Flask课堂笔记

# HTTP通信与Web框架

## 1.1 流程

客户端将请求打包成HTTP的请求报文（HTTP协议格式的请求数据）

采用TCP传输发送给服务器端

服务器接收到请求报文后按照HTTP协议进行解析

服务器根据解析后获知的客户端请求进行逻辑执行

服务器将执行后的结果封装成HTTP的响应报文（HTTP协议格式的响应数据）

采用刚才的TCP连接将响应报文发送给客户端

客户端按照HTTP协议解析响应报文获取结果数据

## 1.2 细节

客户端不一定是浏览器，也可以是PC软件、手机APP、程序

根据服务器端的工作，将其分为两部分：

服务器：与客户端进行tcp通信，接收、解析、打包、发送http格式数据

业务程序：根据解析后的请求数据执行逻辑处理，形成要返回的数据交给服务器

服务器与Python业务程序的配合使用WSGI协议

## 1.3 Web框架

能够被服务器调用起来，根据客户端的不同请求执行不同的逻辑处理形成要返回的数据的 程序

核心：实现路由和视图（业务逻辑处理）

## 1.4 框架的轻重

重量级的框架：为方便业务程序的开发，提供了丰富的工具、组件，如Django

轻量级的框架：只提供Web框架的核心功能，自由、灵活、高度定制，如Flask、Tornado

## 1.5 明确Web开发的任务

视图开发：根据客户端请求实现业务逻辑（视图）编写

模板、数据库等其他的都是为了帮助视图开发，不是必备的

# 认识Flask

## 2.1 简介

Flask诞生于2010年，是Armin ronacher（人名）用Python语言基于Werkzeug工具箱编写的轻量级Web开发框架。它主要面向需求简单的小应用。

Flask本身相当于一个内核，其他几乎所有的功能都要用到扩展（邮件扩展Flask-Mail，用户认证Flask-Login），都需要用第三方的扩展来实现。比如可以用Flask-extension加入ORM、窗体验证工具，文件上传、身份验证等。Flask没有默认使用的数据库，你可以选择MySQL，也可以用NoSQL。其 WSGI 工具箱采用 Werkzeug（路由模块） ，模板引擎则使用 Jinja2 。

可以说Flask框架的核心就是Werkzeug和Jinja2。

Python最出名的框架要数Django，此外还有Flask、Tornado等框架。虽然Flask不是最出名的框架，但是Flask应该算是最灵活的框架之一，这也是Flask受到广大开发者喜爱的原因。

## 2.2 与Django对比

django提供了：

django-admin快速创建项目工程目录

manage.py 管理项目工程

orm模型（数据库抽象层）

admin后台管理站点

缓存机制

文件存储系统

用户认证系统

而这些，flask都没有，都需要扩展包来提供

## 2.3 Flask扩展包：

Flask-SQLalchemy：操作数据库；

Flask-migrate：管理迁移数据库；

Flask-Mail:邮件；

Flask-WTF：表单；

Flask-script：插入脚本；

Flask-Login：认证用户状态；

Flask-RESTful：开发REST API的工具；

Flask-Bootstrap：集成前端Twitter Bootstrap框架；

Flask-Moment：本地化日期和时间；

## 2.4 Flask文档

中文文档： <http://docs.jinkan.org/docs/flask/>

英文文档： <http://flask.pocoo.org/docs/0.11/>

# 创建虚拟环境

虚拟环境是一个互相隔离的目录

1. mkvirtualenv flask\_py2
2. pip install flask==0.10.1

pip freeze > requirements.txt

pip install –r requirements.txt

# Flask的Hello world程序

# coding:utf-8

# 导入Flask类

from flask import Flask

#Flask类接收一个参数\_\_name\_\_

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

# 装饰器的作用是将路由映射到视图函数index

@app.route('/')

def index():

return 'Hello World'

# Flask应用程序实例的run方法启动WEB服务器

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

app.run()

## 4.1 Flask创建app对象

### 4.1.1 初始化参数

import\_name: 导入路径 （寻找静态目录与模板目录位置的参数）

static\_url\_path:访问静态资源的url前缀

static\_folder: 默认‘static’，静态文件的目录

template\_folder: 默认‘templates’

#coding=utf-8  
from flask import Flask,current\_app  
# import demo  
#创建flask应用对象  
# \_\_name\_\_表示当前的模块名字  
app=Flask(\_\_name\_\_,  
 static\_url\_path='/python',#访问静态资源的url前缀，默认是static  
 static\_folder="static",#静态文件的目录 默认就是static  
 template\_folder="templates")#模板文件的目录 默认是templates  
# app=Flask('\_\_main\_\_')  
# app=Flask('abc')  
  
 #配置参数的方式  
 # 1:使用配置文件  
# app.config.from\_pyfile('config.cfg')  
  
# 2 使用队形配置参数  
class Config(object):  
 DEBUG=True  
 ITCAST='python'  
app.config.from\_object(Config)  
  
# 3:直接操作config的字典对象  
# app.config['DEBUG']=True  
  
@app.route("/")  
def index():  
 # a=1/0  
 return "hello word"  
  
if \_\_name\_\_=="\_\_main\_\_":  
 #启动flask程序  
 app.run()

### 4.1.2 配置参数file:/D:/money/flaskDemo/venv/

1新建一个config.cfg的文件 用于配置参数使用

DEBUG=True

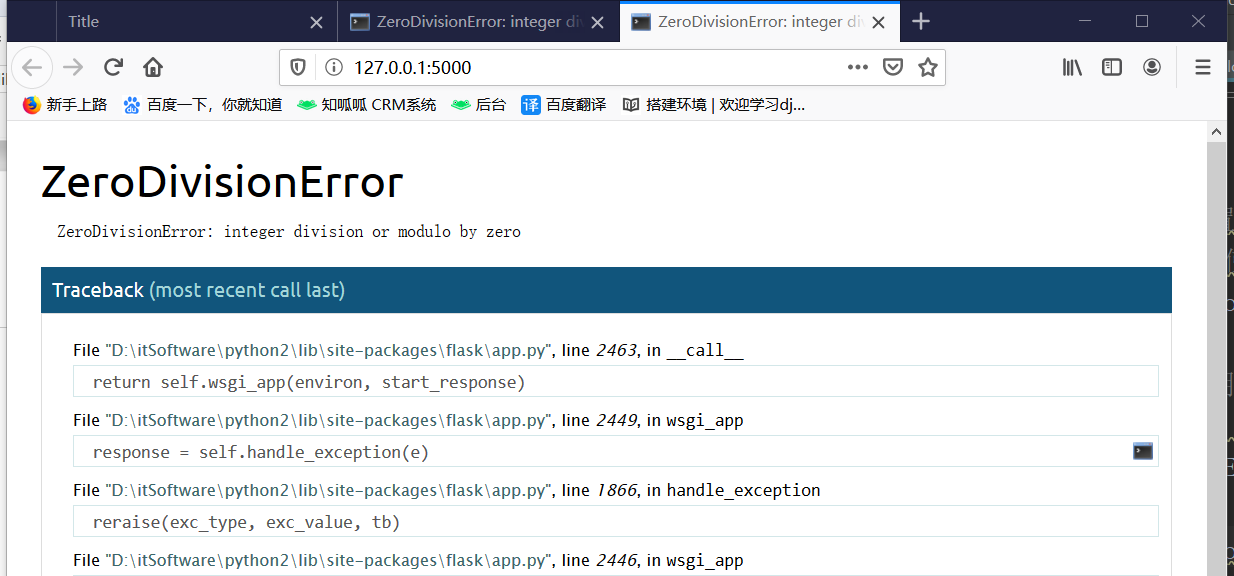
2 再去hello.py里面去写配置参数，在flask中使用配置参数

app.config.from\_pyfile(“config.cfg”) 使用配置文件

app.config.from\_object()使用对象配置参数

app.config[‘DEBUG’]=True 直接操作config字典对象

#配置参数的方式  
 # 1:使用配置文件  
# app.config.from\_pyfile('config.cfg')  
  
# 2 使用队形配置参数  
class Config(object):  
 DEBUG=True  
 ITCAST='python'  
app.config.from\_object(Config)  
  
# 3:直接操作config的字典对象  
# app.config['DEBUG']=True



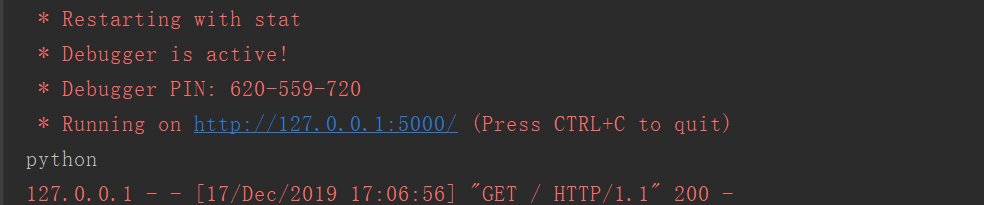
### 4.1.3 在视图读取配置参数

1:app.config.get()

直接从全局对象app的config字典中取值



这样设置好了之后要去页面刷新一下 然后在回来pycharm 在控制台看输出结果



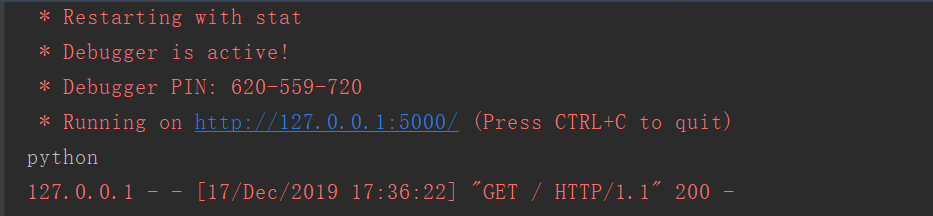
2：current\_app.config.get()：

通过cuttent\_app获取参数

#coding=utf-8  
from flask import Flask,current\_app  
# import demo

@app.route("/")  
def index():  
 # a=1/0  
 #读取配置参数  
 # 1：直接从全局对象app的config的字典中取值  
 # print(app.config.get('ITCAST'))  
  
 # 2:通过cuttent\_app获取参数  
 print(current\_app.config.get("ITCAST"))  
  
 return "hello word"

这样设置好了之后要去页面刷新一下 然后在回来pycharm 在控制台看输出结果



### 4.1.4 app.run的参数

app.run(host=”0.0.0.0”, port=5000)

if \_\_name\_\_=="\_\_main\_\_":  
 #启动flask程序  
 app.run(host='0.0.0.0',port=5000)

设置0.0.0.0 不管是外网还是哪里 虚拟机还是windows都可以访问

另外debug也可以加进去

if \_\_name\_\_=="\_\_main\_\_":  
 #启动flask程序  
 app.run(host='0.0.0.0',port=5000,debug=True)

## 4.2 路由

### 4.2.1 app.url\_map 查看所有路由

if \_\_name\_\_=="\_\_main\_\_":  
 #  
 print(app.url\_map)  
 #启动flask程序  
 app.run(debug=True)

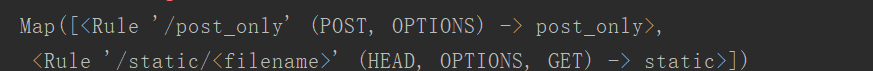
举例：默认是get提交 现在设置 只能以post提交怎么设置？

方法：配置路由的时候加上method=[‘POST’]

#通过method限定访问方式

@app.route('/post\_only',methods=['POST'])  
def post\_only():  
 *'''只能post访问'''* return 'post only page'

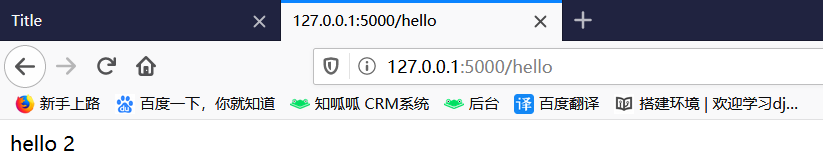
输出结果：



### 4.2.2 同一路由装饰多个视图函数

例：两个视图函数对于同一个路由 不加入method方法的话 只能显示第一个hello1 的视图 ， 如是想显示hello 2 需要加入method方法 就可以显示了

@app.route('/hello',methods=['POST'])  
def hello1():  
 return 'hello 1'  
  
  
@app.route('/hello',methods=['GET'])  
def hello2():  
 return 'hello 2'



### 4.2.3 同一视图多个路由装饰器

例： 一个视图对于两个路由 访问页面hi1 和hi2都可以显示页面

@app.route('/hi1')  
@app.route('/hi2')  
def hi():  
 return 'hi page'





### 4.2.4 利用methods限制访问方式

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

### 4.2.5 使用url\_for进行反解析

from flask import Flask,current\_app,redirect,url\_for

@app.route('/login')  
def login():  
 #使用url\_for的函数，通过视图函数的名字找到对应的url路径  
 url = url\_for('index')  
 return redirect(url)  
  
@app.route('/register')  
def register():  
 url=url\_for('index')  
 return redirect(url)

再去访问页面login 或者Register时 就直接跳转到index页面了

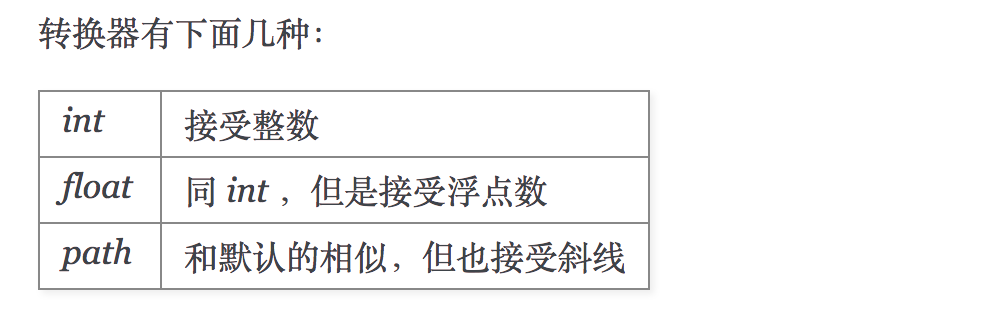
### 4.2.5 动态路由

# 路由传递的参数默认当做string处理，这里指定int，尖括号中冒号后面的内容是动态的

@app.route('/user/<int:id>')

def hello\_itcast(id):

return 'hello itcast %d' %id



#转换器  
# @app.route('/goods\_detail/<int:goods\_id>')  
@app.route('/goods\_detail/<goods\_id>')#不加转换器类型 默认是普通的字符串规则 （除了/的字符  
def goods\_detail(goods\_id):  
 return 'goods detail page %s' % goods\_id

### 4.2.5 自定义转换器

from flask import Flask

from werkzeug.routing import BaseConverter

class Regex\_url(BaseConverter):

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

super(Regex\_url,self).\_\_init\_\_(url\_map)

self.regex = args[0]

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

app.url\_map.converters['re'] = Regex\_url

@app.route('/user/<re("[a-z]{3}"):id>')

def hello\_itcast(id):

return 'hello %s' %id

举例手机号：

from werkzeug.routing import BaseConverter

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

#自定义转换器  
# 举例手机号  
  
# 1：定义自己的转化器  
class RegexConverter(BaseConverter):  
 def \_\_init\_\_(self, url\_map, regex):  
 #调用父类的初始化方法  
 super(RegexConverter, self).\_\_init\_\_(url\_map)  
 #将正则表达式的参数保存到对象的属性中，flask会去使用这个属性来进行路由的正则匹配  
 self.regex = regex  
  
# 2:将自定义的转换器添加到flask的应用中  
app.url\_map.converters['re']=RegexConverter  
  
# 3:使用  
# 127.0.0.1：5000/seed/18612345678  
  
@app.route("/send/<re(r'1[34578]\d{9}'):mobile>")  
def send\_sms(mobile):  
 return 'send sms to %s' % mobile

if \_\_name\_\_=="\_\_main\_\_":  
 #  
 print(app.url\_map)  
 #启动flask程序  
 app.run(debug=True)

访问send/186.....后面加手机号 即可显示send sms to 手机号

**举例手机号二： to\_python 和to\_url方法**

# 1：定义自己的转化器  
class RegexConverter(BaseConverter):  
 def \_\_init\_\_(self, url\_map, regex):  
 #调用父类的初始化方法  
 super(RegexConverter,self).\_\_init\_\_(url\_map)  
 #将正则表达式的参数保存到对象的属性中，flask会去使用这个属性来进行路由的正则匹配  
 self.regex = regex  
 def to\_python(self, value):  
 #value时在路径进行正则表达式匹配的时候提取的参数  
 return value  
 def to\_url(self, value):  
 *'''使用url\_for的方式的时候被调用'''* return value  
  
# 2:将自定义的转换器添加到flask的应用中  
app.url\_map.converters['re']=RegexConverter  
  
# 3:使用  
# 127.0.0.1：5000/seed/18612345678  
  
@app.route("/send/<re(r'1[34578]\d{9}'):mobile\_num>")  
def send\_sms(mobile\_num):  
 return 'send sms to %s' % mobile\_num  
  
@app.route('/index')  
def index():  
 url=url\_for('send\_sms',mobile\_num='18611111111')  
 return redirect(url)

## 4.3 获取请求参数

from flask import request

就是 Flask 中表示当前请求的 request 对象，request对象中保存了一次HTTP请求的一切信息。



### 4.3.1 上传文件

已上传的文件存储在内存或是文件系统中一个临时的位置。你可以通过请求对象的 files 属性访问它们。每个上传的文件都会存储在这个字典里。它表现近乎为一个标准的 Python file 对象，但它还有一个 save() 方法，这个方法允许你把文件保存到服务器的文件系统上。这里是一个用它保存文件的例子:

from flask import request

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

def upload\_file():

if request.method == 'POST':

f = request.files['the\_file']

f.save('/var/www/uploads/uploaded\_file.txt')

...

如果你想知道上传前文件在客户端的文件名是什么，你可以访问 filename 属性。但请记住， 永远不要信任这个值，这个值是可以伪造的。如果你要把文件按客户端提供的文件名存储在服务器上，那么请把它传递给 Werkzeug 提供的 secure\_filename() 函数:

from flask import request

from werkzeug import secure\_filename

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

def upload\_file():

if request.method == 'POST':

f = request.files['the\_file']

f.save('/var/www/uploads/' + secure\_filename(f.filename))

...

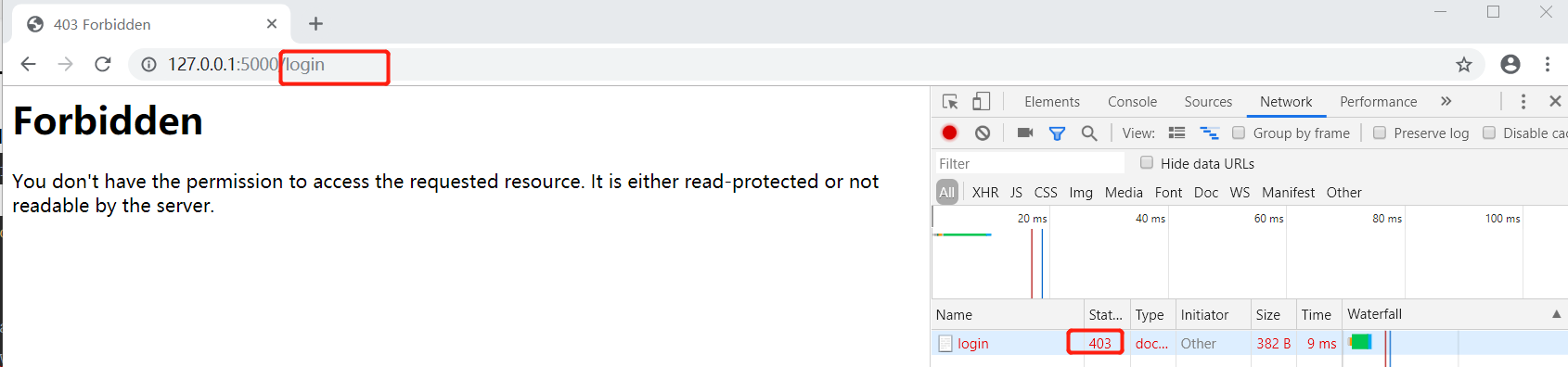
## 4.4 abort函数与自定义异常处理

### 4.4.1 abort函数

from flask import abort

from flask import Flask,request,abort,Response

def login():  
 # name=request.form.get()  
 # pwd=request.form.get()  
 name = ''  
 pwd = ''  
 if name != 'zhangsan' or pwd != 'admin':  
 #使用abort函数可以立即终止视图函数的执行  
 #并可以返回给前端特定的信息  
 # 1：传递状态码信息,必须是标准的http状态码  
 abort(403)  
 # 2：传递响应体信息  
 res=Response('login yyy')  
 abort(res)  
 return 'login success'



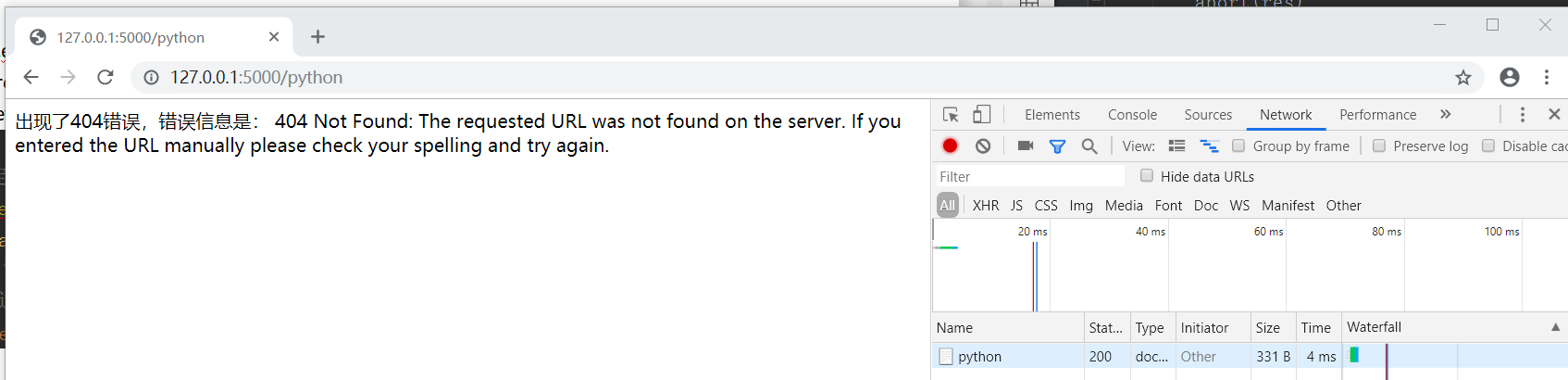
### 4.4.2 自定义异常处理

@app.errorhandler(404)

def error(e):

return '您请求的页面不存在了，请确认后再次访问！%s'%e

# 自定义异常处理  
@app.errorhandler(404)  
def handle\_404\_error(err):  
 *'''自定义的处理错误方法'''* #这个函数的返回值会是前端用户看到的最终结果  
 return u'出现了404错误，错误信息是： %s'% err

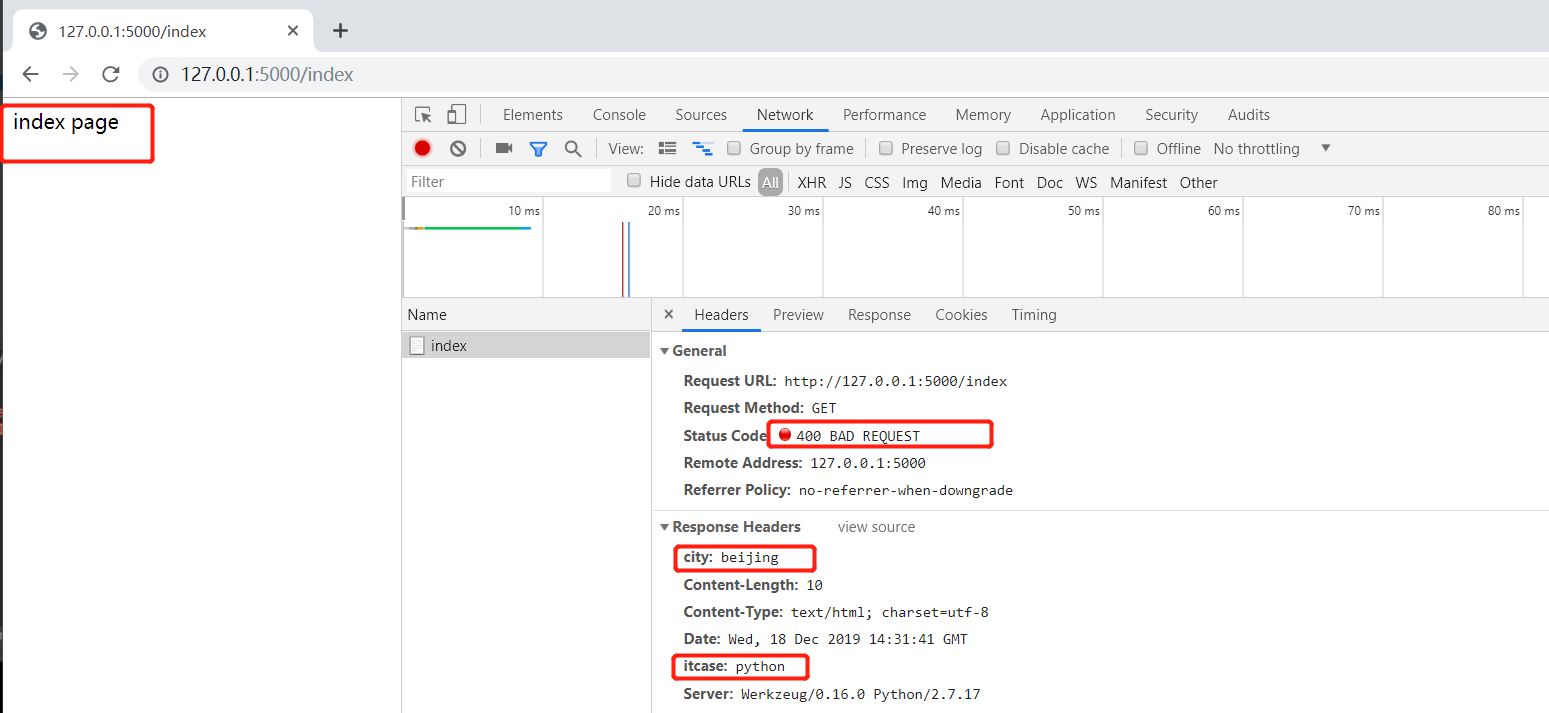


## 4.5 返回的响应数据

### 4.5.1 元组

可以返回一个元组，这样的元组必须是 **(response, status, headers)** 的形式，且至少包含一个元素。 status 值会覆盖状态代码， headers 可以是一个列表或字典，作为额外的消息标头值。

#coding:utf-8  
  
from flask import Flask,abort,Response,request  
app = Flask(\_\_name\_\_)  
  
@app.route('/index')  
def index():  
 #1:使用元组，返回自定义的响应信息  
 # 响应体 状态码 响应头  
 # return 'index page' , 400 , [('city','beijing'),('itcase','python')]  
 # 2: 响应体可以使用字典的形式  
 # return 'index page' , 400, {"city":"beijing","itcase":"python"}  
 # 3：状态码可以改成其他 例如 666  
 # return 'index page' , 666, {"city":"beijing","itcase":"python"}  
 # 4：状态码中可以加说明  
 # return 'index page' , "666 itcast status", {"city":"beijing","itcase":"python"}  
 # 5:后面的响应头可以不写 到那时状态码不可以不写  
 # return 'index page' , "666 itcast status"  
  
if \_\_name\_\_=='\_\_main\_\_':  
 app.run(debug=True)



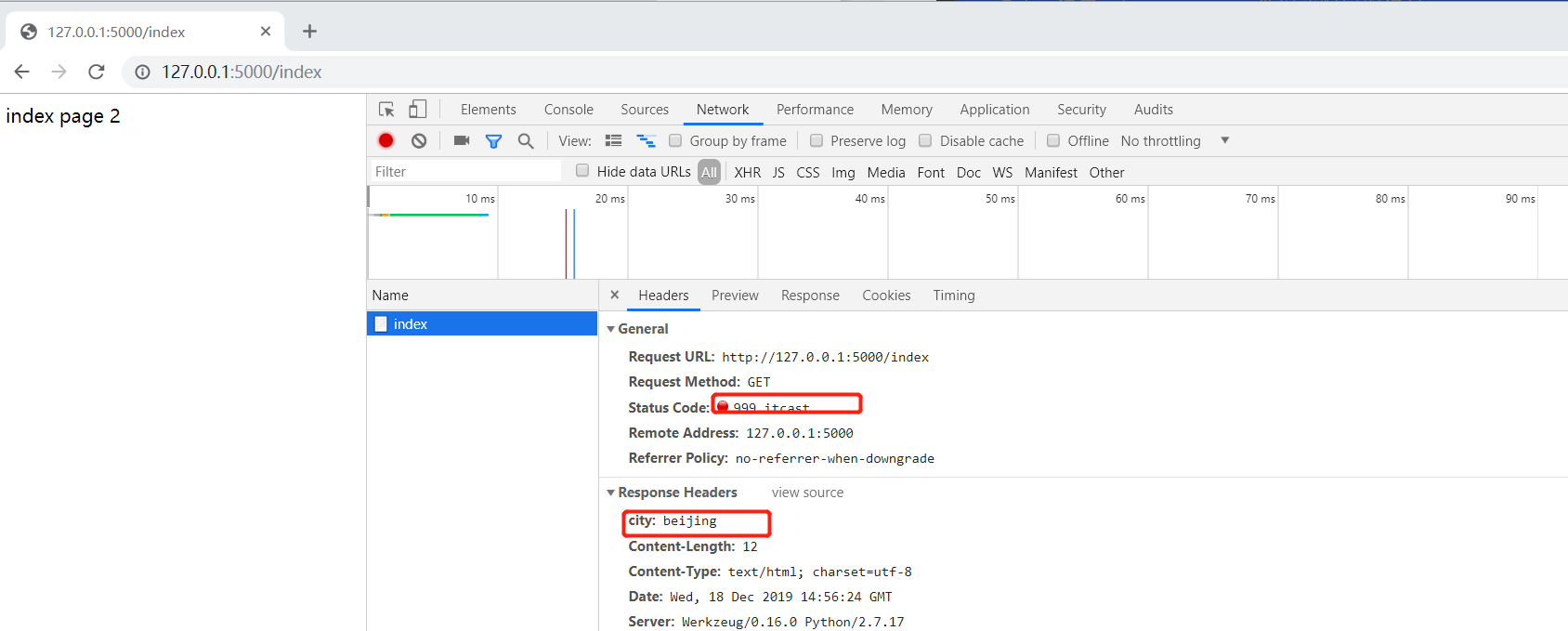
### 4.5.2 make\_response

resp = make\_response()

resp.headers[“sample”] = “value”

resp.status = “404 not found”

# 6：使用make\_response 来构造响应的信息  
res=make\_response('index page 2')  
res.status='999 itcast'#设置状态码  
res.headers['city']='beijing'  
return res



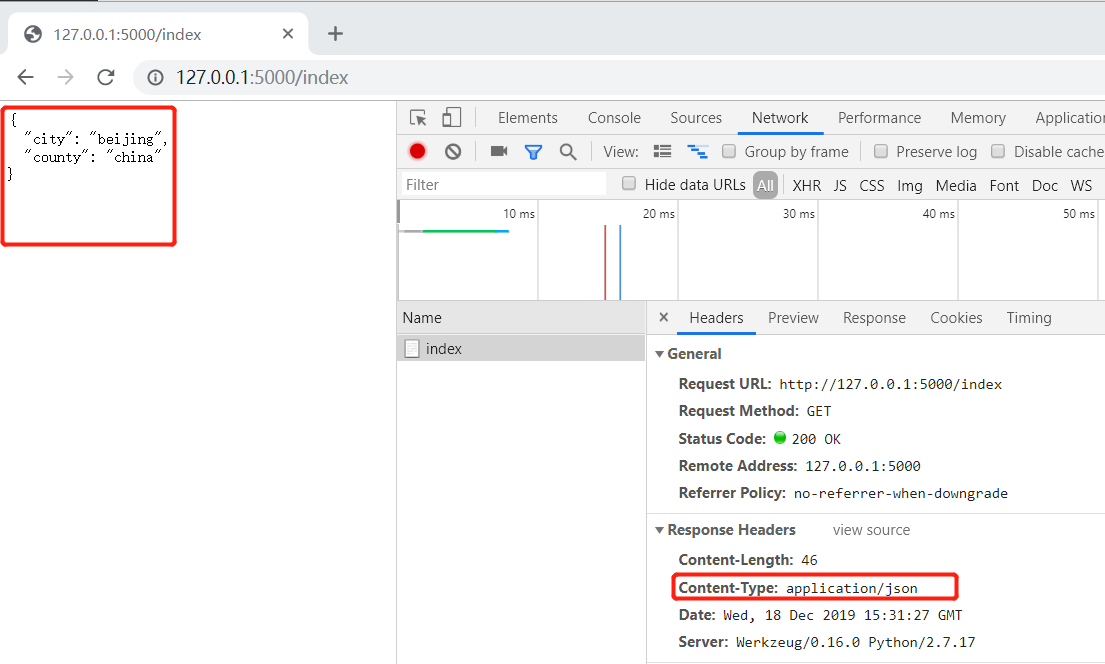
## 4.6 使用jsonify返回json数据

首先json介绍



#coding:utf-8  
from flask import Flask,jsonify  
import json  
  
app = Flask(\_\_name\_\_)

@app.route('/index')  
def index():  
 data={  
 'name':'zahngsan',  
 'age':18  
 }  
# # #json.dumps（字典） 将python的字典转换为接送字符串  
# # # json.loads(字符串) 将字符串转换为python中的字典  
# # json\_str=json.dumps(data)  
# # return json\_str,400,{'Content-Type':'application/json'}  
  
  
#以上是以往的复杂的方式 flask中引入jsonify的操作方式  
#jsonify帮助转为json数据，并设置响应头，Content-Type 为 application/json  
 # return jsonify(data)  
  
#另外一种方式 不用data 可以在里面直接传数据  
 return jsonify(city= 'beijing',county='china')  
  
if \_\_name\_\_=='\_\_main\_\_':  
 app.run(debug=True)



## 4.5 重定向

from flask import redirect

## 4.6 设置和读取cookie

make\_response

set\_cookie(key, value=’’, max\_age=None)

delete\_cookie(key)

#coding:utf-8  
from flask import Flask,request,make\_response  
  
app=Flask(\_\_name\_\_)  
  
@app.route('/set\_cookie')  
def set\_cookie():  
 resp=make\_response('success')  
 resp.set\_cookie('itcast','python')  
 resp.set\_cookie('itcast1','python1')  
 resp.set\_cookie('itcast2','python2',max\_age=3600)  
 return resp  
  
@app.route("/get\_cookie")  
def get\_cookie():  
 c = request.cookies.get("itcast")

#防御性编程，保证程序的健壮性，  
 if c == None:  
 return 'not found'  
 return c

@app.route('/delete\_cookie')  
def delete\_cookie():  
 resp=make\_response('delete success')  
 #删除cookie  
 resp.delete\_cookie('itcast')  
 return resp

if \_\_name\_\_=="\_\_main\_\_":  
 app.run(debug=True)

## 4.7 session

from flask import session

需要设置secret\_key

## 4.8 请求上下文与应用上下文

请求上下文(request context)

request和session都属于请求上下文对象。

应用上下文(application context)

current\_app和g都属于应用上下文对象。

current\_app:表示当前运行程序文件的程序实例。

g:处理请求时，用于临时存储的对象，每次请求都会重设这个变量。

## 4.9 请求钩子

请求钩子是通过装饰器的形式实现，Flask支持如下四种请求钩子：

before\_first\_request：在处理第一个请求前运行。

@app.before\_first\_request

before\_request：在每次请求前运行。

after\_request(response)：如果没有未处理的异常抛出，在每次请求后运行。

teardown\_request(response)：在每次请求后运行，即使有未处理的异常抛出。

# Flask-Script扩展命令行

pip install Flask-Script

from flask import Flask

from flask\_script import Manager

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

manager = Manager(app)

@app.route('/')

def index():

return '床前明月光'

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

manager.run() ...

# Jinja2模板

## 6.1 基本流程

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Template</title>

</head>

<body>

<h1>hello {{ name }}</h1>

</body>

</html>

@app.route("/")

def index():

return render\_template("index.html", name="python")

使用flask 中的**render\_template**渲染模板

## 6.2 变量

<p>{{mydict['key']}}</p>

<p>{{mydict.key}}</p>

<p>{{mylist[1]}}</p>

<p>{{mylist[myvariable]}}</p>

from flask import Flask,render\_template

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

@app.route('/')

def index():

mydict = {'key':'silence is gold'}

mylist = ['Speech', 'is','silver']

myintvar = 0

return render\_template('vars.html',

mydict=mydict,

mylist=mylist,

myintvar=myintvar

)

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

app.run(debug=True)

## 6.3 过滤器

### 6.3.1 字符串过滤器

**safe：禁用转义；**

<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>

**trim：把值的首尾空格去掉；**

<p>{{ ' hello world ' | trim }}</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>

### 6.3.2 支持链式使用过滤器

<p>{{ “ hello world “ | trim | upper }}</p>

### 6.3.3 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>

### 6.3.4 自定义过滤器

自定义的过滤器名称如果和内置的过滤器重名，会覆盖内置的过滤器。

方式一：

通过 **add\_template\_filter (过滤器函数, 模板中使用的过滤器名字)**

def filter\_double\_sort(ls):

return ls[::2]

app.add\_template\_filter(filter\_double\_sort,'double\_2')

方式二：

通过装饰器 **app.template\_filter (模板中使用的装饰器名字)**

@app.template\_filter('db3')

def filter\_double\_sort(ls):

return ls[::-3]

## 6.4 表单

使用Flask-WTF表单扩展，可以帮助进行CSRF验证，帮助我们快速定义表单模板，而且可以帮助我们在视图中验证表的数据

**pip install Flask-WTF**

### 6.4.1 不使用Flask-WTF扩展时，表单需要自己处理

#模板文件

<form method='post'>

<input type="text" name="username" placeholder='Username'>

<input type="password" name="password" placeholder='password'>

<input type="submit">

</form>

from flask import Flask,render\_template,request

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

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

print username,password

return “success”

else:

return render\_template(“login.html”)

### 6.4.2 使用Flask-WTF扩展

需要设置 SECRET\_KEY 的配置参数

模板页：

<form method="post">

#设置csrf\_token

{{ form.csrf\_token() }}

{{ form.us.label }}

<p>{{ form.us }}</p>

{{ form.ps.label }}

<p>{{ form.ps }}</p>

{{ form.ps2.label }}

<p>{{ form.ps2 }}</p>

<p>{{ form.submit() }}</p>

{% for x in get\_flashed\_messages() %}

{{ x }}

{% endfor %}

</form>

视图函数

rf#coding=utf-8

from flask import Flask,render\_template, redirect,url\_for,session,request,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']='1'

#自定义表单类，文本字段、密码字段、提交按钮

class Login(Flask Form):

us = StringField(label=u'用户：',validators=[DataRequired()])

ps = PasswordField(label=u'密码',validators=[DataRequired(),EqualTo('ps2','err')])

ps2 = PasswordField(label=u'确认密码',validators=[DataRequired()])

submit = SubmitField(u'提交')

#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('/',methods=['GET','POST'])

def index():

form = Login()

if form.validate\_on\_submit():

name = form.us.data

pswd = form.ps.data

pswd2 = form.ps2.data

print name,pswd,pswd2

return redirect(url\_for('login'))

else:

if request.method=='POST':

flash(u'信息有误，请重新输入！')

return render\_template('index.html',form=form)

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

app.run(debug=True)

## 6.5 控制语句

### 6.5.1 if语句

{% if %} {% endif %}

### 6.5.2 for语句

{% for item in samples %} {% endfor %}

## 6.6 宏

类似于python中的函数，宏的作用就是在模板中重复利用代码，避免代码冗余。

### 6.6.1 不带参数宏的定义与使用

**定义：**

{% macro input() %}

<input type="text"

name="username"

value=""

size="30"/>

{% endmacro %}

**使用**

{{ input() }}

### 6.6.2 带参数宏的定义与使用

定义

{% macro input(name,value='',type='text',size=20) %}

<input type="{{ type }}"

name="{{ name }}"

value="{{ value }}"

size="{{ size }}"/>

{% endmacro %}

使用

{{ input(value='name',type='password',size=40)}}

### 6.6.3 将宏单独封装在html文件中

文件名可以自定义macro.html

{% macro input() %}

<input type="text" name="username" placeholde="Username">

<input type="password" name="password" placeholde="Password">

<input type="submit">

{% endmacro %}

在其它模板文件中先导入，再调用

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

{% func.input() %}

## 6.4 模板继承

extend

## 6.5 模板包含

include

## 6.6 flask在模板中使用特殊变量和方法

### 6.6.1 config

### 6.6.2 request

### 6.6.3 url\_for