# Flask 课堂笔记

## 1. 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

# 2. 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/

# 3. 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

# 4. 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

### 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
```

转换器有下面几种：

| | |
|---|---|
| *int* | 接受整数 |
| *float* | 同 *int*，但是接受浮点数 |
| *path* | 和默认的相似，但也接受斜线 |

### 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。

request常用的属性如下：

| 属性 | 说明 | 类型 |
| --- | --- | --- |
| data | 记录请求的数据，并转换为字符串 | * |
| form | 记录请求中的表单数据 | MultiDict |
| args | 记录请求中的查询参数 | MultiDict |
| cookies | 记录请求中的cookie信息 | Dict |
| headers | 记录请求中的报文头 | EnvironHeaders |
| method | 记录请求使用的HTTP方法 | GET/POST |
| url | 记录请求的URL地址 | string |
| files | 记录请求上传的文件 | * |

### 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。

# 5. 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()       ...
```

# 6. 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