(刚才的NameForm)渲染成表单。

再看下面：

```
{% <h1>Hello {% if name %}{{ name }}{% else %}stranger!{% endif %}</h1> %}
```

如果有名字近来，那么显示名字，没有名字的话就显示stranger（陌生人）。

再来看一下视图函数：

blog.py

```
class NameForm(FlaskForm):
    name = StringField('你叫什么名字', validators=[Required()])
    submit = SubmitField('提交')

@app.route('/', methods=['GET', 'POST'])
def index():
    form = NameForm()
    name = None
    if form.validate_on_submit():
        name = form.name.data
        form.name.data = ''
    return render_template('index.html', name=name, form=form)
```

注意看，和刚才有差别。

第一个是没有那个validators=[Required()]这一部分的，这段是什么意思呢，就是说你在框框中填写的内容是有要求的，这个Required()的要求是，框框中的内容不为空才可以提交。如果你什么都没写，然后提交，就会出现：

Hello stranger!

你叫什么名字

提交

⚠ Please fill out this field.

类似这样的情况。具体看每个人电脑的具体实现了。还有一点，表单的提交要通过POST方法，这里不再多余说了，具体去看HTTP协议。

但是这个页面还是有问题的，当你按下F5去刷新页面的时候，会给你个提示：问你表单是不是要重新提交一下。因为页面刷新会向浏览器重新发送最后一个请求，这里的请求是提交表达，基于这个原因，我们利用重定向，来改成GET请求。

blog.py

```

@app.route('/', methods=['GET', 'POST'])
def index():
    form = NameForm()
    if form.validate_on_submit():
        session['name'] = form.name.data
        form.name.data = ''
        return redirect(url_for('index'))
    return render_template('index.html', name=session.get('name', None),
                           form=form)

```

这里我用到了一个东西：session（会话），它会记住上下文，把数据存储在这个session中。怎么理解呢？浏览器是一个傻子，他只能记住最后发生的一件事情，这个session就是为了强行让浏览器记住一些东西。还用到了redirect和url\_for这两个组合,url\_for便于我们生成url，当然那一行代码你也可以写成：

```

redirect('/')

```

因为就是要回到主页嘛，主页地址就是'/'，但是随着程序日益的复杂，你不可能完全掌握所有的地址，所以我们通过url\_for来获得地址，url\_for('index')注意看括号中的参数内容'index'对应着视图函数index()的名字。也就是说重新定向到这个函数映射的页面上。

看一下完整代码：

blog.py

```

from flask import Flask
from flask import render_template
from flask import redirect
from flask import url_for
from flask import session
from flask_bootstrap import Bootstrap
from flask_script import Manager
from flask_wtf import FlaskForm
from wtforms import StringField
from wtforms import SubmitField
from wtforms.validators import Required

app = Flask(__name__)
app.config['SECRET_KEY'] = 'a string' # 这一行是必须要加上的，为了防止CSRF攻击
bootstrap = Bootstrap(app)
manager = Manager(app)

class NameForm(FlaskForm):
    name = StringField('你叫什么名字', validators=[Required()])
    submit = SubmitField('提交')

```

```

@app.route('/', methods=['GET', 'POST'])
def index():
    form = NameForm()
    if form.validate_on_submit():
        session['name'] = form.name.data
        form.name.data = ''
        return redirect(url_for('index'))
    return render_template('index.html', name=session.get('name', None),
form=form)

@app.route('/<username>')
def user(username):
    return render_template('user.html', name=username)

@app.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404

@app.errorhandler(500)
def internal_server_error(e):
    return render_template('500.html'), 500

if __name__ == '__main__':
    manager.run()

```

再介绍一点更人性化的东西：

## Flash消息：

有些时候，当你作出了改动的时候，最好有个东西来提醒你:flash

blog.py

```

@app.route('/', methods=['GET', 'POST'])
def index():
    form = NameForm()
    if form.validate_on_submit():
        old_name = session.get('name')
        if old_name is not None and old_name != form.name.data:
            flash('你更改了名字')
        session['name'] = form.name.data
        form.name.data = ''
        return redirect(url_for('index'))
    return render_template('index.html', name=session.get('name', None),
form=form)

```

需要再更改一下我们的模板，让flash消息显示出来

templates/base.html

```
{% block content %}
    {% for message in get_flashed_messages() %}
        <div class="alert alert-info">
            <button type="button" class="close" data-dismiss="alert">&times;</button>
            {{ message }}
        </div>
    {% endfor %}
    <div class="container">
        {% block page_content %}{% endblock %}
    </div>
{% endblock %}
```

赶快去运行程序尝试提交表单看看效果吧。

## Sqlite数据库

源代码: <https://github.com/ltoddy/flask-tutorial>

技术交流群:630398887(欢迎一起吹牛)

用到的数据库是sqlite，这个数据库不需要安装(因为这个数据库的运行是基于文件系统的)，只要你电脑能运行C语言就行（是个能开机的电脑就可以.....）。

安装：

```
pip install flask-sqlalchemy
```

或者通过pycharm内置的pip安装

这里说一下数据库URL（待会要用到）。就是说阿，如果你要链接数据库，得先告诉程序数据库在哪，无论是local的或者remote的。URL就是那个数据库的位置。

数据库引擎	URL
SQLite(Unix)	sqlite:///absolute/path/to/database
SQLite(Windows)	sqlite:///c:/absolute/path/to/database

不同的系统对于SQLite的URL是有一点不同的，我说一下原因，在Unix系统家族中，有一个根目录"/"，Unix系统是不分盘符的，也就是没有C盘，D盘之类的，所有东西都在根目录之下。而windows系统呢是分盘符的，比如你有东西在C盘，那么对于C盘来说，它的根目录也就是c:/所以说，对于这两个系统数据库URL的切换，只需要 '/' 与 'c:/' 互相转换一下就好了。

配置数据库：



```

from flask_sqlalchemy import SQLAlchemy
import os

basedir = os.path.abspath(os.path.dirname(__file__))
# 我们要把数据库放在当下的文件夹,这个basedir就是当下文件夹的绝对路径

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:/// ' + \
    os.path.join(basedir,
        'data.sqlite')
# 这里注意看,写的是URI (统一资源标识符)
app.config['SQLALCHEMY_COMMIT_TEARDOWN'] = True
# SQLALCHEMY_COMMIT_TEARDOWN 因为数据库每次有变动的时候,数据改变,但不会自动的去改变数据库里面的数据,
# 只有你去手动提交,告诉数据库要改变数据的时候才会改变,这里配置这个代表着,不需要你手动
# 的去提交了,自动帮你提交了。
# 待会会有演示
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

```

简单说一下数据库：正规点的定义就是：数据库是长期存储在计算机内，大量有组织可共享的数据的集合。

简单点就是，比如你有一个database（数据库），数据库中有很多表格，就像excel那样子。表格中有每一列的名字（字段），用来标记每一列是存的什么信息。就这么简单，当然数据还会有些特殊的属性，比如primary key, unique, not null等等。

OK，我们来定义一下我们的表格。这个项目为了简单，只设立一个User表。用来之后的注册用的。

```

class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    # 如果你学过数据库的话就知道我们一般通过id来作为主键,来找到对应的信息的,通过id来实现唯一性
    name = db.Column(db.String(64), unique=True)

```

下面将有一堆知识点，没空看的话就跳过去好了，等遇到了这方面的问题再来查看。我复制的，太多了手打得累死我。

## 常见的SQLALCHEMY列类型

类型名称	python类型	描述
Integer	int	常规整形，通常为32位
SmallInteger	int	短整形，通常为16位
BigInteger	int或long	精度不受限整形
Float	float	浮点数
Numeric	decimal.Decimal	定点数
String	str	可变长度字符串
Text	str	可变长度字符串，适合大量文本
Unicode	unicode	可变长度Unicode字符串
Boolean	bool	布尔型
Date	datetime.date	日期类型
Time	datetime.time	时间类型
Interval	datetime.timedelta	时间间隔
Enum	str	字符列表
PickleType	任意Python对象	自动Pickle序列化
LargeBinary	str	二进制

## 常见的SQLALCHEMY列选项

可选参数	描述
primary_key	如果设置为True，则为该列表的主键
unique	如果设置为True，该列不允许相同值
index	如果设置为True，为该列创建索引，查询效率会更高
nullable	如果设置为True，该列允许为空。如果设置为False，该列不允许空值
default	定义该列的默认值

## 数据库操作

### 创建表

```
python blog.py shell
>>> from hello import db
>>> db.create_all()
```

## 删除表

```
db.drop_all()
```

## 插入行

```
#创建对象, 模型的构造函数接受的参数是使用关键字参数指定的模型属性初始值。
admin_role = Role(name='Admin')
user_role = Role(name='User')
user_susan = User(username='susan', role=user_role)#role 属性也可使用,虽然它不是真正的数据库列,但却是一对多关系的高级表示。
user_john = User(username='john', role=admin_role)
#这些新建对象的 id 属性并没有明确设定,因为主键是由 Flask-SQLAlchemy 管理的。
print(admin_role.id)#None
#通过数据库会话管理对数据库所做的改动,在 Flask-SQLAlchemy 中,会话由 db.session 表示。
##首先, 将对象添加到会话中
db.session.add(admin_role)
db.session.add(user_role)
db.session.add(user_susan)
db.session.add(user_john)
#简写: db.session.add_all([admin_role, user_role, user_john, user_susan])
##通过提交会话(事务), 将对象写入数据库
db.session.commit()
```

## 修改行

```
admin_role.name = 'Administrator'
db.session.add(admin_role)
session.commit()
```

## 删除行

```
db.session.delete(mod_role)
session.commit()
```

## 查询行

```
查询全部。Role.query.all()
条件查询（使用过滤器）。User.query.filter_by(role=user_role).all()
user_role = Role.query.filter_by(name='User').first()#filter_by() 等过滤器在
query 对象上调用,返回一个更精确的 query 对象。
```

## 常用过滤器

过滤器	说 明
filter()	把过滤器添加到原查询上,返回一个新查询
filter_by()	把等值过滤器添加到原查询上,返回一个新查询
limit()	使用指定的值限制原查询返回的结果数量,返回一个新查询
offset()	偏移原查询返回的结果,返回一个新查询
order_by()	根据指定条件对原查询结果进行排序,返回一个新查询
group_by()	根据指定条件对原查询结果进行分组,返回一个新查询

## 最常使用的SQLAlchemy查询执行函数

方 法	说 明
all()	以列表形式返回查询的所有结果
first()	返回查询的第一个结果,如果没有结果,则返回 None
first_or_404()	返回查询的第一个结果,如果没有结果,则终止请求,返回 404 错误响应
get()	返回指定主键对应的行,如果没有对应的行,则返回 None
get_or_404()	返回指定主键对应的行,如果没找到指定的主键,则终止请求,返回 404 错误响应
count()	返回查询结果的数量
paginate()	返回一个 Paginate 对象,它包含指定范围内的结果

OK,知识点到此为止。

我们来操作一下数据库: 更改这个路由

```
@app.route('/', methods=[ 'GET', 'POST' ])
def index():
```

代码如下:

```
@app.route('/', methods=[ 'GET', 'POST' ])
def index():
```

```

form = NameForm()
if form.validate_on_submit():
    user = User.query.filter_by(name=form.name.data)
    if user is None:
        user = User(name=form.name.data)
        db.session.add(user)
        session['known'] = False
    else:
        session['known'] = True
    session['name'] = form.name.data
    form.name.data = ''
    return redirect(url_for('index'))
return render_template('index.html',
                       form=form, name=session.get('name'),
                       known=session.get('known', False))

```

注意看 db.session.add(user)。

假设没有

```
app.config['SQLALCHEMY_COMMIT_TEARDOWN'] = True
```

这行代码。

那么那行db.session.add(user)的后面需要加上db.session.commit()才可以把数据放到数据库中。

对应的 /templates/index.html 也要更改一下:

```

{% extends 'base.html' %}
{% import 'bootstrap/wtf.html' as wtf %}
{% block page_content %}
    <div class="page-header">
        <h1>Hello {% if name %}{{ name }}{% else %}stranger!{% endif %}
    </h1>

    {% if not known %}
        <p>Nice to meet you!</p>
    {% else %}
        <p>Happy to see you again!</p>
    {% endif %}
    </div>
    {{ wtf.quick_form(form) }}
{% endblock %}

```

说一下，如果你要运行项目，然后打开页面可能会出现一些SQLAlchemy的异常，这个时候，首先去看一下

```

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:/// ' + \
    os.path.join(basedir, 'data.sqlite')

```

这个，我们关心的是最后data.sqlite这个文件是否存在。如果你没有发现这个文件的话，那么打开terminal(pycharm已经集成好了)。

```
python3 blog.py shell
>>> from blog import *
>>> db # 这一行目的是看看db被添加进来了么。
<SQLAlchemy engine=sqlite:///home/me/PycharmProjects/blog/data.sqlite>
>>> db.drop_all()
>>> db.create_all()
```

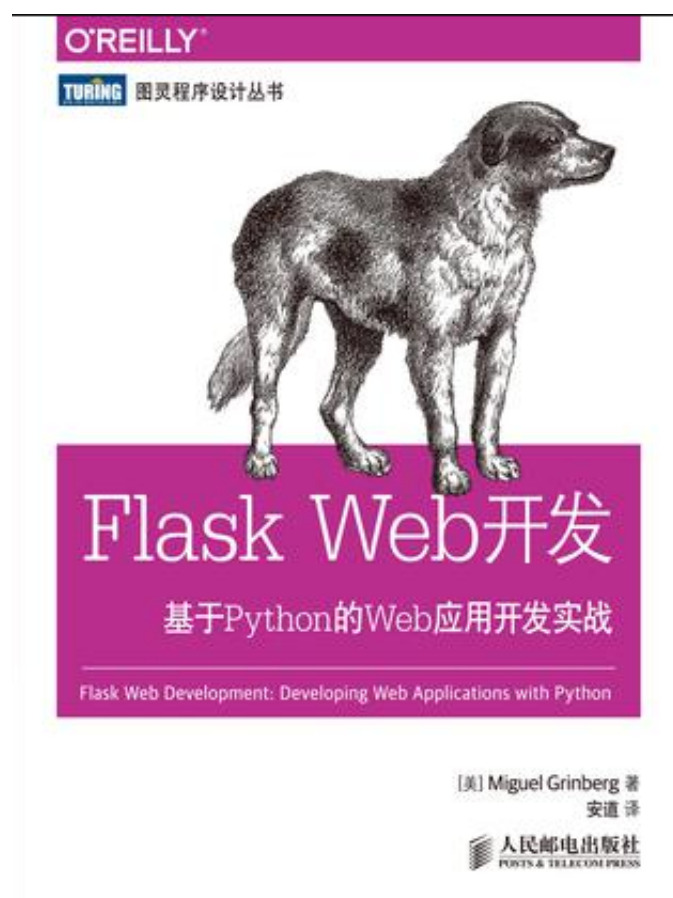
这个样子你就会看到data.sqlite出现在你的文件夹中了。

## 大型程序结构

源代码: <https://github.com/ltoddy/flask-tutorial>

技术交流群:630398887(欢迎一起吹牛)

文中提到的狗书,就是《Flask Web开发 基于Python的Web应用开发实战》,看过的人都是到,这本书坑挺多的.



就是这本,反正大家都叫狗书,我也就跟着叫了.....

参照狗书的内容,以及响应Dijkstra的模块化程序设计,我们这次要改一下程序结构,做一次大手术.

本篇会好好的说明一下蓝本(也叫蓝图).

蓝图简单一点就是,我们之前的程序不都是有一个Flask的实例

```
app = Flask(name)
```

也就是这个变量app,它可以来定义路由.蓝图就是可以将路由分门别类,然后在组合在一起. 不懂没关系,往下看.

我们需要重新设置一下项目结构:

```
.
├── app
│   ├── admin
│   │   ├── errors.py
│   │   ├── forms.py
│   │   ├── __init__.py
│   │   └── views.py
│   ├── __init__.py
│   ├── main
│   │   ├── forms.py
│   │   ├── __init__.py
│   │   └── views.py
│   ├── models.py
│   ├── static
│   └── templates
│       ├── 404.html
│       ├── 500.html
│       ├── admin
│       │   ├── login.html
│       │   └── register.html
│       ├── base.html
│       ├── index.html
│       └── user.html
├── config.py
└── manage.py
```

差不多是这个样子的.

- Flask 程序一般都保存在名为app的包中.
- config.py 保存着配置
- manage.py 用于启动项目

这里说明一下python 的包(package),从目录结构上看,python的package有两部分组成: 文件夹和 **init.py** 文件. 正是因为**init.py** 的存在 python编译器才会把那个文件夹当作是一个python的包来看待. 而那个 **init.py** 的效果就是, 能够有一个与包名字相同的文件. 什么意思呢?

比如 我们有一个名字为 main 的包, 那么

```
from main import *
```

这行代码中,从main包中import所有的东西,你想啊,main是个包,import进来是啥??? 其实阿,是import进来的是init.py中的内容.

把原先那个blog.py中的东西复制一下就好了.

config.py

```
import os

basedir = os.path.abspath(os.path.dirname(__file__))

class Config:
    SECRET_KEY = 'a string'
    # 数据库配置
    SQLALCHEMY_DATABASE_URI = 'sqlite:/// ' + os.path.join(basedir,
    'data.sqlite')
    SQLALCHEMY_COMMIT_TEARDOWN = True
    SQLALCHEMY_TRACK_MODIFICATIONS = False

    @staticmethod
    def init_app(app):
        pass
```

仿照狗书,创建一个程序工厂函数. 设计模式中有一个模式叫做:工厂模式,比如你需要一个东西,但这个东西你需要配置很多,这样你就可以用到工厂模式,在把配置(比如汽车厂,把各个零件组装起来)的活交给工厂,那么工厂出来的产品就是好的产品.这样可以降低程序的耦合度,怎么理解呢,如果这个产品是坏的,那么你也不需要到处去程序的代码,只需要去那个工厂的程序中去寻找bug就好了.

## 程序工厂函数:

说一下工厂函数,我们之前单个文件开发程序很方便,但却有个很大的缺点,因为程序在全局作用中创建,所以无法动态修改配置.运行脚本时,实例已经创建,再修改配置为时已晚,解决问题的方法就是延迟创建程序实例,把创建过程移到可显式调用的工厂函数中.

app/init.py

```
from flask import Flask
from flask.ext.bootstrap import Bootstrap
from flask.ext.sqlalchemy import SQLAlchemy

from config import Config

bootstrap = Bootstrap()
db = SQLAlchemy()

def create_app():
    app = Flask(__name__)
```



```
app.config.from_object(Config)
Config.init_app(app)

bootstrap.init_app(app)
db.init_app(app)

return app
```

看一下这段代码,我们的bootstrap和db实例,先于app创建,app的创建只能调用了create\_app()函数之后才创建.

## 蓝图

Flask 中的蓝图为这些情况设计:

- 把一个应用分解为一个蓝图的集合。这对大型应用是理想的。一个项目可以实例化一个应用对象,初始化几个扩展,并注册一集合的蓝图。
- 以 URL 前缀和/或子域名, 在应用上注册一个蓝图。URL 前缀/子域名中的参数即成为这个蓝图下的所有视图函数的共同的视图参数(默认情况下)。
- 在一个应用中用不同的 URL 规则多次注册一个蓝图。
- 通过蓝图提供模板过滤器、静态文件、模板和其它功能。一个蓝图不一定要实现应用或者视图函数。
- 初始化一个 Flask 扩展时, 在这些情况中注册一个蓝图。

蓝图问题,最后总结.往下看.

在蓝本中定义的路由处于休眠状态,直到蓝本注册到程序上后,路由才真正成为程序的一部分.

### 创建蓝本:

app/main/init

```
from flask import Blueprint

main = Blueprint('main', __name__)

from . import views, errors
```

蓝图是通过实例化Blueprint类对象来创建的。这个类的构造函数接收两个参数: 蓝图名和蓝图所在的模块或包的位置。与应用程序一样, 在大多数情况下, 对于第二个参数值使用Python的**name**变量即可。

应用程序的路由都保存在app/main/views.py模块内部, 而错误处理程序则保存在app/main/errors.py中。导入这些模块可以使路由、错误处理与蓝图相关联。重要的是要注意, 在app/init.py脚本的底部导入模块要避免循环依赖, 因为view.py和errors.py都需要导入main蓝图。

### 注册蓝图:

app/init.py

```

from flask import Flask
from flask.ext.bootstrap import Bootstrap
from flask.ext.sqlalchemy import SQLAlchemy

from config import Config

bootstrap = Bootstrap()
db = SQLAlchemy()

def create_app():
    app = Flask(__name__)
    app.config.from_object(Config)
    Config.init_app(app)

    bootstrap.init_app(app)
    db.init_app(app)

    from .main import main as main_blueprint
    app.register_blueprint(main_blueprint)

    return app

```

app.register\_blueprint(main\_blueprint)

通过register\_blueprint方法将蓝图注册进来.

app/main/errors.py

```

from flask import render_template

from . import main

@main.app_errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404

@main.app_errorhandler(500)
def internal_server_error(e):
    return render_template('500.html'), 500

```

app/main/views.py

```

from flask import render_template
from flask import session
from flask import redirect
from flask import url_for

```

```

from . import main
from .forms import NameForm
from ..models import User
from .. import db

@main.route('/', methods=['GET', 'POST'])
def index():
    form = NameForm()
    if form.validate_on_submit():
        user = User.query.filter_by(name=form.name.data)
        if user is None:
            user = User(name=form.name.data)
            db.session.add(user)
            session['known'] = False
        else:
            session['known'] = True
        session['name'] = form.name.data
        form.name.data = ''
        return redirect(url_for('main.index'))
    # 这里注意一下,我们这里要写main.index,因为我们这个视图函数隶属于main这个蓝
    # 本,
    # main.index是他完整的名字.
    return render_template('index.html',
                           form=form, name=session.get('name'),
                           known=session.get('known', False))

```

app/main/forms.py

```

from flask.ext.wtf import FlaskForm
from wtforms import StringField, SubmitField
from wtforms.validators import Required

class NameForm(FlaskForm):
    name = StringField('你叫什么名字', validators=[Required()])
    submit = SubmitField('提交')

```

manage.py

```
from flask.ext.script import Manager

from app import create_app, db

app = create_app()
manager = Manager(app)

if __name__ == '__main__':
    manager.run()
```

我们运行一下: 打开Pycharm为我们集成好的终端(Terminal)

```
python3 manage.py shell
>>> from manage import *
>>> db
<SQLAlchemy engine=sqlite:////home/me/PycharmProjects/blog/data.sqlite>
>>> db.drop_all()
>>> db.create_all()
>>> exit()
```

先把我们的表重新创建一下,因为表已经移动位置了.

然后

```
python3 manage.py runserver
```

就可以看到程序正常运行了.

好像看不出蓝本有啥用.....好吧。现在还看不出,下一篇就确实看的出来了,因为我们要把后台搭建好,然后在后台把博客文章提交上去,然后文章存到数据库里面,然后数据有了,在前端把数据显示出来.

你的笔记本上有很多接口: USB、电源接口、SD卡槽、耳机孔、HDMI (可插拔视图) 等等; 隔壁老王的电脑, 就一type-C口 (蓝图接口), 其他的接口只能通过type-C的扩展坞 (在蓝图中添加url规则) 再接到电脑上 (注册蓝图)。老王下班, 直接拔了那一根type-C走入 (取消注册蓝图), 而你要拔四五根线, 这时候你就发现了这根type-C的方便, 甚至当某个外接设备出问题 (业务逻辑需要修改) 时, 你只需要在外接设备与那个拓展坞 (蓝图中) 之间修复, 基本没你电脑 (主程序) 什么事, 因为它降低了其他外接设备与你电脑的耦合。

---

## 创建用户

---

源代码: <https://github.com/ltoddy/flask-tutorial>

技术交流群:630398887(欢迎一起吹牛)

写在前面的话: 如果你启动了项目,要去看本篇的内容,需要如下几个地址:

- localhost:5000/admin
- localhost:5000/admin/login
- localhost:5000/admin/register

还有就是,本篇我们数据库的设计改变了,所以在修改好app/models.py文件之后,要:

```
$ python3 manage.py shell
>>> from manage import *
>>> db
<SQLAlchemy engine=sqlite:////home/me/PycharmProjects/blog/data.sqlite>
>>> db.drop_all()
>>> db.create_all()
>>> exit()
```

如果你启动项目之后,页面给出了一个sqlalchemy相关的异常,看看你有没有执行上面的语句。

OK,这次要好好说明一下蓝图的好处.

一般我们网上的一些网站都是可以用户登录的,通过用户名(邮箱或者手机号)和密码。我们也不需要那么麻烦,只需要用户名和密码就够了。

我们需要更改一下 models.py 代码:

app/models.py

```
from . import db

class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    # 如果你学过数据库的话就知道我们一般通过id来作为主键,来找到对应的信息的,通过id来实
    现唯一性
    username = db.Column(db.String(64), unique=True)
    password = db.Column(db.String(64))

    def __repr__(self):
        return 'users表: id为:{}, name为:{}'.format(self.id, self.name)
```

其实就加上了 password 那一行.因为我们登录的时候需要帐号和密码这两样东西.

我们还需要处理一下我们的表单,毕竟我们需要注册和登录,都需要填写表单。

app/admin/forms.py

```
from flask.ext.wtf import FlaskForm
from wtforms import StringField
from wtforms import SubmitField
from wtforms import PasswordField
from wtforms.validators import Required
from wtforms.validators import EqualTo

class RegistrationForm(FlaskForm):
```

```

username = StringField('用户名:', validators=[Required()])
password = PasswordField('密码:', validators=[Required()])
password2 = PasswordField('确认密码', validators=[Required(),
EqualTo('password', message='两次密码不一致')])
# 再这里, 一般来说你注册的时候都要求你输入两次相同的密码的, 而且要求是一样的, 所以我们可以使用wtforms帮我们集成好的EqualTo方法.
# 那个message就是当你两次密码不一样的时候的提示信息
submit = SubmitField('确认,提交')

class LoginForm(FlaskForm):
    username = StringField('用户名:', validators=[Required()])
    password = PasswordField('密码:', validators=[Required()])
    submit = SubmitField('确认,提交')

```

关于这两个表单, 我曾经思考过:因为这两个表单中有相同的部分,我想提取出来:

```

class BaseForm(FlaskForm):
    username = StringField('用户名:', validators=[Required()])
    password = PasswordField('密码:', validators=[Required()])
    submit = SubmitField('确认,提交')

class RegisterForm(BaseForm):
    password2 = PasswordField('确认密码', validators=[Required(),
EqualTo('password', message='两次密码不一致')])

class LoginForm(BaseForm):
    pass

```

就像上面代码那样子,然而我否决了自己,这样子写大幅度的降低了可读性,而且,这个表单类,与HTML中的表单是一个一一对应的关系,所以最好不要提取一个父类来写.

写完这些,我们创建一下响应的蓝图:

app/admin/init.py

```

from flask import Blueprint

admin = Blueprint('admin', __name__)

from . import forms
from . import views

```

别忘了注册一下蓝图

app/init.py

```

from flask import Flask
from flask.ext.bootstrap import Bootstrap

```

```

from flask.ext.sqlalchemy import SQLAlchemy
from flask.ext.login import LoginManager

from config import Config

bootstrap = Bootstrap()
db = SQLAlchemy()
login_manager = LoginManager()
login_manager.session_protection = 'strong' # 这行代码就记住吧，我也不想解释了
login_manager.login_view = 'admin.login' # 设置登录界面，名为admin这个蓝本的
login视图函数

def create_app():
    app = Flask(__name__)
    app.config.from_object(Config)
    Config.init_app(app)

    bootstrap.init_app(app)
    db.init_app(app)
    login_manager.init_app(app)

    from .main import main as main_blueprint
    app.register_blueprint(main_blueprint)

    from .admin import admin as admin_blueprint
    app.register_blueprint(admin_blueprint, url_prefix='/admin')
    # 这里我们 url_prefix='/admin'，这个样子我们访问admin蓝本中的路由的时候，就需要
    加上前缀：admin
    return app

```

这里注意一下啊:

```

from flask.ext.login import LoginManager
login_manager = LoginManager()
login_manager.session_protection = 'strong' # 这行代码就记住吧，我也不想解释了
login_manager.login_view = 'admin.login' # 设置登录界面，名为admin这个蓝本的
login视图函数

```

我们需要有一个管理用户登录的东西—— LoginManager

还是要说一下蓝图，我们要做的这个blog，这个应用程序(app),他会有各个模块。举个例子，比如你去淘宝网页。你会看到：比如主页各种商品，用户登录注册，银行卡绑定等等一大堆的模块。如果像最开始的那个样子，路由和视图函数全部是由那个app实例来做的话，就得把app添加到各个地方去，很麻烦，也很混乱。

采用蓝本，就相当于每个模块有了自己构件路由和视图函数的工具，然后再把蓝本注册到app中，这样就方便多了。就比如，北方种植小麦，南方种植水稻，北方就专心种小麦，南方就专心种水稻。如果你都想吃，那么等他们收割了，集中在一块就好了，差不多这个意思。

OK，我们来设计一下我们的视图函数。考虑这么一件事情，比如你去知乎，如果你没有登录，它是要求你登录的。也就是说：假设你没有登录，那么你去主页，就会重定向到登录界面。

这里顺便说一下，我们一般会把index.html当作主页，这是默认的。比如我的技术博客，<http://algo.site> 和访问 <http://algo.site/index.php> 是一样的。

为了能够将你登录之后的状态记录下来，我们需要为我们的User模型添加点东西：

```
from . import db
from . import login_manager
from flask.ext.login import UserMixin

class User(db.Model, UserMixin):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    # 如果你学过数据库的话就知道我们一般通过id来作为主键，来找到对应的信息的，通过id来实
    现唯一性
    username = db.Column(db.String(64), unique=True)
    password = db.Column(db.String(64))

    def __repr__(self):
        return 'users表: id为:{}, name为:{}'.format(self.id, self.name)

    def check(self, password):
        return password == self.password

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
```

只是多继承了一个UserMixin类。他帮我们提供了一些功能：比如判断是否登录，比如获取你的主键(id)，为了唯一标识用户，这一点很重要，因为就像一个大型网站，肯定会多人同时在新，如何去确认你，这里非常重要，靠的就是这个id。

还需要加上一个回调函数，就是为了识别出你来的。

OK，最后是视图函数：

app/admin/views.py

```
from . import admin
from ..models import User
from .forms import LoginForm, RegistrationForm
from .. import db
from flask import render_template
from flask import redirect
from flask import url_for
from flask import flash
```



```

from flask.ext.login import current_user
from flask.ext.login import login_user
from flask.ext.login import logout_user
from flask.ext.login import login_required

@admin.route('/')
def index():
    if not current_user.is_authenticated:
        return redirect(url_for('admin.login'))
    return render_template('admin/index.html')

@admin.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()
        if user is not None:
            if user.check(form.password.data):
                login_user(user)
                return redirect(url_for('admin.index'))
            else:
                flash("用户名或者密码错误")
                return redirect(url_for('admin.login'))
        else:
            flash('没有你这个用户，请注册')
    return render_template('admin/login.html', form=form)

@admin.route('/logout')
@login_required
def logout():
    logout_user()
    flash('你已经退出了。')
    return redirect(url_for('admin.login'))

@admin.route('/register', methods=['GET', 'POST'])
def register():
    form = RegistrationForm()
    if form.validate_on_submit():
        try:
            user = User(username=form.username.data,
password=form.password.data)
            db.session.add(user)
            flash('注册成功')
            return redirect(url_for('admin.login'))
        except:

```

```
flash('帐号已存在')
return redirect(url_for('admin.register'))
return render_template('admin/register.html', form=form)
```

讲一下这些视图函数中的业务逻辑。

- 第一个index():

就是我们后台管理的那个页面，比如你把博客文章写好之后，在这里提交，然后在blog的主页显示出来。当然目前还没做好。顺便说一下,current\_user.is\_authenticated这个东西，是一个标志的变量，用来标志你有没有登录。如果登录的，就是True，没登录就是False。在这里，如果你没有登录，那么index这个页面你是看不到的，会强行重定向到登录页面

- 第二个login():

这个就是登录页面，当你提交表单之后，会去数据库查询，是否有你这个用户。我们这里有两层用于判断的if。

第一层判断的if，用来判断数据库里有没有你这个用户，如果没有你这个用户，会给你一个提示信息。

第二层判断的if，用来判断你密码是否正确，我们在User这个类中添加了check()这个方法。如果登录成功会跳转到后台的主页，没成功就提示密码错误。

- 第三个logout():

这个就是用来用户退出用的。他被@login\_required所保护着，什么意思呢？就是你如果没有登录的话，你去强行访问:localhost:5000/admin/logout这个页面的话，它会重定向到登录页面。这里我要说明一下，虽然代码写的是：return redirect(url\_for('admin.login'))这个，但是实际上阿，如果你没有登录访问了这个logout视图，它其实是根据login\_manager.login\_view = 'admin.login'你的这个设置来重定向的。

- 最后register(): 我们在数据库设计中阿，那个username字段设置的是unique，也就是唯一，用户名有且仅有一个，如果你创建的用户已经在数据库中存在了，那么添加到数据库会报一个异常，所以用try except语句来处理异常.没什么好说的.....

说点好玩的：你看阿，我们有了蓝图之后，从项目结构上看，一个蓝图对应一个文件夹，我们仅仅需要做的是，写好蓝图管辖下的路由就好，而不需要管其他蓝图的路由，这个样子，我们就把程序模块话了，每个模块各自有各自的功能，不会因为修改了这个模块而去影响其他模块，这个是非常好的一件事情，然后只需要最后的时候，把蓝本注册一下，把这些路由添加到应用中就好。

还有，

```
bootstrap.init_app(app)
db.init_app(app)
login_manager.init_app(app)
```

这些东西，我们把app这个实例当作参数传了进去，有个技术叫依赖注入(也叫控制反转)，当app传进去之后，那么与app相关的数据也就传了进去，在比如bootstrap中，比如

```
{% import 'bootstrap/wtf.html' as wtf %}
```

他是怎么凭借一个名字: 'bootstrap/wtf.html'来获得这个页面的, 如果你是用的pycharm来写的话, 软件会给你提示信息, 说找不到这个模板的。很简单, 我们把app给了bootstrap, 然后bootstrap中就有了这个app, 他识别了这个app, 所以就可以用bootstrap了。

---

## 后台管理与发布文章

---

源代码: <https://github.com/ltoddy/flask-tutorial>

技术交流群:630398887(欢迎一起吹牛)

写在前面的话: 如果你实在不会写页面,复制粘贴你会吧.

<https://getbootstrap.com/docs/3.3/examples/theme/>

这个页面是,bootstrap样式表的例样,

<http://getbootstrap.com/docs/4.0/examples/>

这个页面是,你进去看那个页面合适,你点进去,然后右键查看网页源代码,复制就好了,顺便说一下,别忘了把CSS也复制了.

我们来定义下我们的文章模型,文章内容放到数据库里面, 然后通过查询文章标题来, 在主页建立文章链接。

app/models.py

```
class Article(db.Model):
    __tablename__ = 'articles'
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(64))
    content = db.Column(db.Text)
```

文章模型很简单, 一个id, 一个文章标题, 一个文章的内容这三个字段。content = db.Column(db.Text),这里, 回顾一下或者百度一下SQLAlchemy的列模型。

因为我们文章内容可不是几个字, 而是很多上千上万字说不定, 所以就不能再用简单的db.String来处理了。我们使用db.Text这个, 他对长文本做了优化。

当我们有了这个数据库的文章模型之后, 为了给数据库添加数据, 那么我们还需要相对应的表单:

app/admin/forms.py

```
class PostForm(FlaskForm):
    title = StringField('文章标题:', validators=[Required()])
    content = TextAreaField('文章内容', validators=[Required()])
    submit = SubmitField('发布')
```

你看注意看变量的名字, 我们最可能的让表单和文章模型中的变量去相同的名字, 这样方便。

OK,继续。我们有了表单, 要在页面中呈现出来。

```

from ..models import Article

@admin.route('/', methods=['GET', 'POST'])
def index():
    form = PostForm()
    if not current_user.is_authenticated:
        return redirect(url_for('admin.login'))
    if form.validate_on_submit():
        try:
            article = Article(title=form.title.data,
                              content=form.content.data)
            db.session.add(article)
            form.title.data = ''
            form.content.data = ''
            flash('发布成功')
        except:
            flash('文章标题有重复')
    return render_template('admin/index.html', form=form)

```

这段代码也没什么难度，先去创建一个表单form，然后判断一下你是否登录了，没登录的话就会重定向到登录界面。继续往下，让你提交表单的时候，这里用到了一个try - catch语句，其实这里原因是，我最初想把那个数据库文章模型中title字段设置成unique的，title = db.Column(db.String(64),unique=True),后来想想算了，所不定有相同标题的文章呢。然后就是从表单中获取数据,来构件新的文章，然后存到数据库里面。flash就是用来做一个提示，方便你自己知道你都干了啥事。

OK，把我们的HTML页面也说一下。

```

{% extends 'admin/base.html' %}

{% import 'bootstrap/wtf.html' as wtf %}

{% block navbar %}
    <nav class="navbar navbar-inverse">
        <div class="container-fluid">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-
toggle="collapse" data-target=".navbar-collapse">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a class="navbar-brand" href="/">Just for fun</a>
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">

```

```

        <li><a href="/">Home</a></li>
    </ul>
    <ul class="nav navbar-nav navbar-right">
        <li><a href="#" data-toggle="dropdown" class="dropdown-
toggle">当前用户名为:{{ current_user.username }} <b
class="caret"></b></a>
        <ul class="dropdown-menu">
            <li><a href="{{ url_for('admin.logout') }}">退
出当前账户</a></li>
        </ul>
    </li>
    </ul>
</div>
</div>
</nav>
{%endblock%}

{% block page_content %}
    <div class="page-header">
        {{ wtf.quick_form(form) }}
    </div>
{%endblock%}

```

唯一做出了更改的地方就是 `{% block page_content %}` 这里，里面加入了一个表单. OK,启动一下我们的项目看看效果。

The screenshot shows a web browser at `localhost:5000/admin/`. The page has a dark header with 'Just for fun' and 'Home' on the left, and '当前用户名为:toddy' on the right. The main content area contains a form with two input fields. The first field, labeled '文章标题:', contains the text 'Hello'. The second field, labeled '文章内容', contains the text 'Hello World'. Below these fields is a button labeled '发布'.

当然你可能会出现一个SQLAlchemy的异常，如何解决：

```
$ python3 manage.py shell
>>> from manage import *
>>> db.drop_all()
>>> db.create_all()
>>> exit()
```

更新一下我们的数据库结构，毕竟我们更改了数据库模型，添加了文章这个模型。注意哦，每当我们更改了app/models.py这个文件,我们最好都要重新设置一下数据库。

我们现在已经可以把文章放到数据库了，现在我们要把数据库的文章显示出来。在这里，我修改了一下templates/base.html页面：

```
{% extends 'bootstrap/base.html' %}

{% block title %}Just for fun{% endblock %}

{% block navbar %}
    <nav class="navbar navbar-inverse" id="top">
        <div class="container-fluid">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-
toggle="collapse" data-target=".navbar-collapse">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a class="navbar-brand" href="/">Just for fun</a>
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li><a href="/">Home</a></li>
                </ul>
            </div>
        </div>
    </nav>
{% endblock %}

{% block content %}
    {% for message in get_flashed_messages() %}
        <div class="alert alert-info">
            <button type="button" class="close" data-
dismiss="alert">&times;</button>
            {{ message }}
        </div>
    {% endfor %}
    <div class="container">
        <div class="col-sm-8">
            {% block page_content %}{% endblock %}
        </div>
    </div>
{% endblock %}
```

```

        </div>
        <div class="col-sm-3 col-sm-offset-1">
            {% block slider %}{% endblock %}
        </div>
    </div>
{%endblock%}

```

只更改了最后，我把

，切成了两个块，第一个块来放文章的内容，第二个块来放文章目录。

改完templates/base.html，我们在改一下templates/index.html，这是我们的主页，总显示一句话不好看，所以我们改一改。

```

{% extends 'base.html' %}

{% import 'bootstrap/wtf.html' as wtf %}

{% block page_content %}
    <h1>欢迎来到我的blog</h1>
    <p>技术交流群：630398887</p>
    <hr>
    <div class="row">
        <h2>如下内容凑的字数</h2>
        <hr>
        <h3>登鹳雀楼</h3>
        <p>白日依山尽，黄河入海流。</p>
        <p>欲穷千里目，更上一层楼。</p>
        <hr>
        <h2>论语</h2>
        <p>子谓公冶长：“可妻也，虽在缁纆之中，非其罪也！”以其子妻之。</p>
        <p>子谓南容：“邦有道不废；邦无道免于刑戮。”以其兄之子妻之。</p>
        <p>子谓子贱：“君子哉若人！鲁无君子者，斯焉取斯？”</p>
        <p>子贡问曰：“赐也何如？”子曰：“女器也。”曰：“何器也？”曰：“瑚璉也。”</p>
        <p>或曰：“雍也仁而不佞。”子曰：“焉用佞？御人以口给，屡憎于人。不知其仁，焉用佞？”</p>
        <p>子使漆雕开仕，对曰：“吾斯之未能信。”子说。</p>
        <p>子曰：“道不行，乘桴浮于海，从我者其由与？”子路闻之喜，子曰：“由也好勇过我，无所取材。”</p>
        <p>
            孟武伯问：“子路仁乎？”子曰：“不知也。”又问，子曰：“由也，千乘之国，可使治其赋也，不知其仁也。”“求也何如？”子曰：“求也，千室之邑、百乘之家，可使为之宰也，不知其仁也。”“赤也何如？”子曰：“赤也，束带立于朝，可使与宾客言也，不知其仁也。”</p>
        <p>子谓子贡曰：“女与回也孰愈？”对曰：“赐也何敢望回？回也闻一以知十，赐也闻一以知二。”子曰：“弗如也，吾与女弗如也！”</p>
        <p>宰予昼寝，子曰：“朽木不可雕也，粪土之墙不可朽也，于予与何诛？”子曰：“始吾于人也，听其言而信其行；今吾于人也，听其言而观其行。于予与改是。”</p>
        <p>子曰：“吾未见刚者。”或对曰：“申枋。”子曰：“枋也欲，焉得刚。”</p>
    </div>

```

```

        <p> 子贡曰：“我不欲人之加诸我也，吾亦欲无加诸人。”子曰：“赐也，非尔所及也。”</p>
        <p> 子贡曰：“夫子之文章，可得而闻也；夫子之言性与天道，不可得而闻也。”</p>
        <p> 子路有闻，未之能行，唯恐有闻。</p>
        <p> 子贡问曰：“孔文子何以谓之文也？”子曰：“敏而好学，不耻下问，是以谓之文也。”</p>
        <p> 子谓子产：“有君子之道四焉：其行己也恭，其事上也敬，其养民也惠，其使民也义。”</p>
        <p> 子曰：“晏平仲善与人交，久而敬之。”</p>
        <p>
            子张问曰：“令尹子文三仕为令尹，无喜色，三已之无愠色，旧令尹之政必以告新令尹，何如？”子曰：“忠矣。”曰：“仁矣乎？”曰：“未知，焉得仁？”“崔子弑齐君，陈文子有马十乘，弃而违之。至于他邦，则曰：‘犹吾大夫崔子也。’违之。之一邦，则又曰：‘犹吾大夫崔子也。’违之，何如？”子曰：“清矣。”曰：“仁矣乎？”曰：“未知，焉得仁？”</p>
    </div>
{%endblock%}

{% block slider %}
    <ol class="list-unstyled">
        <li><h2>文章列表</h2></li>
        {% for article in articles %}
            <li><a href="article/{{ article.title }}">{{ article.title }}</a></li>
        {% endfor %}
    </ol>
{%endblock%}

```

当然，大部分内容去百度复制粘贴的，不过这里你要看一下这一段代码。

```

{% block slider %}
    <ol class="list-unstyled">
        <li><h2>文章列表</h2></li>
        {% for article in articles %}
            <li><a href="article/{{ article.title }}">{{ article.title }}</a></li>
        {% endfor %}
    </ol>
{%endblock%}

```

我们利用一个for循环，通过视图函数传过来的参数，做了一个文章列表，效果如下：



# 文章列表

钗头凤·红酥手

武陵春·春晚

水龙吟·登建康赏心亭

为了把效果实现出来，我们需要在视图函数中把参数传进来，然后让Jinja2引擎渲染一下。还记得之前提到Jinja2引擎的时候，它可以想在python里面一样可以解析复杂的数据类型，比如咱这个article。

```
from ..models import Article

@main.route('/', methods=['GET', 'POST'])
def index():
    articles = Article.query.all()
    return render_template('index.html', articles=articles)
```

也很简单拉，把所有文章从数据库获取一下，传进去就好了。

能显示文章列表了，我们需要单独把文章显示出来。

```
<li><a href="admin/article/{{ article.title }}">{{ article.title }}</a>
</li>
```

其实阿，这段代码，是我已经把代码都写好了的，因为这个时候你可能还没有把相关的html页面写好。所以说那个a标签中的href属性可以先不写，然后等会在写就好。

app/templates/article.html

```
{% extends 'base.html' %}

{% block page_content %}
    <div class="page-header"><h1>{{ title }}</h1></div>
    <div class="well">{{ content }}</div>
{% endblock %}
```

文章的页面也很简单，一个标题，一个内容。两个变量。

再看一下视图函数：

```
@main.route('/article/<title>')
def article(title):
    article = Article.query.filter_by(title=title).first()
    return render_template('article.html', title=title,
                           content=article.content)
```

你看这里，还记得动态路由么，这里用到了动态路由，通过文章的标题来从数据库把这篇文章数据获取出来。然后把标题和文章内容传参到HTML页面就可以了。

这个时候，你就可以把app/templates/index.html页面中那个文章列表的a标签的href属性补全了。

这里说一下，你看啊，我们的功能在不断的增加，可是基本上都没有大幅度改动我们的原先的代码，而是为新功能写好代码，添加进去。这是一个非常好的表现，增加新功能而又不与原先代码有冲突。



---

## 难说再见

---

源代码: <https://github.com/ltoddy/flask-tutorial>

技术交流群:630398887(欢迎一起吹牛)

写在前面的话：如果要运行这次的代码，请先：

```
$ python3 manage.py shell
>>> from manage.py import *
>>> db.drop_all()
>>> db.create_all()
>>> exit()
```

因为我已经注册了用户了。

到目前为止，blog还是有很多问题的：

- 数据库建模方面，没有使用外键，也就是说没有一对多或者多对一的关系。也就是说，对于目前的blog，如果你注册了多个用户，那么这些用户对于所有文章是共用的。而不是各自用户有各自的文章——如何解决，当然你也可以使用外键来链接两个表实现，或者我们就只允许一个用户(管理员)的存在，私人使用。
- 我们的后台功能太欠缺。目前只有一个发布功能，其实还需要：
  - 文章的修改
  - 文章的删除
- 我们没有设置文章的类型，所以目前无法给文章分类，其实也很好解决，在app/models.py的文章模型中加入一个type字段就好。

第一个问题，简单的解决方法:在config.py中加一个全局变量(bool类型)，初始值为False，表示目前没有用户注册，当第一个用户注册之后，这个变量设为True，表示已经有一个用户注册了，那么就可以不再让第二个用户注册了。

config.py

```

import os

basedir = os.path.abspath(os.path.dirname(__file__))

is_exist_admin = False

class Config:
    SECRET_KEY = 'a string'
    # 数据库配置
    SQLALCHEMY_DATABASE_URI = 'sqlite:/// ' + os.path.join(basedir,
'data.sqlite')
    SQLALCHEMY_COMMIT_ON_TEARDOWN = True
    SQLALCHEMY_TRACK_MODIFICATIONS = False

    @staticmethod
    def init_app(app):
        pass

```

多了这个: is\_exist\_admin = False

```

from config import is_exist_admin

@admin.route('/register', methods=['GET', 'POST'])
def register():
    global is_exist_admin
    form = RegistrationForm()
    if form.validate_on_submit() and not is_exist_admin:
        try:
            user = User(username=form.username.data,
password=form.password.data)
            db.session.add(user)
            is_exist_admin = True
            flash('注册成功')
            return redirect(url_for('admin.login'))
        except:
            flash('帐号已存在')
            return redirect(url_for('admin.register'))
    else:
        flash('管理员已存在')
    return render_template('admin/register.html', form=form)

```

第二个问题:

先解决添加文章的修改功能, 我们可不可以这样子, 把发布文章和更新文章视作同一种操作。我们将发布和更新文章这两个功能合二为一, 怎么说呢。当你去发布文章的时候, 根据标题去查询数据库有没有这篇文章, 如果没有这篇文章, 那么就在数据库中添加这个信息, 如果有这篇文章, 那么就去更改数据库这条信息的内容。

```

@admin.route('/', methods=['GET', 'POST'])
def index():
    form = PostForm()
    if not current_user.is_authenticated:
        return redirect(url_for('admin.login'))
    if form.validate_on_submit():
        article = Article(title=form.title.data, content=form.content.data)
        if Article.query.filter_by(title=form.title.data).first() is None:
            # 文章不存在
            db.session.add(article)
            flash('发布成功')
        else: # 文章已存在
            article =
Article.query.filter_by(title=form.title.data).first()
            article.content = form.content.data
            db.session.add(article)
            # db.session.commit()
            flash('文章更新成功')
            form.title.data = ''
            form.content.data = ''
        return render_template('admin/index.html', form=form)

```

最后是删除文章：这里我就不写了，我相信你可以自己做到。

## 说点额外话:

一个网站构成的,

- 用户看到的界面（前端）
- 后端表单验证+数据库

前端从来不是问题，因为毕竟你去百度可以搜索到各种各样炫酷的模板。后端，这是核心。数据的处理在这里。

# 使用Flasky-SQLAlchemy 管理数据库

本文主要解决那本《Flask Web开发 基于Python的Web应用开发实战》这本书坑不少，书是挺好的，但是你会踩不少坑，导致你会有很多bug，即使你复制的源代码。

先来一段代码

```

from flask import Flask
from flask.ext.sqlalchemy import SQLAlchemy
import os

```

```

basedir = os.path.abspath(os.path.dirname(__file__))

app = Flask(__name__)

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:/// ' + \
    os.path.join(basedir, 'data.sqlite')
app.config['SQLALCHEMY_COMMIT_TEARDOWN'] = True
#app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

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

if __name__ == '__main__':
    app.run()

```

注意，我在app.config中有一行是打了注释的。如果加了注释：效果如下：



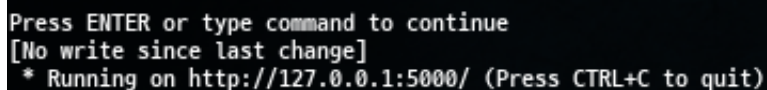
```

[No write since last change]
/usr/local/lib/python3.5/dist-packages/flask_sqlalchemy/__init__.py:839: FSADeprecationWarning: SQLALCHEMY_TRACK_MODIFICATIONS adds signifi
suppress this warning.
'SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and '
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

```

它会显示一个warning。一个警告

如果那一行去掉注释，效果如下：



```

Press ENTER or type command to continue
[No write since last change]
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

```

这样子警告就消失了。

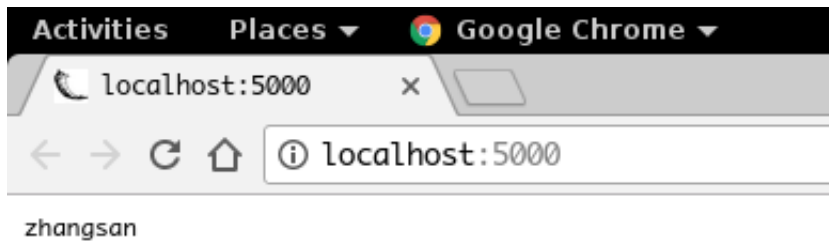
接下来，为数据库建个表，简单起见，表中就两个字段，一个id，一个username，

```

class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64))

@app.route('/')
def index():
    u = User(username='zhangsan')
    return u.username

```



看上去还可以。再换个方式。真正把用户数据(就是那个变量u)添加到数据库去.

```
from flask import Flask
from flask.ext.sqlalchemy import SQLAlchemy
import os

basedir = os.path.abspath(os.path.dirname(__file__))

app = Flask(__name__)

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:/// ' + \
    os.path.join(basedir, 'data.sqlite')
app.config['SQLALCHEMY_COMMIT_TEARDOWN'] = True
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64))

@app.route('/')
def index():
    u = User(username='zhangsan')
    db.session.add(u)
    return User.query.filter_by(username='zhangsan').first().username

if __name__ == '__main__':
    # db.drop_all()
```

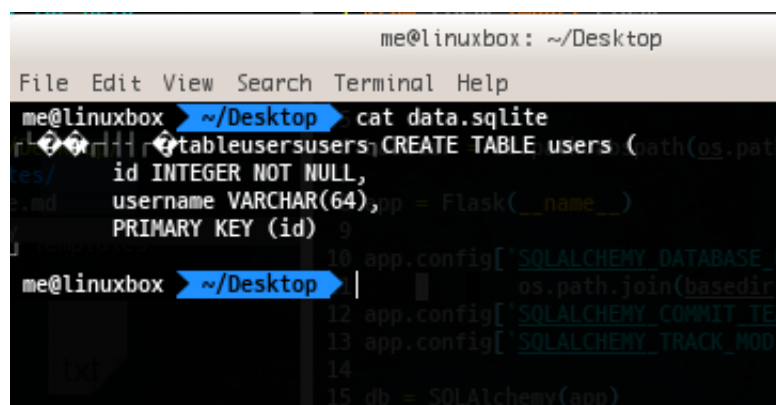
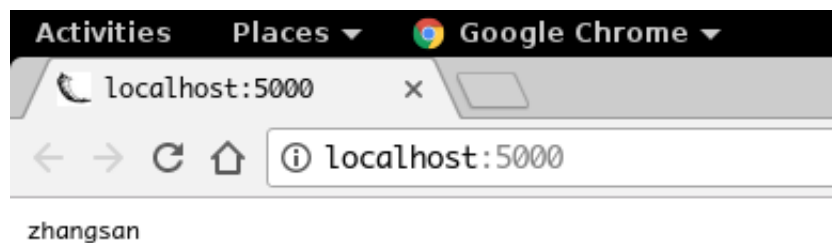
```
# db.create_all()  
app.run()
```

注意看最下面，注释的那两行，如果注释掉的话，你去运行，会出现一个500的错误



这是你仔细观察目录中的文件，多了一个名字叫:data.sqlite的文件，然后你去查看这个文件的内容，你会发现里面什么都没有。原因很简单，因为数据库的表你根本就还没有创建呢，所以在最下面if里面的两行，是为了创建表用的。估计大多数人被这个坑了。

当我们把注释解除之后。



这个样子就有东西了。

所以总的来说，当你去按照那个书学习的时候，如果打开localhost:5000之后出现SQLAlchemy的Error的时候，先去看看有没有那个.sqlite文件(一般都有),然后去看看里面有没有东西，如果没有请按照如下方式：

这里假设你的db变量所在文件为:manage.py

```
python3 manage.py shell
```

```
>>>from manage import db
>>>db.drop_all()
>>>db.create_all()
```

## 使用Flask-Mail提供电子邮件支持

使用pip安装Flask-Mail

```
pip install flask-mail
```

这里使用163的邮箱作为发送者。注册163邮箱之后，去设置 -> POP3/SMTP/IMAP(在右侧导航栏)，然后开启你的SMTP服务，这时候会让你设置客户端授权码，这个授权码是重点，一定要记住。

类型	服务器名称	服务器地址	SSL协议端口号	非SSL协议端口号
发件服务器	SMTP	smtp.163.com	465/994	25

把这个表格也关注一下，里面的内容要去写到配置中去。

源码:

```
from flask import Flask
from flask.ext.mail import Mail, Message

app = Flask(__name__)
# 下面是SMTP服务器配置
app.config['MAIL_SERVER'] = 'smtp.163.com' # 电子邮件服务器的主机名或IP地址
app.config['MAIL_PORT'] = 25 # 电子邮件服务器的端口
app.config['MAIL_USE_TLS'] = True # 启用传输层安全
# 注意这里启用的是TLS协议(transport layer security)，而不是SSL协议所以用的是25号端口
app.config['MAIL_USERNAME'] = 'username@163.com' # 你的邮件账户用户名
app.config['MAIL_PASSWORD'] = 'password' # 邮件账户的密码,这个密码是指的授权码!
授权码!授权码!

mail = Mail(app)

@app.route('/')
def index():
    msg = Message('你好', sender='username@163.com', recipients=
['you@example.com'])
    # 这里的sender是发信人，写上你发信人的名字，比如张三。
    # recipients是收信人，用一个列表去表示。
```



```
msg.body = '你好'
msg.html = '<b>你好</b> stranger'
mail.send(msg)
return '<h1>邮件发送成功</h1>'

if __name__ == '__main__':
    app.run(debug=True)
```

值得注意的一点是，如果你是刚刚创建的163的邮箱，你最好先用163邮箱发送一封邮件，因为你在发送第一封邮件的时候，会让你设置发件人名字，如果不设置的话，你的邮件会被退回。