初始化参数

from flask import Flask

app = Flask(\_\_name\_\_,

static\_url\_path='/static',//静态文件访问路径，可以不传，默认为：/ + static\_folder

static\_folder='static', //静态文件存储的文件夹，可以不传，默认为 static

template\_folder='templates') //模板文件存储的文件夹，可以不传，默认为 templates

@app.route('/')

def index():

return 'Hello World'

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

app.run()

程序加载配置

从配置对象中加载(常用)

app.config.from\_object()

eg:修改后自动重启服务器

#配置对象

class Config(object):

DEBUG = True

# 从配置对象中加载配置

app.config.from\_object(Config)

一些常用配置直接通过app.设置

app.debug = True

app.config[‘DEBUG’] = True

从配置文件中加载

app.config.form\_pyfile() //…(‘ config.ini ’)

从环境变量中加载(了解)

app.config.from\_envvar()

app.run的参数

app.run(host="0.0.0.0", port=5000, debug = True)

指定路由

# 路由添加参数<参数名>

@app.route('/demo1/<int:user\_id>' ) //int:限定整数

def demo1(user\_id):

return 'hello %d' % user\_id

@app.route('/demo2' ,methods=[‘GET’,’POST’])

def demo2():

return 'demo2 %s' % request.method

JSON

json的格式是str类型

str = json.dumps( dict )

dict = json.loads( str )

return jsonify( dict ) 返回的也是str类型

重定向

from flask import redirect

from flask import url\_for

@app.route('/redirect')

def demo2():

return redirect('http://www.baidu.com') #外部网址

return redirect(url\_for('demo1',user\_id=123)) #内部地址

函数名 传值

自定义状态码

@app.route('/demo3')

def demo3():

return 'demo3',666

正则

#自定义正则转换器

from werkzeug.routing import BaseConverter

class RegexConverter(BaseConverter):

def \_\_init\_\_(self,url\_map,\*args):

super(RegexConverter, self).\_\_init\_\_(url\_map)

self.regex = args[0]

#添加到默认的转换器字典中

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

app.url\_map.converters['re'] = RegexConverter

#实现

@app.route('/demo4/<re("[0-9]{3}"):user\_id>')

def demo4(ueserId):

return "demo4 %s"%user\_id

正则匹配完可以转换数据类型

url\_for获取视图函数之后才调用to\_url函数

🡪to\_url🡪 re 🡪 to\_python

class ListConverter(BaseConverter):

def \_\_init\_\_(self, url\_map, \*args):

super(RegexConverter, self).\_\_init\_\_(url\_map)

# 将接受的第1个参数当作匹配规则进行保存

self.regex = args[0]

def to\_python(self, value):

return int(value)

def to\_url(self, value):

return value

抛出异常

from flask import abort

…abort(404) #抛出异常

捕获异常或状态码

@app.errorhandler(404) / (ZeroDivisionError)

def not\_found(error):

return "???"

请求勾子

# 在第一次请求之前调用，可以在此方法内部做一些初始化操作

@app.before\_first\_request

def before\_first\_request():

print("before\_first\_request")

# 在每一次请求之前调用，这时候已经有请求了，可能在这个方法里面做请求的校验

# 如果请求的校验不成功，可以直接在此方法中进行响应，直接return之后那么就不会执行视图函数

@app.before\_request

def before\_request():

print("before\_request")

# if 请求不符合条件:

# return "laowang"

# 在执行完视图函数之后会调用，并且会把视图函数所生成的响应传入,可以在此方法中对响应做最后一步统一的处理

@app.after\_request

def after\_request(response):

print("after\_request")

response.headers["Content-Type"] = "application/json"

return response

# 每一次请求之后都会调用，会接受一个参数，参数是服务器出现的错误信息

@app.teardown\_request

def teardown\_request(err):

print("teardown\_request")

request属性



获取客户端的图片

@app.route('/', methods=['POST'])

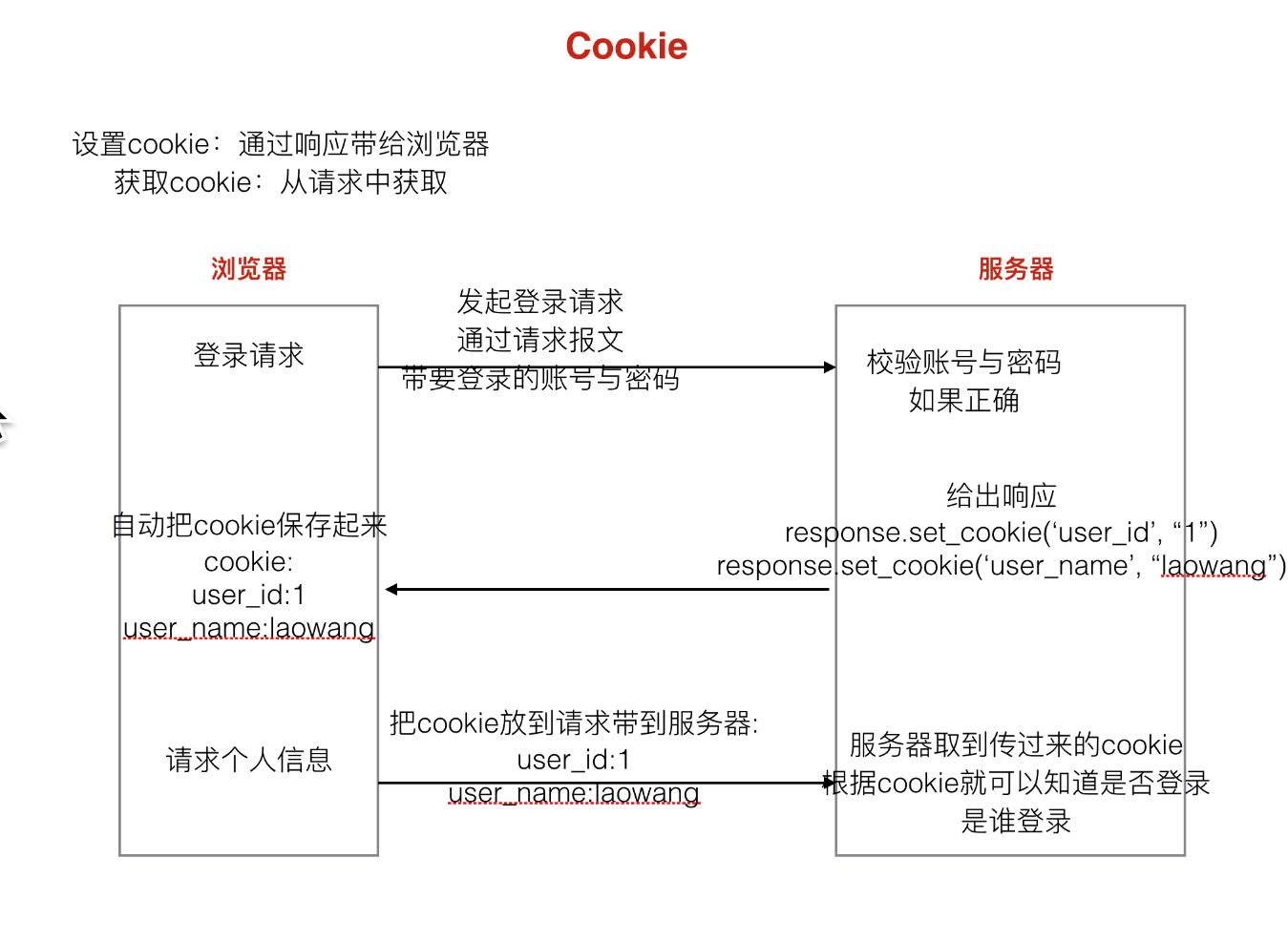
def index():

pic = request.files.get('pic')

pic.save('./static/aaa.png')

return “OK”

Cookie



设置cookie 属性与访问时间

from flask import make\_response, request

@app.route('/login')

def login():

response = make\_response("success") //创建response对象

response.set\_cookie('user\_id','1',max\_age = 3600)

response.set\_cookie('user\_name', 'haha',max\_age = 3600)

return response

访问cookie

@app.route('/')

def index():

user\_id = request.cookies.get('user\_id')

user\_name = request.cookies.get('user\_name')

return user\_id +'<br>'+user\_name

删除cookie

@app.route('/loginout')

def login\_out():

response = make\_response("success")

response.delete\_cookie('user\_id')

response.delete\_cookie('user\_name')

return response

Session



创建session

@app.route('/login')

def login():

session['user\_id']='1'

session['user\_name'] = 'hello'

return 'success'

加密

app.config['SECRET\_KEY'] = 'safe' //加密的字符串

查看session

@app.route('/')

def index():

user\_id= session.get('user\_id','0') //空取0

user\_name = session.get('user\_name','0')

return '%s %s' %(user\_id,user\_name)

删除session

@app.route('/loginout')

def loginout():

session.pop('user\_id')

session.pop('user\_name')

return 'success'

上下文

请求上下文(发生在请求之后)

request

session

应用上下文

current

g变量

Flask-Script

作用,只能用命令执行py程序

#导包

from flask\_script import Manager

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

# 把 Manager 类和应用程序实例进行关联

manager = Manager(app)

if \_\_name\_\_ == "\_\_main\_\_":

#运行出口

manager.run()

运行程序

python demo.py runserver -host ip

模板

设置templates智能提示

setting🡪project🡪Templates

设置html模板

setting🡪python Templates languages🡪jinja2

源文件py中设置

from flask import render\_template

@app.route('/')

def index():

my\_str = 'hello'

my\_int = 10

my\_list = [1,2,3]

my\_dict = {'a':'1','b':2}

return render\_template('demo\_template.html',

my\_str=my\_str,

my\_int=my\_int,

my\_list=my\_list,

my\_dict=my\_dict)

模板html中调用

<div>{{ my\_int + 10 }}</div>

过滤器

字符串过滤器

safe：禁用转义,可以使用标签,默认escape不安全,

<p>{{ '<em>hello</em>' | safe }}</p>

capitalize：把变量值的首字母转成大写，其余字母转小写

<p>{{ 'hello' | capitalize }}</p>

lower：把值转成小写

<p>{{ 'HELLO' | lower }}</p>

upper：把值转成大写

<p>{{ 'hello' | upper }}</p>

title：把值中的每个单词的首字母都转成大写

<p>{{ 'hello' | title }}</p>

reverse：字符串反转

<p>{{ 'olleh' | reverse }}</p>

format：格式化输出

<p>{{ '%s is %d' | format('name',17) }}</p>

striptags：渲染之前把值中所有的HTML标签都删掉

<p>{{ '<em>hello</em>' | striptags }}</p>

truncate: 字符串截断

<p>{{ 'hello every one' | truncate(9)}}</p>

列表过滤器

first：取第一个元素

<p>{{ [1,2,3,4,5,6] | first }}</p>

last：取最后一个元素

<p>{{ [1,2,3,4,5,6] | last }}</p>

length：获取列表长度

<p>{{ [1,2,3,4,5,6] | length }}</p>

sum：列表求和

<p>{{ [1,2,3,4,5,6] | sum }}</p>

sort：列表排序

<p>{{ [6,2,3,1,5,4] | sort }}</p>

sum: 列表指定值求和

my\_new\_list = [{'name':'a','num':1},{'name':'b','num':2}]

<p>{{ my\_new\_list | sum('num') }}</p>

自定义过滤器

方式一、装饰器形式

列表反转

@app.template\_filter('lireverse')

def do\_listreverse(li):

# 通过原列表创建一个新列表

temp\_li = list(li)

# 将新列表进行返转

temp\_li.reverse()

return temp\_li

方式二、直接添加

def do\_listreverse(li):

# 通过原列表创建一个新列表

temp\_li = list(li)

# 将新列表进行返转

temp\_li.reverse()

return temp\_li

app.add\_template\_filter(do\_listreverse,'lireverse') #(func,name)

控制代码块

if/elif/else/endif

{% if 条件 %}

执行的内容

{% elif 条件 %}

执行的内容

{% else %}

执行的内容

{% endif %}



{{ loop.cycle('a','b',’c’) }} #a,b,c循环出现

for/endfor

{% for 参数 in 循环体 %}

每次循环执行的内容

{% endfor %}

宏

#宏定义,类似函数

{% macro function(…)%}

#功能内容

{% endmacro %}

#调用

{{ function(…)}}

#文件外调用

{% import 'macro.html' as func %}

{% func.function() %}

继承

父模板 base.html

{% block top %}

顶部菜单

{% endblock top %}

{% block content %}

{% endblock content %}

{% block bottom %}

底部

{% endblock bottom %}

子模版

extends指令声明这个模板继承自哪

{% extends 'base.html' %}

{% block content %}

需要填充的内容

{% endblock content %}

包含

hello.html中,增加内容

<h1>aaa</h1>

{% include 'hello.html' %}

{% include 'hello.html' ignore missing %} #忽略报错

模板中特有函数

{{ config.DEBUG }} #查看配置

{{ request.url }} #查看路由

{{ session.name }} #session取值

{{ g.name }} #取g变量

{{ url\_for(‘index’) }} #跳转链接

<a herf=” {{ url\_for(‘index’) }}”>回到首页</a>

{% for message in get\_flashed\_messages() %} #获取闪现消息

{{message}}

{% endfor %}

web表单

视图函数:

from flask import Flask,render\_template, flash

#导入wtf扩展的表单类

from flask\_wtf import FlaskForm

#导入自定义表单需要的字段

from wtforms import SubmitField,StringField,PasswordField

#导入wtf扩展提供的表单验证器

from wtforms.validators import DataRequired,EqualTo

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

app.config['SECRET\_KEY']='SECRET\_KEY'

app.config['WTF\_CSRF\_ENABLED'] = False #关闭CSRF校验

#自定义表单类，文本字段、密码字段、提交按钮

class RegisterForm(FlaskForm):

username = StringField("用户名：", validators=[DataRequired("请输入用户名")], render\_kw={"placeholder": "请输入用户名"}) #(laber,验证,添加css)

password = PasswordField("密码：", validators=[DataRequired("请输入密码")])

password2 = PasswordField("确认密码：", validators=[DataRequired("请输入确认密码"), EqualTo("password", "两次密码不一致")])

submit = SubmitField("注册")

#定义根路由视图函数，生成表单对象，获取表单数据，进行表单数据验证

@app.route('/demo2', methods=["get", "post"])

def demo2():

register\_form = RegisterForm()

# 验证表单

if register\_form.validate\_on\_submit():

# 如果代码能走到这个地方，那么就代码表单中所有的数据都能验证成功

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

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

password2 = request.form.get("password2")

# 假装做注册操作

print(username, password, password2)

return "success"

else:

if request.method == "POST":

flash("参数有误或者不完整")

return render\_template('temp\_register.html', form=register\_form)

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

app.run(debug=True)

模板:

<form method="post">

{{ form.username.label }} {{ form.username }}<br/>

{{ form.password.label }} {{ form.password }}<br/>

{{ form.password2.label }} {{ form.password2 }}<br/>

{{ form.submit }}

</form>





CSRF

app.config['WTF\_CSRF\_ENABLED'] = True #开启csrf

模板中添加

{{ from.csrf.token() }}

数据库

from flask\_sqlalchemy import SQLAlchemy

#设置数据库连接

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'mysql://root:mysql@127.0.0.1:3306/test'

# 动态追踪修改设置，如未设置只会提示警告

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = True

#创建表

class Role(db.Model):

\_\_tablename\_\_ = 'roles' # 定义表名

id = db.Column(db.Integer, primary\_key=True) # 定义列对象id

us = db.relationship('User', backref='role', lazy='dynamic') #创建User表的外键关系,方便查看ro1.us , us1. 'role' , lazy='dynamic'开启的话us就是查询的对象,否则加载时就返回数值, 双向开启backref=db.backref('role', lazy='dynamic')

#查找后显示一个可读的字符串

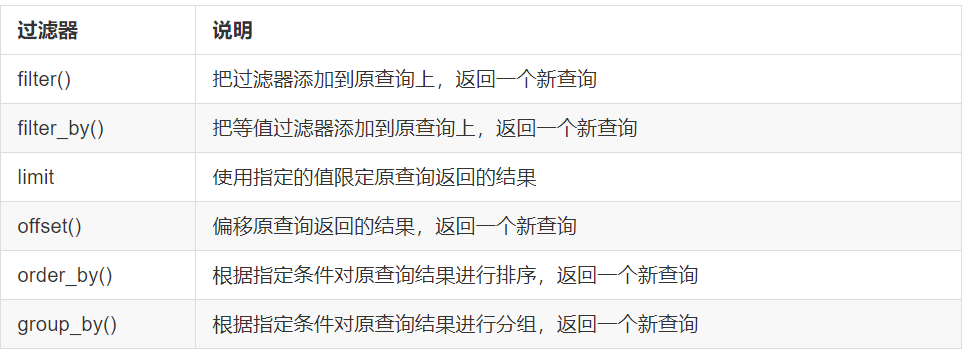
def \_\_repr\_\_(self):

return 'Role:%s'% self.name

class User(db.Model):

role\_id = db.Column(db.Integer, db.ForeignKey('roles.id')) #创建对应表的外键

常用过滤器



常用SQLAlchemy查询执行器



创建表 db.create\_all()

删除表 db.drop\_all()

插入一条数据

ro1 = Role(name='admin')

db.session.add(ro1)

db.session.commit()

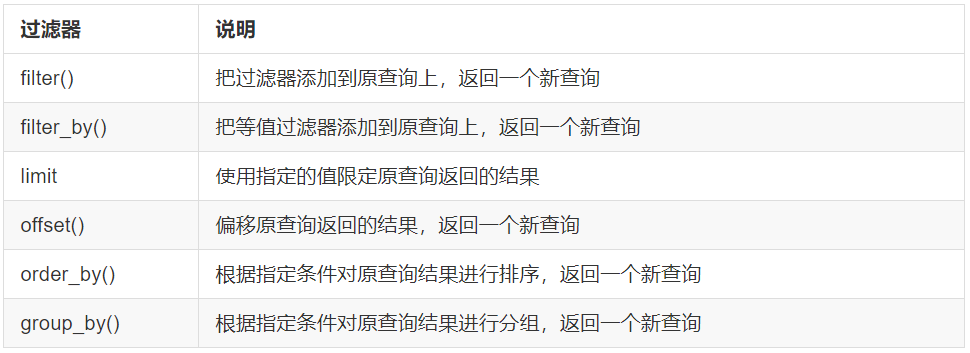
插入多条数据

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)

db.session.add\_all([us1,us2])

db.session.commit()





eg:

查询所有用户数据

User.query.all()

查询有多少个用户

User.query.count()

查询第1个用户

User.query.first()

查询id为4的用户[3种方式]

User.query.get(4)

User.query.filter\_by(id=4).first()

User.query.filter(User.id == 4).first()

查询名字结尾字符为g的所有数据[开始statswith/包含contains]

User.query.filter(User.name.endswith('g')).all()

查询名字不等于wang的所有数据[2种方式]

from sqlalchemy import not\_

User.query.filter(not\_(User.name == 'wang')).all()

User.query.filter(User.name != 'wang').all()

查询名字和邮箱都以 li 开头的所有数据[2种方式]

from sqlalchemy import and\_

User.query.filter(User.name.startswith('li'), User.email.startswith('li')).all()

User.query.filter(and\_(User.name.startswith('li'), User.email.startswith('li'))).all()

查询password是 `123456` 或者 `email` 以 `itheima.com` 结尾的所有数据

from sqlalchemy import or\_

User.query.filter(or\_(User.password=='123456', User.email.endswith('itheima.com'))).all()

查询id为 [1, 3, 5, 7, 9] 的用户列表

User.query.filter(User.id.in\_([1, 3, 5, 7, 9])).all()

查询name为liu的角色数据

User.query.filter(User.name == 'liu').first().role

查询所有用户数据，并以邮箱排序

User.query.order\_by(User.email.desc()).all()

每页3个，查询第2页的数据

paginate = User.query.paginate(2, 3) # 第1个参数代表查询第几页，第2个参代表每页几个

paginate.items 当前页数据

paginate.pages 总页数

paginate.page 当前页

蓝图

单文件蓝图

demo.html中

from flask import Blueprint

#创建蓝图

admin\_blu = Blueprint('order',\_\_name\_\_)

#创建蓝图中的路由

@ admin \_blu.route('/admin')

def admin\_print():

return ' admin\_print '

index.html中

#导入蓝图

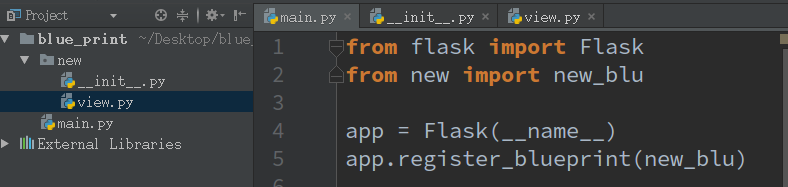
from demo import admin\_blu

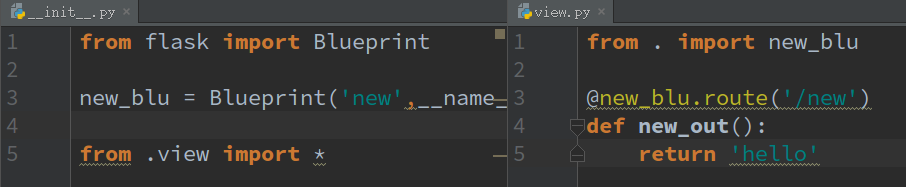
app = Flask(\_\_name\_\_)

#注册蓝图

app.register\_blueprint(admin\_blu)

模块蓝图





# 添加子静态文件,子模板访问的路径, url\_prefix添加访问路径的前缀

new\_blu=Blueprint('new',\_\_name\_\_,url\_prefix='/new',static\_folder='static',template\_folder='templates')

单元测试