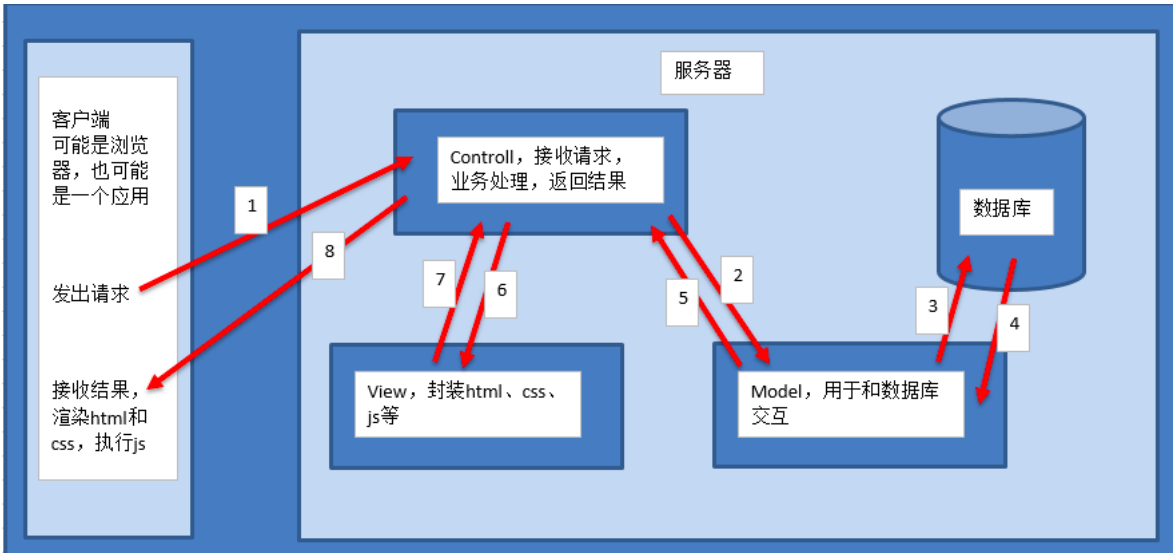
Day1 Flask入门

一、MVC与MTV

一种软件设计典范，用一种业务逻辑，使数据，界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面与用户交互的同时，不需要重新编写业务逻辑。MVC被独特的发展起来用于映射传统的输入，处理和输出功能在一个逻辑的图形化界面结构中。

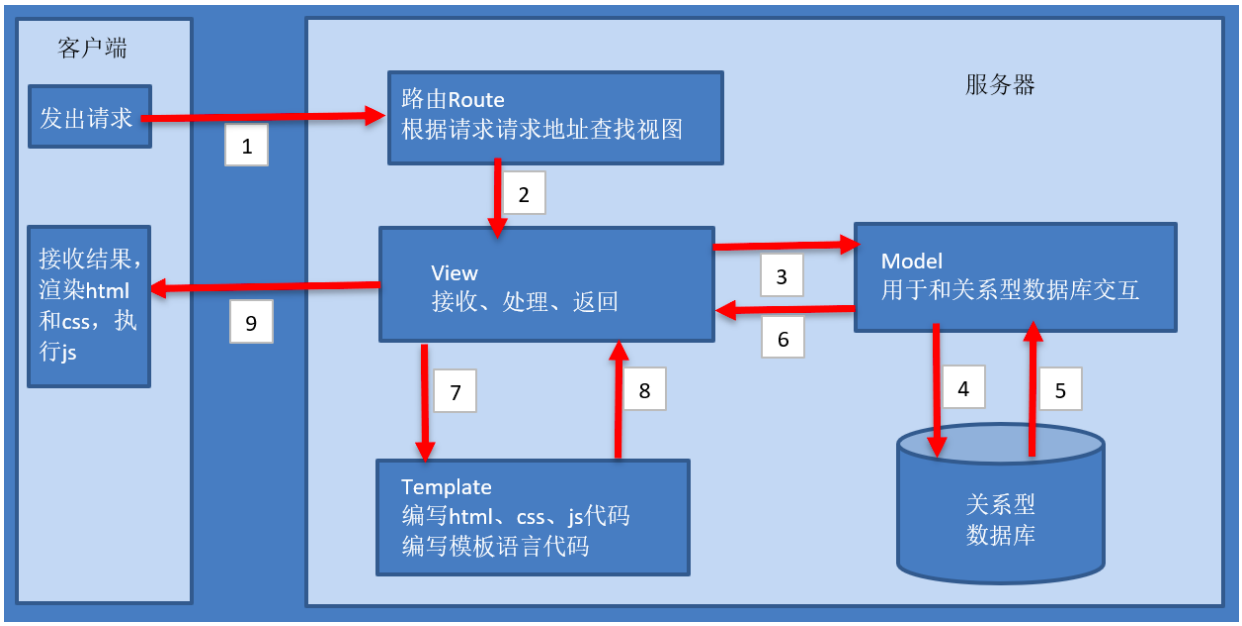
- 核心思想：解耦
- 优点：降低个模块之间的耦合性，方便变更，更容易重构代码，最大程度实现了代码的重用。
- MVC：
    - M：Model，模型，主要封装对数据库层的访问，对数据库中的数据进行增、删、改、查操作。
    - V：View，视图，负责数据的显示和呈现。
    - C：Controller，控制器，负责从用户端收集用户的输入，业务逻辑的处理。



- MTV：

  本质上与MVC没什么差别，也是各组件之间为了保持松耦合关系，只是定义上有些许不同。
    - M：Model，模型，负责业务对象与数据库（ORM）的对象

- T：Template，模板，负责数据的展示效果。
- V：View，视图函数，负责业务逻辑，并在适当的时候调用Model和Template



二、Flask简介

Flask是一个非常小的Python WEB框架，被称为微型框架。只提供了一个高效稳定的核心，其它的全部通过添加扩展来实现。意思就是你可以根据项目的需要进行量身定制，也意味着你需要学习相关的扩展库。

- flask两大核心
  - WerkZueg 实现了路由、调试和Web服务器的网关接口
  - jinja2 模板引擎

三、安装和启动

(1) 命令

在虚拟开发环境中运行

pip install flask

(2) hello Flask

```
from flask import Flask #导入flask类

#flask 类的实例化
app = Flask(__name__)

#flask 路由地址 /
@app.route('/')
def index():
    return 'Hello Flask! '


if __name__ == '__main__':
    app.run()#运行flask
```

浏览器测试:http://127.0.0.1:5000

(3) 启动项run的参数

参数	参数说明
host	主机地址 默认127.0.0.1,设置为'0.0.0.0'之后可以通过IP进行访问。
port	默认端口为5000
debug	调试模式 默认 False,开启后出错有调试信息。
threaded	开启多线程,默认是关闭的。

实例

```
app.run(host='0.0.0.0',port=5001,debug=True,threaded=True)
```

(4) 项目参数配置

- 使用配置文件

```
#settings.py
DEBUG = True

#helloflask.py
app.config.from_pyfile("settings.py")
```

- 直接修改app.config的字典对象

```
app.config['DEBUG'] = True
```

- 使用对象配置参数

```
class Setting:
    DEBUG = True
app.config.from_object(Setting)
```

(5)flask-script插件

简单来说，就是flask终端启动的参数解析器；这样就可以不改代码完成不同方式的启动。

- 安装 `pip install flask-script`
- 用法：

```
from flask import Flask
from flask_script import Manager
app = Flask(__name__)
manager = Manager(app)
。。。。
if name == 'main':
    # app.run(host='0.0.0.0',port=8081,debug=True)
    manager.run()
```

- 启动参数：

```
-?, --help              # 查看帮助
-h, --host              # 指定主机
-p, --port              # 指定端口
-d                      # 开启调试模式
-r                      # 自动加载
```

示例：python manage.py runserver -d -r -h 0.0.0.0 -p 5555

四、路由

(1) 无参路由

```
@app.route('/')
def index():
    return 'hello flask'
```

(2) 带一个参数的路由地址

```
# 带一个参数的视图函数
#访问  127.0.0.1:5000/arg/name
# @app.route('/arg/<name>')
#访问  127.0.0.1:5000/arg/name
#访问  127.0.0.1:5000/arg/name/
@app.route('/arg/<name>/')
def arg(name):
    return '带一个参数的视图函数{}'.format(name)
```

(3) 限制参数值的类型

```
#在访问的时候 只有路由带参 限制不了值的类型 接收以后都为字符串
@app.route('/arg/<name>/') #默认字符串
@app.route('/arg/<int:name>/') #限制值的类型为整形
@app.route('/arg/<float:name>/') #限制值的类型为 浮点形
@app.route('/arg/<path:name>/') #值的类型为字符串 但是路由地址的分隔符/是作为参数值的一部分
def arg(name):
    print(type(name))
    return '带一个参数的视图函数{}'.format(name)
```

(4) 传递多个参数

```
#传递多个参数
@app.route('/args/<name>/<int:age>/')
@app.route('/args/<name>_<int:age>/')
def args(name,age):
    return 'name值为{} age值为{}'.format(name,age)

访问地址:
    http://127.0.0.1:5000/args/zhangsan/18/
    http://127.0.0.1:5000/args/zhangsan_18/
```

（5）HTTP方法

缺省情况下，一个路由只回应 GET 请求。 可以使用 route() 装饰器的 methods 参数来处理不同的 HTTP 方法：

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

注意：

1. 路由地址结尾的/，如果没有/作为结尾 则在访问的时候 路由地址不能有/作为结尾 如果定义的路由有/作为结尾 在访问的时候 可以不添加 但是会有301重定向 所有在访问的时候还是建议添加/
2. 可以使用int/float/string/path 来限制参数值的匹配类型，默认为字符串
3. 视图函数传递多个参数，使用路由地址分割符/或者_进行拼接
4. 视图函数可以有多个路由地址
5. 视图函数 不可以重名

五、重定向 redirect

作用：

通过重定向 实现从一个视图函数跳转到另外的一个视图函数的操作

导入重定向

from flask import redirect,url_for

说明

1. redirect 参数为要跳转的路由地址
2. url_for 函数用于构建指定函数的 URL。它把函数名称作为第一个 参数。它可以接受任意个关键字参数，每个关键字参数对应 URL 中的变量。未知变量将添加到 URL 中作为查询参数。
3. 组合使用 redirect(url_for(endpoint,**values))

实例

要跳转的视图函数

```
@app.route('/')
def index():
    return '无参路由'

@app.route('/arg/<arg1>/')
def arg(arg1):
    return 'arg1的值为%s'%arg1


@app.route('/args/<arg1>/<arg2>/')
def args(arg1,arg2):
    return "arg1的值为{} arg2的值为{}".format(arg1,arg2)
```

重定向的实例代码

```
#这是一个重定向的视图函数
@app.route('/redirect/')
def myRedirect():
    return redirect('/')  #重定向到无参路由
    return redirect('/arg/zhangsan/') #重定向到一个参数的路由地址
    return redirect('/args/zhangsan/18/') #重定向到多个参数的路由地址
    return url_for('index') #输出反向构造出函数index的路由地址
    return url_for('arg',arg1='zhangsan') #/arg/zhangsan/
    return url_for('arg',arg1='zhangsan',name='tom') #/arg/zhangsan/?
name=tom
    return url_for('args',arg1='zhangsan',arg2=18)
#/args/zhangsan/18/
    return url_for('indexx') # Could not build url for endpoint
'indexx'. Did you mean 'index' instead?
    return url_for('args') #Could not build url for endpoint 'args'.
Did you forget to specify values ['arg1', 'arg2']?
    return redirect(url_for('index')) # 反向构造出路由定制并重定向
```

注意：

- url_for根据视图函数名，不是请求路径中名称，反向构造路由地址
- 如果给定的视图函数不存在，或者视图函数有参没有传参，则构造失败（报错）

六、abort 通过抛出给定的状态码的错误信息

导入：from flask import abort

说明：和raise类似 上面的代码正常执行 下面代码不再执行

实例

```
@app.route('/')
def index():
    abort(500)
    abort(404)
```

捕获状态码抛出的异常

```
#捕获500的错误
@app.errorhandler(500)
def server_error(err):
    # return '500错误'
    return '错误信息为{}'.format(err)

#捕获404的错误
@app.errorhandler(404)
def page_not_found(err):
    # return '500错误'
    return '错误信息为{}'.format(err)
```

七、请求 request

概述：

浏览器发送到服务器被flask接收以后 创建出请求对象 request ， 请求对象使用在视图函数中 来获取请求用户所带的数据

request由flask创建 使用的时候 导入就可以

导入：

from flask import request

request属性

属性	说明
url	完整的请求url地址
base_url	去掉传参的路由地址
host_url	只有协议、主机和端口
host	只有ip和端口号
path	请求路径
full_path	请求路径+参数
method	获取请求方式
remote_addr	客户端IP地址
args	获取GET传参。request.args.get(key)获取单个值 request.args.getlist获取多个值
form	获取POST传递过来的数据。request.form.get(key)获取单个值 request.form.getlist获取多个值
values	可以获取GET或POST传参
files	获取文件上传
headers	获取所有的请求头信息
cookies	获取cookie
json	获取请求过来的json数据

八、响应 response

(1) 使用return直接响应字符串

```
@app.route('/')
def index():
    return 'Hello Flask! '
    return 'Hello Flask! ', 404 #响应内容并响应状态码
```

(2) 通过make_response构造响应

导入：

from flask import make_reponse

```
@app.route('/')
def index():
    return make_response('Hello Flask! ')
    return make_response('Hello Flask! ', 404) #响应内容并响应状态码
```

注意：响应状态码默认为200

九、会话控制 COOKIE

在网站中，http请求是无状态的。也就是说即使第一次和服务器连接后并且登录成功后，第二次请求服务器依然不能知道当前请求是哪个用户。cookie的出现就是为了解决这个问题，第一次登录后服务器返回一些数据（cookie）给浏览器，然后浏览器保存在本地，当该用户发送第二次请求的时候，就会把上次请求存储的cookie数据自动的携带给服务器，服务器通过浏览器携带的数据就能判断当前是哪个用户了。cookie存储的数据量有限，不同的浏览器有不同的存储大小，但一般不超过4kb。因此使用cookie只能存储一些小量的数据。

(1) 设置cookie

格式

response.set_cookie(key,value,max_age,expires,path='/')

参数	说明
key	键
value	值
max_age	设置过期时间，单位秒，如果没有设置过期时间，默认为浏览器关闭后过期
expires	设置过期时间，以时间戳的形式设置
path	主域名，默认是'/',表示在当前站点可用

```
@app.route('/set_cookie/')
def set_cookie():
    res = make_response('设置cookie')
    res.set_cookie('name','zhangsan')
    return res
```

注意：默认存活时间为浏览会话结束（关闭浏览器）

(2) 设置cookie并设置过期时间

```
import time
#设置cookie并设置过期时间
@app.route('/set_cookie_lefttime/')
def set_cookie_lefttime():
    res = make_response('设置cookie并设置过期时间')
    # res.set_cookie('name','zhangsan',max_age=40)
    timeout = time.time()+40
    res.set_cookie('name','zhangsan',expires=timeout)
    return res
```

(3) 获取cookie

```
#获取cookie
@app.route('/get_cookie/')
def get_cookie():
    print(request.cookies)
    return 'cookie的值为{}'.format(request.cookies.get('name'))
```

(4) 删除cookie

```
@app.route('/delete_cookie/')
def delete_cookie():
    res = make_response('删除cookie')
    res.delete_cookie('name')
    # res.set_cookie('name','zhangsan',expires=0) #重新设置cookie 刚开始就死了
    return res
```

十、会话控制 session

session和cookie的作用有点类似，都是为了存储用户相关的信息。不同的是，cookie是存储在本地浏览器，而session存储在服务器。存储在服务器的数据会更加安全，不容易被窃取。但存储在服务器也有一定的弊端，就是会占用服务器的资源，但现在服务器已经发展至今，存储一些session信息还是绰绰有余的。

服务器要区分哪一个用户的请求 会根据cookie携带着的唯一sessionid来进行区分 session将数据存储于服务器端 安全性高于cookie

注意：

sessionid需要根据secret_key 来进行生成（如果没有 则报错）

导入：from flask import session

(1) 设置session

```
@app.route('/set_session/')
def set_session():
    session['name'] = 'zhansgan'
    return '设置session'
```

(2) 设置session并设置过期时间

```
@app.route('/session_lefttime/')
def session_lefttime():
    session.permanent = True #开启session存储持久化
    app.permanent_session_lifetime = timedelta(minutes=1) #存活一分钟
    session['name'] = 'zhansgan'
    return '设置session并设置过期时间'
```

(3) 获取session

```
@app.route('/get_session/')
def get_session():
    return '值为{}'.format(session.get('name'))
```

(4) 删除session

```python
@app.route('/delete_session/')
def delete_session():
    # session.pop(key)
    # session.pop('name')
    session.clear()
    return '删除session'
```