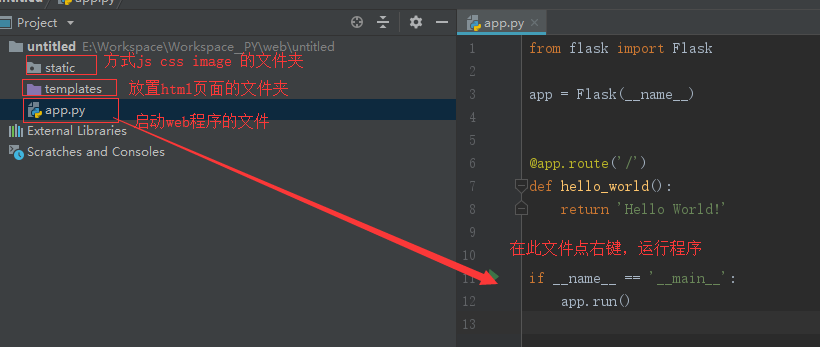
**Flask Web框架**

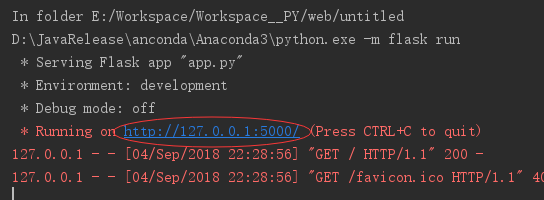
# 框架概述

描述：Flask是一个使用 Python 编写的轻量级 Web 应用框架。其 WSGI 工具箱采用 Werkzeug ，模板引擎则使用 Jinja2 。Werkzeug是一个socket服务端程序。通过发给flask的请求，其实底层由Werkzeug接受，然后出发Flask框架。

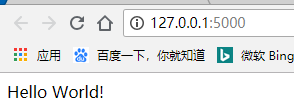
默认项目结构：



默认本机+5000端口：



点击链接：



说在前面：当修改了代码不生效，请到资源管理器，关闭所有python进程

# 框架安装

## 在conda中安装

conda install flask

## 源码安装

到flask源代码 setup.py目录使用命令：

python setup.py install

## pip安装

pip install flask

# 基本使用

在pycharm 创建flask项目后，发开app.py文件，看到如下内容：

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def hello\_world():

return 'Hello World!'

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

这里，创建flask实例，然后运行，即可通过浏览器，访问路由，获取对应的返回值。

## [实例化Flask对象时，可选的参数](https://www.cnblogs.com/huchong/p/8227606.html#_lab2_1_0)

app = Flask (\_\_name\_\_)：Flask有很多参数，重要参数说明：

static\_url\_path=None,静态文件的地址前缀，写成【test】，访问静态文件时，就要在对应的url里面加上【test】

static\_folder='static',：静态文件所在文件的名字，默认是static，可以不用填

template\_folder='templates',：模板所在文件夹的名字

root\_path=None 可以不用填，会自动找到，当前执行文件，所在目录地址

后面两个参数的作用：

在return render\_template时会将上面两个进行拼接，找到对应的模板地址

不常用：

instance\_path和instance\_relative\_config：手动导入配置文件会用到。先做说明：

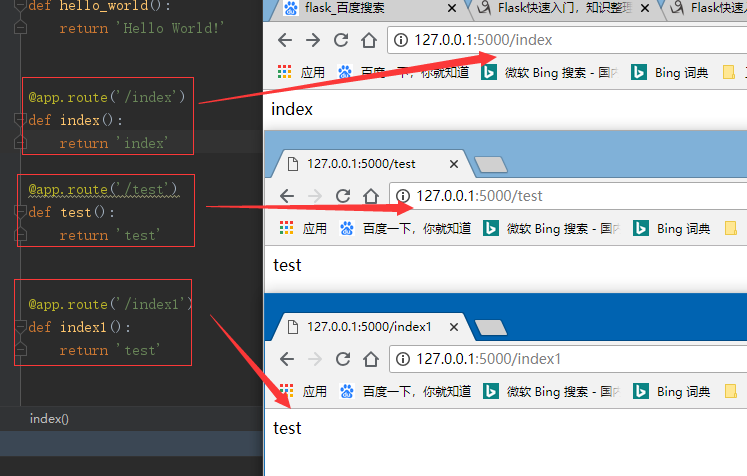
instance\_relative\_config：默认为False，当设置为True时，from\_pyfile会从instance\_path指定的地址下查找文件。

instsnce\_path：指定from\_pyfile查询文件的路径，不设置时，默认寻找和app.run()的执行文件同级目录下的instance文件夹；如果配置了instance\_path（注意需要是绝对路径），就会从指定的地址下里面的文件

## [绑定路由关系的方式](https://www.cnblogs.com/huchong/p/8227606.html#_label3_1_0_0)

方式1：在视图函数上添加路由规则

通过路由，返回具体的字符串



方式2：通过app对象添加到路由表

app.add\_url\_rule(rule='/index/', endpoint="test", view\_func=index, methods=["GET", "POST"])

## 渲染模板与装饰器

from flask import Flask, render\_template, request, redirect, session

app = Flask(\_\_name\_\_)

app.secret\_key = "fiuahfouaifh" # 设置sessionkey

def wrapper(func):

def inner(\*args, \*\*kwargs):

if not session.get("user\_info"):

return redirect("/login")

ret = func(\*args, \*\*kwargs)

return ret

return inner

@app.route("/login", methods=["GET", "POST"]) # 默认接受请求方式GET

def login():

if request.method == "GET":

return render\_template("login.html")

else:

# print(request.values) #request.values包含body所有的值

name = request.form.get("name")

pwd = request.form.get("pwd")

if name == "test" and pwd == "123":

session["user\_info"] = name

# session.pop("user\_info") #删除session

return redirect("/index")

else:

return render\_template("login.html", msg="用户名或者密码错误")

@app.route("/index", methods=["GET", "POST"])

@wrapper # 自己定义装饰器时，必须放在路由的装饰器下面

def index():

return render\_template("index.html") # 渲染模板

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

## 请求和响应

request

request.form #POST请求的数据

request.args #GET请求的数据，通过.to\_dict可以转换成字典

request.querystring #GET请求，string形式

response

return render\_tempalte() #返回模板

return redirect()#重定向到另一个url

return "" #直接返回字符串

v = make\_response(返回值) #可以把返回的值包在了这个函数里面，然后再通过.set\_cookie绑定cookie等

session

存在浏览器上，并且是加密的

依赖于：secret\_key

## [flask中获取URL后面的参数](https://www.cnblogs.com/huchong/p/8227606.html#_label3_1_2_1)

from flask import Flask, request

from urllib.parse import urlencode

app = Flask(\_\_name\_\_)

@app.route("/index")

def index():

if request.method == 'GET':

a = request.args

b = request.args.to\_dict() # 参数转字典

c = urlencode(a) # 中文解码

d = urlencode(b)

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

# 配置文件

类似javaweb开发，里面的spring mvc

修改settings.py：

class BaseConfig(object):

NNN = 123 # 注意是大写

SESSION\_COOKIE\_NAME = "session\_sss"

class TestConfig(BaseConfig):

DB = "127.0.0.1"

class DevConfig(BaseConfig):

DB = "52.5.7.5"

class ProConfig(BaseConfig):

DB = "55.4.22.4"

使用配置：

from flask import Flask, current\_app

app = Flask(\_\_name\_\_)

app.config.from\_object('settings.DevConfig')

@app.route("/index")

def index():

conf = current\_app.config

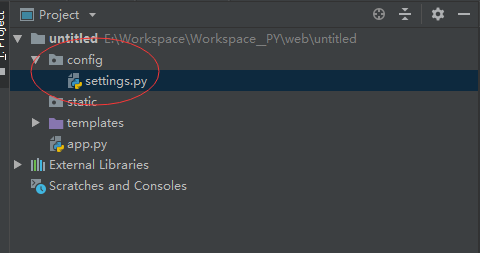
for k in conf:

print('key:', k, '\t', 'val:', conf[k])

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

settings.py文件默认路径要放在程序root\_path目录，如果instance\_relative\_config为True，则就是instance\_path目录，那么该文件需要放到：



修改app.py，指定配置文件路径：

from flask import Flask, current\_app

app = Flask(\_\_name\_\_)

app.config.from\_object('config.settings.DevConfig')

@app.route("/index")

def index():

conf = current\_app.config

for k in conf:

print('key:', k, '\t', 'val:', conf[k])

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

# 路由规则

查看实例：

红色字体这两个参数，是在实际项目中使用最多的

from flask import Flask, current\_app, render\_template

app = Flask(\_\_name\_\_)

app.config.from\_object('config.settings.DevConfig')

@app.route("/index1", methods=['GET', 'POST'], redirect\_to='/index2')

def index1():

conf = current\_app.config

for k in conf:

print('key:', k, '\t', 'val:', conf[k])

@app.route("/index2", methods=["GET", "POST"])

def index2():

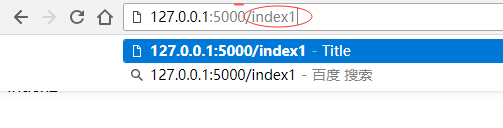
# if not session.get("user\_info"):

# return redirect("/login")

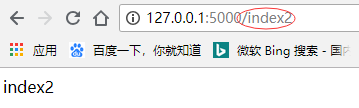
return render\_template("index2.html")

if \_\_name\_\_ == '\_\_main\_\_':

app.run()



自动跳转到：

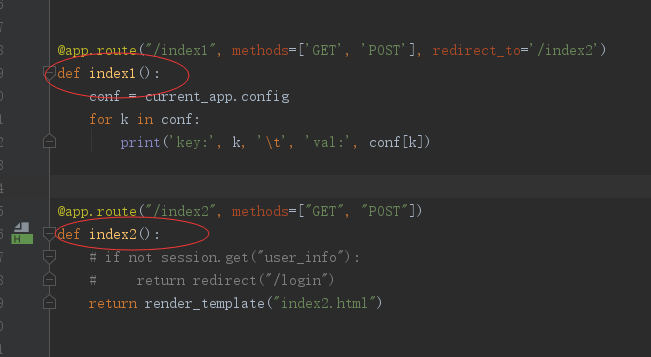


# 视图函数

## [类视图及其使用](https://www.cnblogs.com/huchong/p/8227606.html#_label3_4_1_0)

红圈里面的函数，都叫做视图函数。但是，视图函数不能面向对象编程。因为会导致路由系统找不到对应的关联函数。但是，可以使用 类视图解决

视图函数：



类视图函数:

注意：类视图函数只能是通过app对象添加路由表的方式绑定

from flask import Flask

from flask.views import View

app = Flask(\_\_name\_\_)

class MyView(View): # MyView继承于View

def test(self): # 自定义的方法

return '测试类视图'

def dispatch\_request(self): # 必须重写这个方法

resp = self.test()

return resp

app.add\_url\_rule('/test1/', view\_func=MyView.as\_view('test')) # MyView.as\_view('test') 返回的是一个方法

app.add\_url\_rule(rule='/test2/', view\_func=MyView.as\_view('test'), endpoint="test2", methods=["GET", "POST"])

print(app.url\_map)

if \_\_name\_\_ == '\_\_main\_\_':

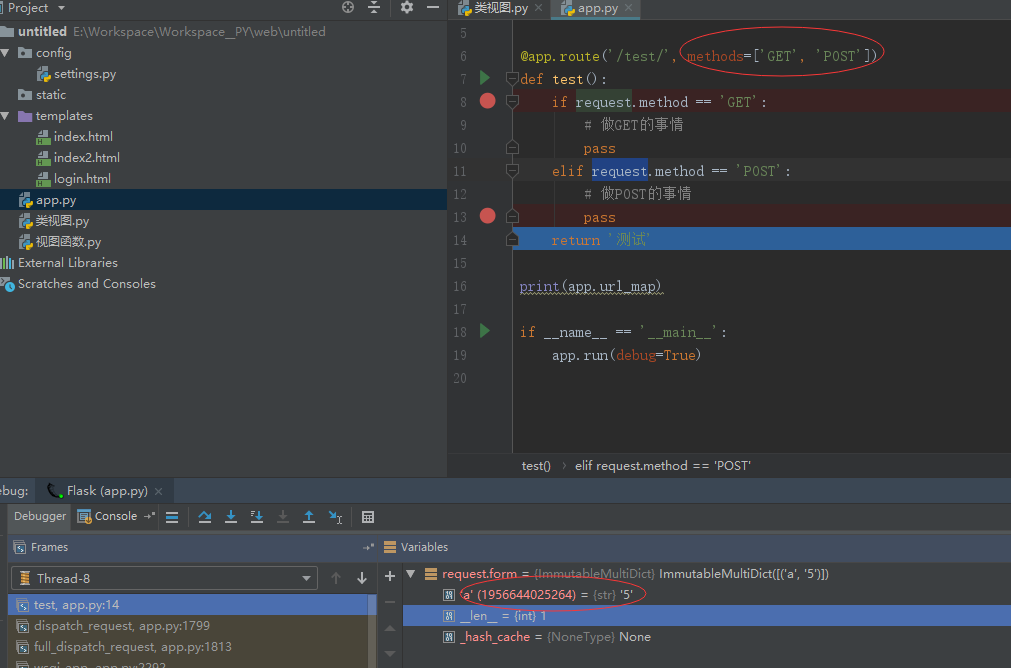
app.run(debug=True)

内部原理：

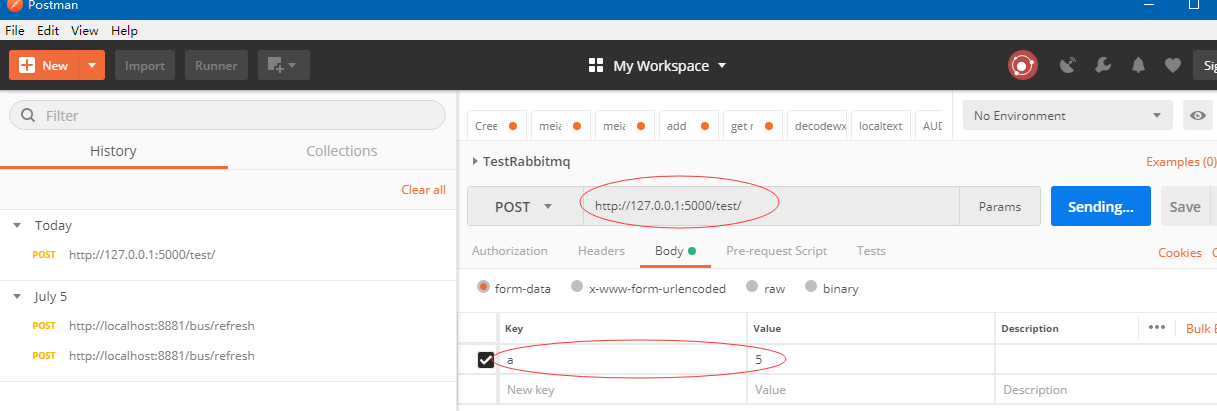
as\_view是一个装饰器，返回的是一个函数，在函数内部会调用dispatch\_request。通过dispatch\_request调用具体的类的函数。

## 根据视图函数协议执行逻辑

在视图函数中，通过协议类型区分如何执行下一步逻辑：



POST协议推荐测试方法：使用postman工具



## 导入方法视图类

方法视图类中的函数就是自动根据协议来进行处理

class TestMethodView(MethodView):

def get(self):

# 处理Get请求

return 'GET请求'

def post(self):

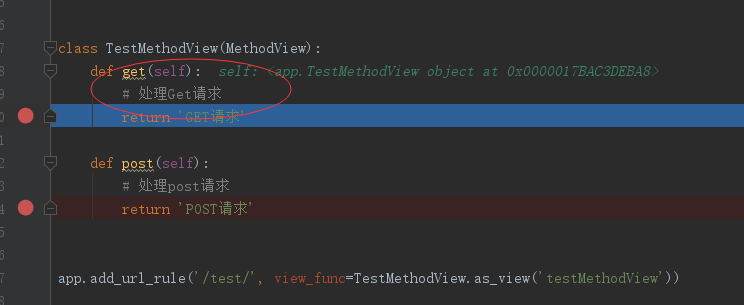
# 处理post请求

return 'POST请求'

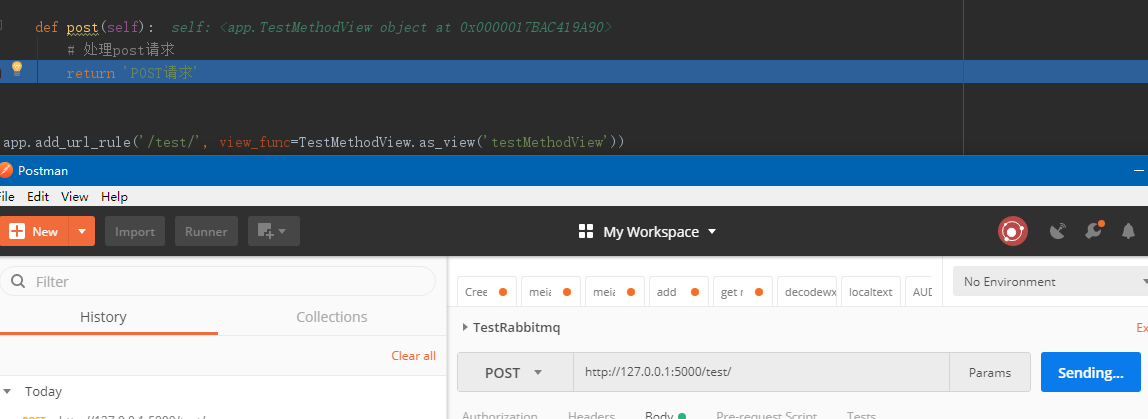
绑定路由：

app.add\_url\_rule('/test/', view\_func=TestMethodView.as\_view('testMethodView'))

get：通过浏览器自动发送



post：通过postman发送



上面绑定的方式，使一个url处理了两种情况。flask提供了一种机制，是不同的url处理不同的情况，但视图函数还是同一个

f1 = TestMethodView.as\_view('testMethodView02');

app.add\_url\_rule('/test1/', view\_func=f1, methods=['GET'])

f2 = TestMethodView.as\_view('testMethodView03')

app.add\_url\_rule('/test2/', view\_func=f2, methods=['POST'])

# 模板语法

## [模板的使用](https://www.cnblogs.com/huchong/p/8227606.html#_lab2_6_0)

后台传递参数：

from flask import Flask, url\_for, render\_template, Markup

app = Flask(\_\_name\_\_)

def test1(a, b): # 自定义的标签，此方法在使用时，需要在render\_temlate中传入到指定以页面使用

return a + b

@app.template\_global() # 不需要传入，可直接在页面使用

def test2(a1, a2):

return a1 + a2 + 100

@app.route('/index')

def index():

v1 = "这个一个文本"

v2 = [1, 2, 3]

v3 = {"k1": "test1", "k2": "test2"}

v4 = Markup("<input type='text' />") # 推荐使用

return render\_template("index.html", v1=v1, v2=v2, v3=v3, v4=v4, test1=test1)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

前台通过模板进行处理：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

</head>

<body>

<div>显示字符串：{{ v1 }}</div>

<div>循环列表：

{% for foo in v2 %}

<li>{{ foo }}</li>

{% endfor %}

</div>

<div>

通过索引获取列表值：v2.2

</div>

<div>

遍历字典：

{% for k,v in v3.items() %}

<li>{{ k }} {{ v }}</li>

{% endfor %}

{{ v3.k1 }}

{{ v3.get("k1") }}

</div>

<div>

输出html标签：

{{ v4 }}

</div>

<div>

调用后台函数：

<h1>{{ test1(1,2) }}</h1>

<h1>{{ test2(1,2) }}</h1>

</div>

</body>

</html>

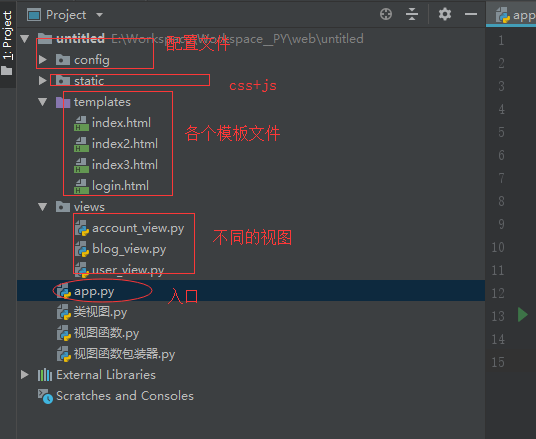
## [静态文件的两种导入方式](https://www.cnblogs.com/huchong/p/8227606.html#_label3_6_0_1)

<link rel="stylesheet" href="{{ url\_for('static',filename='bootstrap.css') }}">

<link rel="stylesheet" href="/static/bootstrap-theme.css">

# 项目结构划分

在实际开发项目的时候，项目代码是需要按模块划分，分别放入不同的文件和py文件中，接下来，搭建一个项目框架。



使用蓝图划分url模块：

from flask import Flask

from views.account\_view import account

from views.blog\_view import blog

from views.user\_view import user

app = Flask(\_\_name\_\_)

app.register\_blueprint(account, url\_prefix="/account")

app.register\_blueprint(blog, url\_prefix="/blog")

app.register\_blueprint(user, url\_prefix="/user")

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

# 数据库

## 编写配置

from flask import Flask

from flask\_sqlalchemy import SQLAlchemy

app = Flask(\_\_name\_\_)

app = Flask(\_\_name\_\_)

# url的格式为：数据库的协议：//用户名：密码@ip地址：端口号（默认可以不写）/数据库名

app.config["SQLALCHEMY\_DATABASE\_URI"] = "mysql://root:root@localhost/mytest"

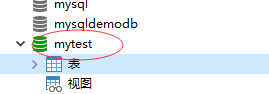
# 可移除

app.config["SQLALCHEMY\_TRACK\_MODIFICATIONS"] = False

# 创建数据库的操作对象

db = SQLAlchemy(app)

## 创建数据库



## 创建表模型

from config import db

class Role(db.Model):

\_\_tablename\_\_ = "roles"

id = db.Column(db.Integer, primary\_key=True)

name = db.Column(db.String(16), unique=True)

# 给Role类创建一个uses属性，关联users表。

# backref是反向的给User类创建一个role属性，关联roles表。这是flask特殊的属性。

users = db.relationship('User', backref="role")

# 相当于\_\_str\_\_方法。

def \_\_repr\_\_(self):

return "Role: %s %s" % (self.id, self.name)

## 操作数据库

#!/usr/bin/env python

# -\*- coding:utf-8 -\*-

from flask import Blueprint, render\_template

from config import db

from dbmodels.models import Role, User

user = Blueprint('user', \_\_name\_\_)

@user.route('/index', methods=['GET', "POST"])

def index():

# 删除所有的表

db.drop\_all()

# 创建表

db.create\_all()

ro1 = Role(name="admin")

# 先将ro1对象添加到会话中，可以回滚。

db.session.add(ro1)

ro2 = Role()

ro2.name = 'user'

db.session.add(ro2)

# 最后插入完数据一定要提交

db.session.commit()

us1 = User(name='wang', email='wang@163.com', password='123456', role\_id=ro1.id)

us2 = User(name='zhang', email='zhang@189.com', password='201512', role\_id=ro2.id)

us3 = User(name='chen', email='chen@126.com', password='987654', role\_id=ro2.id)

us4 = User(name='zhou', email='zhou@163.com', password='456789', role\_id=ro1.id)

us5 = User(name='tang', email='tang@itheima.com', password='158104', role\_id=ro2.id)

us6 = User(name='wu', email='wu@gmail.com', password='5623514', role\_id=ro2.id)

us7 = User(name='qian', email='qian@gmail.com', password='1543567', role\_id=ro1.id)

us8 = User(name='liu', email='liu@itheima.com', password='867322', role\_id=ro1.id)

us9 = User(name='li', email='li@163.com', password='4526342', role\_id=ro2.id)

us10 = User(name='sun', email='sun@163.com', password='235523', role\_id=ro2.id)

db.session.add\_all([us1, us2, us3, us4, us5, us6, us7, us8, us9, us10])

db.session.commit()

return render\_template('index3.html')

## 连接数据库错误解决

出现找不到pymysql模块，在views包下创建\_\_init\_\_文件，并录入以下内容：

import pymysql

pymysql.install\_as\_MySQLdb()