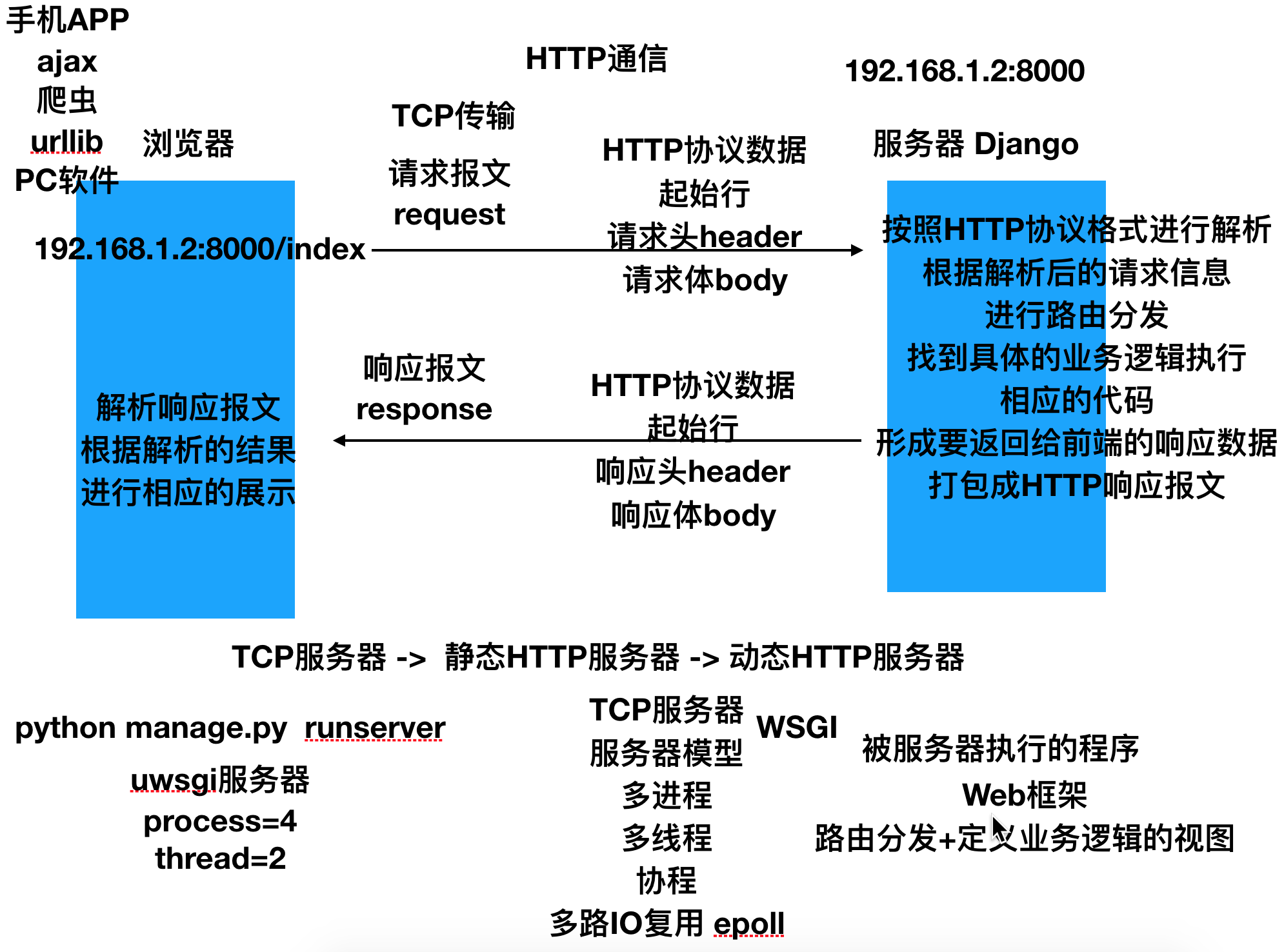
Flask课堂笔记

# HTTP通信与Web框架

## 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、 Webpy

## 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:

static\_url\_path:

static\_folder: 默认‘static’

template\_folder: 默认‘templates’

### 4.1.2 配置参数

app.config.from\_pyfile(“yourconfig.cfg”) 或

app.config.from\_object()

### 4.1.3 在视图读取配置参数

app.config.get() 或者 current\_app.config.get()

### 4.1.4 app.run的参数

app.run(host=”0.0.0.0”, port=5000)

## 4.2 路由

### 4.2.1 app.url\_map 查看所有路由

### 4.2.2 同一路由装饰多个视图函数

谁在上，谁被执行

### 4.2.3 同一视图多个路由装饰器

### 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

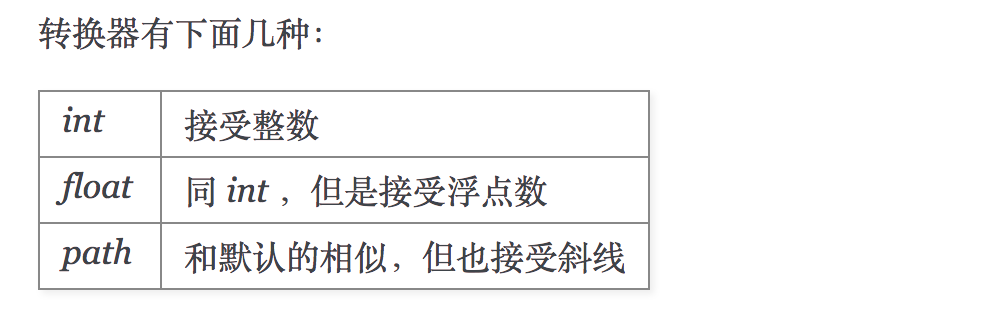
### 4.2.5 动态路由

# 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



### 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\_\_)

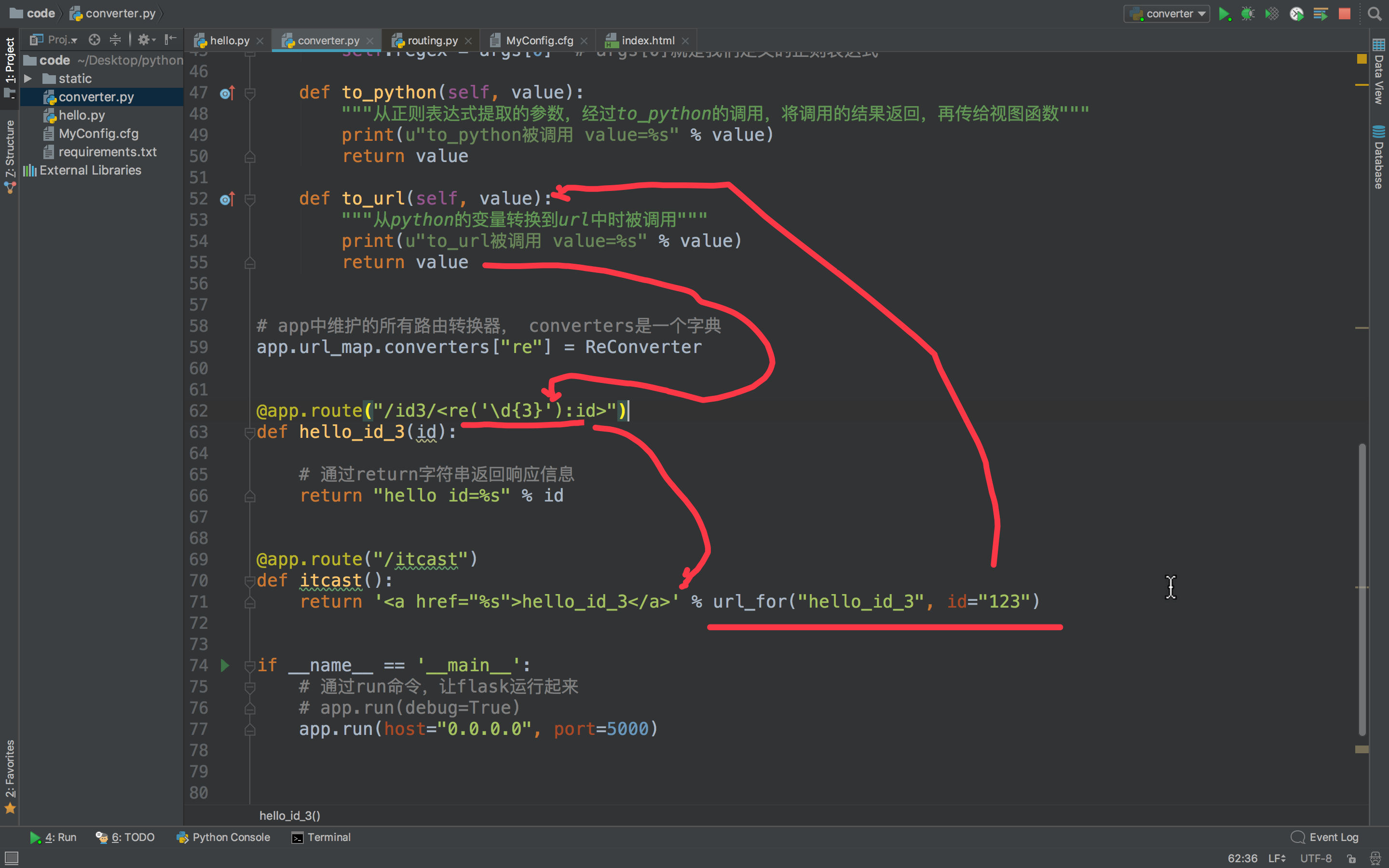
app.url\_map.converters['re'] = Regex\_url

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

def hello\_itcast(id):

return 'hello %s' %id

## 



## 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

### 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

## 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。

### 4.5.2 make\_response

resp = make\_response()

resp.headers[“sample”] = “value”

resp.status = “404 not found”

## 4.6 使用jsonify返回json数据

## 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)

## 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() ...