flask视图和URL

url_for

进行反转,把函数名字转成URL

```
url_for("函数名字", 参数1, 参数2)

如果参数2,在函数中不存在   会以?形式当做参数
```

- url_for会进行转码
- 修改了URL地址,对咱们的代码影响不大，要比硬编码的方式更友好

url末尾的反斜线

改变端口

```
app.run(debug=True, port=xxx, host=xxx)

host 地址
```

指定HTTP方法

```
@app.route("路由", methods=["请求方法"])


GET请求
request.args.get("xx")

POST请求
request.form.get("xxx")
```

重定向

```
return redirect(url_for("函数名字"))
```

- 301 永久
- 302 暂时

函数的返回值

- 字符串

    - `return "字符串"`

- 元组
    - `return "字符串", 状态码`

- Response
    - `return Response("字符串", "状态码", mimetype="")`

- make_response

模板

模板渲染

先创建一个目录，templates，将模板文件放到目录中，默认查找templates目录

```
return render_template("模板文件名字")
```

指定模板目录

```
template_folder="模板目录"
```

模板传参

```
return render_template("index.html", username="xxxx")
{{ username }}


context = {
    xxx: "xxx"
}
return render_template("index.html", context=context)
{{ context.xxx }}


context = {
    xxx: "xxx"
}
return render_template("index.html", **context)
{{ xxx }}
```

模板过滤器

内置模板过滤器

- abs
- default
- last
- first
- upper
- safe
- lower
- length
- replace
- trim
- format
- int/float/string
- wordcount
- truncate
- striptags

自定义过滤器

```
@app.template_filter('自定义过滤器名字')
```

控制语句

if

```
{% if 判断条件 %}
{% endif %}
```

for

```
{% for xx in xxx %}
{% endfor%}

遍历字典和列表 是有区别的
```

获取循环的状态

```
loop.index      从1
loop.index0     从0
```

宏和import语句

宏

```
{% macro 宏名字(参数) %}
    xxxx
{% endmacro %}
```

import

```
{% import "xxx.html" as xxx %}

{% from "xxx.html" import 宏名字 %}
{% from "xxx.html" import 宏名字 as xxx%}

{% import "xxx.html" as xxx  with context %}    将参数传到宏模板中
```

include和set

include

```
{% include "xxx.html" %}
```

set赋值语句

```
{% set name='xxx' %}        #全局


{% with %}
    {% set name='xxx' %}    # 局部
{% endwith %}

{% with xxx='' %}   # 局部
{% endwith %}
```

模板继承

代码的复用

```
{% extends '父模板.html' %}
```

继承的注意点

1. `{% extends 'base.html' %}` 放到block模块上面
2. 子模板不可以多继承
3. 子模板只能重写父模板中的block，自己定义的block不显示，没有在block模块中的标签也不会显示
4. 调用父模板中的block `{{ super() }}`
5. 父模板中block，是可以嵌套的

加载静态资源文件

创建static目录,静态资源文件放到这个目录下面

```
<link rel="stylesheet" href="{{ url_for('static', filename='css/index.css') }}">
<script src="{{ url_for('static', filename='js/index.js') }}"></script>
<img src="{{ url_for('static', filename='images/demo.jpg') }}" alt="">
```

还可以自定义资源文件的目录

```
static_folder=""  修改static 目录位置
```

案例

思路！！！

视图高级

类视图

标准类视图

```
from flask import views

class XxxXxx(views.View):
    # 这个方法必须实现
    def dispatch_request(self):
        return xxxx

app.add_url_rule("URL", view_func=XxxXxx.as_view('名字'))
```

基于调度方法的类视图

```
from flask import views

class XxxXxx(views. MethodView):
    def get(self):
        pass

    def post(self):
        pass

app.add_url_rule("URL", view_func=XxxXxx.as_view('名字'))
```

蓝图

拆分模块，把模块独立出来

1.创建文件夹 blueprints

2.把模块放到这个目录下面

news.py蓝图文件

```
from flask import Blueprint

xxx = Blueprint("news", __name__)

@xxx.route("/")
def xxx():
    return  xxx
```

app.py主文件

```
from blueprints.news import xxx

app.register_blueprint(xxx)
```

寻找模板文件

```
1.寻找templates 目录下的模板文件
2.如果templates目录下面没有找到，template_folder存在  在template_folder目录下面寻找模板文件
```

寻找静态资源文件

```
# 在static目录下面
<link rel="stylesheet" href="{{ url_for('static', filename='news.css') }}">

# 在蓝图中static目录下
<link rel="stylesheet" href="{{ url_for('news.static', filename='news.css') }}">

news_bp = Blueprint('news', __name__, url_prefix="/news", template_folder="lgcoder",
static_folder='static')
```

子域名

1.创建蓝图

```
from flask import Blueprint
cms_bp = Blueprint("cms", __name__, subdomain="cms")
@cms_bp.route("/")
def cms():
    return "cms页面"
```

2.主app

```
app.config['SERVER_NAME'] = 'xxxx.com:port'
```

修改hosts文件

```
127.0.0.1   xxxx.com
127.0.0.1   cms.xxxx.com
```

数据库

介绍数据库

数据库特点

- 读写速度快
- 持久化存储
- 扩展好
- 数据的有效性

理解数据库

- 表
    - 字段
    - 记录

通过SQLAlchemy连接

安装库

```
pip install pymysql
pip install sqlalchemy
pip instal mysql-connector
```

连接数据库

```
from sqlalchemy import ereate_engine

HOSTNAME = 'xxx'
PORT    = 'xxx'
USERNAME = 'xxx'
PASSWORD = 'xxx'
DATABASE = 'xxx'

DB_URL = 'mysql+pymysql://{}:{}@{}:{}/{}?charset=utf8'.format(USERNAME, PASSWORD, HOSTNAME, PORT,
DATABASES)

engine = create_engine(DB_URL)
```

执行原生SQL语句

```
with engine.connect() as conn:
    conn.execute('原生的SQL语句')
```

介绍ORM

ORM，object relationship mapping

ORM优点

- 安全
- 封装原生的SQL
- 易用，简洁

ORM映射

- 类 -> 表
- 属性 -> 表中的字段
- 对象 -> 一条数据

使用ORM映射到数据库

```
from sqlalchemy import create_engine
from sqlalchemy import Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base

# 地址->小区地址
HOSTNAME = "127.0.0.1"
# 数据库->单元
DATABASE = 'demo0417'
# 端口->门牌号
PORT = 3306
# 用户名和密码
USERNAME = 'root'
PASSWORD = 'root'

# 创建数据库引擎
# dialect+driver://username:password@host:port/database?charset=utf8
```

```
DB_URL = 'mysql+mysqlconnector://{}:{}@{}:{}/{}?charset=utf8'.format(USERNAME, PASSWORD,
HOSTNAME, PORT, DATABASE)

engine = create_engine(DB_URL)
Base = declarative_base(engine)

class User(Base):
        __tablename__ = '表名'

        # 字段的定义
        id = Column(Integer, 主键, 自增)
        # nullable=False 非空  默认是可以为空
        name = Column(String(50), nullable=False)
        # comment 相当对这个字段进行注释
        gender = Column(Integer, default=1, comment="1为男,2为女")
```

ORM的增删改查

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String
from sqlalchemy.orm import sessionmaker


def add():
        user = User(xxxx)
        session.add(user)
        session.add_all([user1, user2])
        session.commit()

def search():
        all()
        first()
        get(id)
        filter(User.name=='xx')
        filter_by(name='xx')

def update():
        先查询
        在修改
        session.commit()
        # 回滚
        session.rollback()

def delete():
        session.delete(对象)
        session.commit()
```

sqlalchemy常用数据类型

- String
- Float
- Boolean
- Time/Date/DateTime
- Enum
- LONGTEXT
- Text
- DECIMAL
- Integer

Column常用参数

- default
- name
- nullable
- autoincrement
- primary_key
- onupdate

聚合函数

```
from sqlalchemy import func

max
min
count
avg
sum

session.query(func.聚合函数(模型.列名)).first()
```

过滤条件

- eq
  - `filter(模型.字段 == 'xxx').all()`
- not eq
  - ```
    filter(模型.字段 != 'xxx').all()
    filter(~模型.字段 == 'xxx').all()
    ```
- and_
    ```

- like
    - ```
      session.query(Article).filter(Article.title.like('%title%')).all()

      %
      _
      title
      ```
- or_
- in_
- notin_
- None
    - Null != 空
- is_

外键及约束

```
from sqlalchemy import ForeignKey

# 表.字段
uid = Column(Integer, ForeignKey('user.id', ondelete='SET NULL'))

ondelete 删除数据的时候要怎么做

# 外键的查询
session.query(User).filter(User.id == Article.uid).first()
```

表关系

一对多

```
# 反向访问的属性
articles = relationship("Article", backref='author')
```

一对一

多对多
```

```
中间表
from sqlalchemy import Table

xxx = Table(
        '表名字',
        Base.metadata,
        Column(外键),
        Column(外键),
)

classes = relationship('Classes', backref='teachers', secondary=xxx)
```

排序

- session.query(模型).order_by(字段).all()
    - 默认排序顺序 升序 asc
    - 降序 desc()
- 在模型中定义
    - ```
      __mapper_args = {
          "order_by": 字段
      }
      查询出来的顺序就是order_by的顺序
      ```

limit offset 切片

```
limit  条数
offset 偏移量
切片  Python层面的切片

# 查询0-2
articles = session.query(Article).offset(0).limit(3).all()
# 查询2-4
articles = session.query(Article).offset(2).limit(3).all()
```

分组和分组之后的筛选

```
group_by
前面查询的字段要和分组的字段保持一致

having
分组之后的筛选  不是对原始的数据集进行查询
```

SQLAlchemy插件

```
安装
pip install flask-sqlalchemy
```
```

```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy(app)

class User(db.Model):
        __tablename__ = 'user'
        id = db.Column(db.Integer, primary_key)
        name = db.Column(db.String(50))

class Article(db.Model):

        uid = db.Column(db.Integer, db.ForeignKey('user.id'))
        author = db.relationship('User', backref='articles')

db.drop_all()
db.create_all()

user = User(name='xxx')
# User.query.all()
# db.session.query(User).all()
```

Flask-Script

安装

```
pip install flask-script
```

在命令行执行自己定义的函数

```
from flask_script import Manager
from xxxx import app

manager = Manager(app)

@manager.command
def index():
        print("hello")
```

给命令行传参数

```
@manager.option("-u", "--name", dest='username')
def info(username):
        print(username)

python manage.py info -u xxxx
```

项目目录

- static
  - 静态资源文件
- templates
  - 模板文件
- blueprint
  - 蓝图
- models
  - 模型
- config
  - 配置文件
- forms
  - 表单验证
- manage
  - 数据库映射文件
- 项目入口文件
  - app.py

映射数据库

```
python manage.py db init

python manage.py db migrate             不会真正的把模型映射到数据库

python manage.py db upgrade
```

表单验证

安装

```
pip install flask-wtf
```

注册表单验证

```
forms.py
from wtforms import Form

class LoginForm(Form):
        username = StringField(validators=[Length(min=3, max=10, message='用户名长度不正确')])
```

常用的验证器

- EqualTo
  - 验证两个变量是否相等
- Email
  - 邮箱
- InputRequired
  - 必须输入
- Regexp
  - 正则表达式
- URL
  - URL格式
- NumberRange
  - 数字范围
- 文件上传
  - FileRequired 文件必须上传
  - FileAllowed 文件允许上传的类型
  - 前端的form表单中 enctype属性
  - ```
    CombinedMultiDict
    ```

cookie 和 session

cookie

HTTP无状态 cookie 记录用户信息的 保存在浏览器中的！！！

```
from flask import Response

@app.route("xxx")
def xxx():
        res = Response("xxx")
        res.set_cookie()

key                     键
value           值
max_age         过期时间   秒
domain       子域名
```

session

保存在服务器中的！！！ 但是flask中 将session加密保存到cookie 保存到浏览器中
```

```
from flask import session

app.config['SECRET_KEY'] = '随机字符串'

@app.route("xxx")
def xxx():
        session['xxx'] = 'xxxx'

session.get('xxx')              #取值

session.pop('xxx')       删除键值对应的session
session.clear()         清空session
```

flask中上下文

请求上下文

request

session

应用上下文

current_app

g对象

钩子函数 hook

```
@app.befroe_first_request               # 处理第一次请求
@app.before_request                             # 每次请求之前


@app.after_request                              # 每次请求之后
def xxx(response):
        return response
@app.teardown_appcontext                # 每次请求之后 不管有没有异常都会执行 debug=False
def xxx(response):
        return response


@app.context_processor                  # 上下文处理器  返回数据到所有的模板中
def xxx(error):
@app.errorhandler(错误状态码)    # 关闭debug模式
def xxx(error):
```