# flask框架

## helloworld和路由定义

```
#1.导入Flask扩展
from flask import Flask
#2.创建Flask应用程序实例
#需要传入__name__
app = Flask(__name__)
#定义路由及视图函数
#路由默认只支持get，如果需要增加，请自行指定
@app.route('/')
def hello_world():  # put application's code here
    return 'Hello World!'

#使用同一个视图函数，来显示不同的用户订单信息
#<>来定义路由的参数，<>内需要起一个名字
@app.route('/orders/<int:order_id>')
def get_order_id(order_id):
    #参数类型默认是字符串
    print(type(order_id))
    #限定为int尖括号加int
    #需要在视图函数（）内填入参数名，后面的代码才能使用
    return 'order_id %s' %order_id
#4.启动程序
if __name__ == '__main__':
    app.run()
```

# 模板

## jinja2模板

```
from flask import Flask,render_template
app=Flask(__name__)
#1.如何返回一个网页（模板）
#2.如何给模板填充数据
@app.route('/')
def index():
    #比如需要传入网址
    url_str1='www.111.com'
    return render_template('index.html',url_str=url_str1)
if __name__ == '__main__':
    app.run()
```

## 使用

注释：{#注释#}（快捷键：ctrl加斜杠/）

### 变量代码块：{{ }}

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
这是模板<br>
这是首页<br>
{#下面是一个代码块的使用#}
{{ url_str }}<br>
{{ my_list }}<br>
{{ my_list.2 }}<br>
{#字典的使用#}
{{ my_dict }}<br>
{{ my_dict.url }}<br>
</body>
</html>
```

```
return render_template('index.html',
                           url_str=url_str,
                           my_list=my_list,
                           my_dict=my_dict)
```

### 控制代码块：{%%}（写for+tab键或者写{%for)

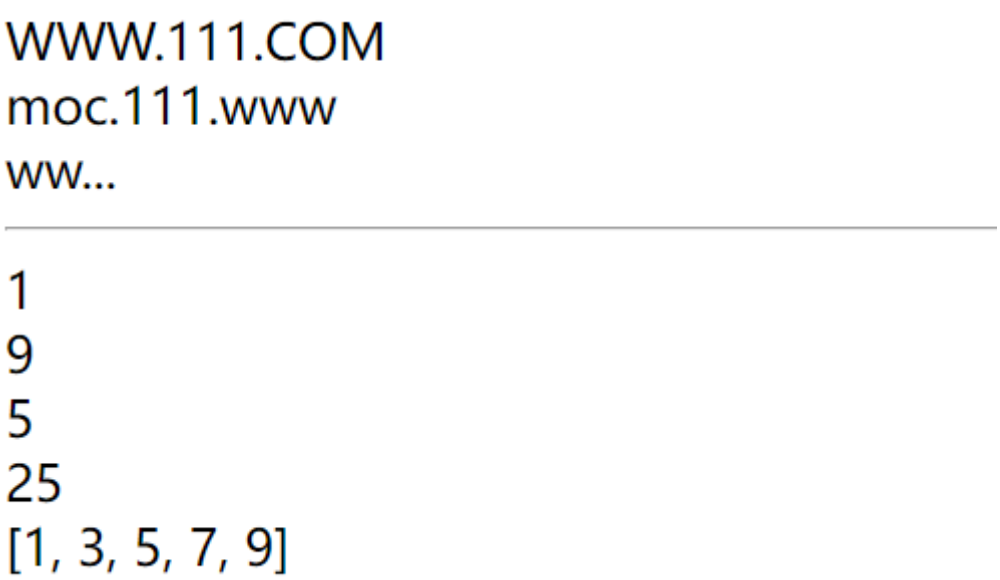
```
{% for my in my_list %}
    {{ my }}<br>
{% endfor %}
```

```
{% for my in my_list %}
{#    数据大于三才显示#}
    {% if my>3 %}
        {{ my }}<br>
    {% endif %}
{% endfor %}
```

## 常见内建过滤器(变量名|过滤器)

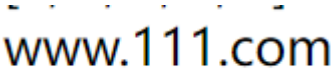
```
<hr>
{#过滤器#}
{#字符串大写#}
{{ url_str |upper }}<br>
{#字符串反转#}
{{ url_str|reverse }}<br>
{#字符串截断#}
{{ url_str|truncate(5) }}<br>
<hr>
{#列表取第一个最后一个元素#}
{{ my_list|first }}<br>
{{ my_list|last }}<br>
{#长度#}
{{ my_list|length }}<br>
{#求和#}
```

```
{{ my_list|sum }}<br>
{#排序#}
{{ my_list|sort }}<br>
```




链式调用：

```
{#过滤器链式调用#}
{{ url_str|upper|reverse|lower|reverse }}<br>
```




# web表单

html的form表单：

```
<form method="post">
    <label>用户名:</label><input type="text" name="username"><br>
    <label>密码:</label><input type="password" name="password"><br>
    <label>确认密码:</label><input type="password" name="password2"><br>
    <input type="submit" value="提交"><br>
    {% for message in get_flashed_messages() %}
        {{ message }}
    {% endfor %}
</form>
```

```
from flask import Flask,render_template,request
app=Flask(__name__)
'''
目的：实现简单的登录的逻辑处理
1.路由需要有get，post请求方式
2.获取请求方式
3.判断参数是否填写&密码是否相同
4.如果判断都没有问题，就返回success
'''
@app.route('/',methods=['GET','POST'])
```

```
def index():
    #request:请求对象-->获取请求方式，数据
    #1.判断请求方式
    if request.method=='POST':
        #2.获取请求方式
        username=request.form.get('username')
        password=request.form.get('password')
        password2=request.form.get('password2')
        #print(username)
        #3.判断参数是否填写&密码是否相同
        if not all([username,password,password2]):
            print('参数不完整')
        elif(password!=password2):
            print('密码不一致')
        else:
            return 'success'
    return render_template('index.html')
if __name__ == '__main__':
    app.run()
```

优化:

```
from flask import Flask,render_template,request,flash
app=Flask(__name__)
app.secret_key='12345'
'''
目的：实现简单的登录的逻辑处理
1.路由需要有get，post请求方式
2.获取请求方式
3.判断参数是否填写&密码是否相同
4.如果判断都没有问题，就返回success
'''
'''
给模板传递消息
flash需要对内容加密，需要设置一个secret_key,做加密消息的混淆
模板中需要遍历消息
'''
@app.route('/',methods=['GET','POST'])
def index():
    #request:请求对象-->获取请求方式，数据
    #1.判断请求方式
    if request.method=='POST':
        #2.获取请求方式
        username=request.form.get('username')
        password=request.form.get('password')
        password2=request.form.get('password2')
        #print(username)
        #3.判断参数是否填写&密码是否相同
        if not all([username,password,password2]):
            #print('参数不完整')
            flash('参数不完整')
        elif(password!=password2):
            #print('密码不一致')
            flash('密码不一致')
        else:
            return 'success'
    return render_template('index.html')
```

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

```
{#使用遍历获取闪现的消息#}
    {% for message in get_flashed_messages() %}
        {{ message }}
    {% endfor %}
```

## 使用Flask-WTF实现表单

```
from flask import Flask,render_template,request,flash
from flask_wtf import FlaskForm
from wtforms import StringField,PasswordField,SubmitField
class LoginForm(FlaskForm):
    username=StringField('用户名:')
    password=PasswordField('密码:')
    password2=PasswordField('确认密码:')
    submit=SubmitField('提交')
@app.route('/form',methods=['GET','POST'])
def login():
    login_from=LoginForm()
    return render_template('index.html',form=login_from)
```

```
<form method="post">
    {{ form.username.label }}{{ form.username }}<br>
    {{ form.password.label }}{{ form.password }}<br>
    {{ form.password2.label }}{{ form.password2 }}<br>
    {{ form.submit }}
</form>
```

# flask使用数据库

## 连接配置:

```
from flask import Flask
from  flask_sqlalchemy import SQLAlchemy
app = Flask(__name__)
DB_URI='mysql+pymysql://root:123456@127.0.0.1:3306/flask_SQL'
app.config['SQLALCHEMY_DATABASE_URI']=DB_URI
app.config['SQLALCHEMY_TRACK_MODIFICATIONS']=False
db=SQLAlchemy(app)
@app.route('/')
def hello_world():
    engine=db.get_engine()
    with engine.connect() as conn:
        result=conn.execute("select 1")
        print(result.fetchone())
    return 'Hello World!'

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

## 定义数据模型:

| 类型名 | python中类型 | 说明 |
| --- | --- | --- |
| Integer | int | 普通整数，一般是32位 |
| SmallInteger | int | 取值范围小的整数，一般是16位 |
| BigInteger | int或long | 不限制精度的整数 |
| Float | float | 浮点数 |
| Numeric | decimal.Decimal | 普通整数，一般是32位 |
| String | str | 变长字符串 |
| Text | str | 变长字符串，对较长或不限长度的字符串做了优化 |
| Boolean | bool | 布尔值 |
| Date | datetime.date | 时间 |

| 选项名 | 说明 |
| --- | --- |
| primary_key | 如果为True，代表表的主键 |
| unique | 如果为True，代表这列不允许出现重复的值 |
| index | 如果为True，为这列创建索引，提高查询效率 |
| nullable | 如果为True，允许有空值，如果为False，不允许有空值 |
| default | 为这列定义默认值 |

```
'''
两张表
角色（管理员，普通用户）
用户（角色id）
'''

#数据库的模型需要继承db.Model
class Role(db.Model):
    #定义表名
    __tablename__='roles'
    #定义字段(db.Column表示是一个字段)
    id=db.Column(db.Integer,primary_key=True)
    name=db.Column(db.String(16),unique=True)
class User(db.Model):
    __tablename__='users'
    id=db.Column(db.Integer,primary_key=True)
    name=db.Column(db.String(16),unique=True)
    #db.ForeignKey(XXX)表示是外键
    role_id=db.Column(db.Integer,db.ForeignKey('roles.id'))
```

## 增删改

终端操作

增：

```
>>> from app import *
>>> role =Role(name='admin')
>>> db.session.add(role)
>>> db.session.commit()
>>> user =User(name='heima',role_id=role.id)
>>> db.session.add(user)
>>> db.session.commit()
```

```
mysql> use flask_sql;
Database changed
mysql> select *from roles;
Empty set (0.00 sec)

mysql> select *from roles;
+----+-------+
| id | name  |
+----+-------+
|  1 | admin |
+----+-------+
1 row in set (0.00 sec)

mysql> select *from users;
+----+-------+---------+
| id | name  | role_id |
+----+-------+---------+
|  1 | heima |       1 |
+----+-------+---------+
1 row in set (0.00 sec)
```

改删：

```
>>> user.name='chengxuyuan'
>>> db.session.commit()
>>> db.session.delete(user)
>>> db.session.commit()
```

```
mysql> select *from users;
+----+-------------+---------+
| id | name        | role_id |
+----+-------------+---------+
|  1 | chengxuyuan |       1 |
+----+-------------+---------+
1 row in set (0.00 sec)

mysql> select *from users;
Empty set (0.00 sec)
```

```
db.session.add(role)      添加到数据库的session中
db.session.add_all([user1, user2]) 添加多个信息到session中
db.session.commit()       提交数据库的修改(包括增/删/改)
db.session.rollback()     数据库的回滚操作
db.session.delete(user) 删除数据库(需跟上commit)
```

查询:

| 过滤器 | 说明 |
| --- | --- |
| filter() | 把过滤器添加到原查询上，返回一个新查询 |
| filter_by() | 把等值过滤器添加到原查询上，返回一个新查询 |
| limit | 使用指定的值限定原查询返回的结果 |
| offset() | 偏移原查询返回的结果，返回一个新查询 |
| order_by() | 根据指定条件对原查询结果进行排序，返回一个新查询 |
| group_by() | 根据指定条件对原查询结果进行分组，返回一个新查询 |

| all() | 以列表形式返回查询的所有结果 |
| --- | --- |
| first() | 返回查询的第一个结果，如果未查到，返回None |
| first_or_404() | 返回查询的第一个结果，如果未查到，返回404 |
| get() | 返回指定主键对应的行，如不存在，返回None |
| get_or_404() | 返回指定主键对应的行，如不存在，返回404 |
| count() | 返回查询结果的数量 |
| paginate() | 返回一个Paginate对象，它包含指定范围内的结果 |

# 综合案例-图书管理

| Author(作者表) | | | |
| --- | --- | --- | --- |
| id | name | books | |
| 1 | 黑马程序员 | python入门/falsk入门 | |
| | | | |
| **Book(书籍表)** | | | |
| id | name | author_id(外键) | author（引用） |
| 1 | python入门 | 1 | author.name=黑马程序员 |
| 2 | flask入门 | 1 | |

```
from flask import Flask,render_template,flash,request,redirect
from flask_sqlalchemy import SQLAlchemy
from flask_wtf import FlaskForm
```

```
from wtforms import StringField,SubmitField
from wtforms.validators import DataRequired
import pymysql
app = Flask(__name__)
DB_URI='mysql+pymysql://root:123456@127.0.0.1:3306/flask_sql'
app.config['SQLALCHEMY_DATABASE_URI']=DB_URI
app.config['SQLALCHEMY_TRACK_MODIFICATIONS']=False
app.secret_key='111'
db=SQLAlchemy(app)
'''
1.配置数据库
2.添加书和作者的模型
3.添加数据
4.使用模板显示数据库查询的数据
5.使用WTF显示表单
6.实现相关增删逻辑
'''

#定义书和作者模型
class Author(db.Model):
    __tablename__='authors'
    id=db.Column(db.Integer,primary_key=True)
    name=db.Column(db.String(16),unique=True)
    #books给自己（Author模型）用的，author是给Book模型用的
    books=db.relationship('Book',backref='author')

class Book(db.Model):
    __tablename__='books'
    id=db.Column(db.Integer,primary_key=True)
    name=db.Column(db.String(16),unique=True)
    author_id=db.Column(db.Integer,db.ForeignKey('authors.id'))
class AuthorForm(FlaskForm):
    author=StringField('作者',validators=[DataRequired()])
    book=StringField('书籍',validators=[DataRequired()])
    submit=SubmitField('提交')
db.drop_all()
db.create_all()
# 生成数据
au1 = Author(name='Author1')
au2 = Author(name='Author2')
au3 = Author(name='Author3')
# 把数据提交给用户会话
db.session.add_all([au1, au2, au3])
# 提交会话
db.session.commit()
bk1 = Book(name='Book1', author_id=au1.id)
bk2 = Book(name='Book2', author_id=au1.id)
bk3 = Book(name='Book3', author_id=au2.id)
bk4 = Book(name='Book4', author_id=au3.id)
bk5 = Book(name='Book5', author_id=au3.id)
# 把数据提交给用户会话
db.session.add_all([bk1, bk2, bk3, bk4, bk5])
# 提交会话
db.session.commit()


#删除作者
@app.route('/delete_author/<author_id>')
```

```
def delete_author(author_id):
    author=Author.query.get(author_id)
    if author:
        try:
            #查询后直接删除
            Book.query.filter_by(author_id=author_id).delete()
            db.session.delete(author)
            db.session.commit()
        except Exception as e:
            print(e)
            flash('删除作者出错')
            db.session.rollback()
    else:
        flash('作者找不到')
    return redirect('http://127.0.0.1:5000/')
'''
删除书籍-->网页中删除-->点击需要发送书籍的ID给删除书籍的路由-->路由需要接收参数
'''
@app.route('/delete_book/<book_id>')
def delete_book(book_id):
    #查询数据库是否有该ID的书，如果有就删除，没有提示错误
    book=Book.query.get(book_id)
    if book:
        try:
            db.session.delete(book)
            db.session.commit()
        except Exception as e:
            print(e)
            flash('删除书籍出错')
            db.session.rollback()
    else:
        flash('书籍找不到')
    #返回当前网址-->重定向
    return redirect('http://127.0.0.1:5000/')

@app.route('/',methods=['GET','POST'])
def hello_world():
    #创建自定义的表单类
    author_form=AuthorForm()
    '''
    验证逻辑
    1.调用WTF的函数实现验证
    2.验证通过获取数据
    3.判断作者是否存在
    4.如果作者存在，判断书籍是否存在，没有重复书籍就添加数据，如果重复就提示错误
    5.如果作者不存在，添加作者和书籍
    6.验证不通过就提示错误
    '''
    if author_form.validate_on_submit():
        #获取数据
        author_name=author_form.author.data
        book_name=author_form.book.data
        #判断作者是否存在
        author=Author.query.filter_by(name=author_name).first()
        if author:
            book=Book.query.filter_by(name=book_name).first()
            if book:
                flash('已存在同名书籍')
```

```
            else:
                try:
                    new_book=Book(name=book_name,author_id=author.id)
                    db.session.add(new_book)
                    db.session.commit()
                except Exception as e:
                    flash('添加书籍失败')
                    db.session.rollback()
        else:
            try:
                new_author=Author(name=author_name)
                db.session.add(new_author)
                db.session.commit()
                new_book=Book(name=book_name,author_id=new_author.id)
                db.session.add(new_book)
                db.session.commit()
            except Exception as e:
                print(e)
                flash('添加作者和书籍失败')
                db.session.rollback()
    else:
        if request.method=='POST':
            flash('参数不全')

    #查询所有作者信息传给模板
    authors=Author.query.all( )
    return render_template('books.html',authors=authors,form=author_form)
if __name__ == '__main__':
    app.run(debug=True)
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form method="post">
    {{ form.csrf_token() }}
    {{ form.author.label }}{{ form.author }}
    {{ form.book.label }}{{ form.book }}
    {{ form.submit }}
{#    显示消息闪现的内容#}
    {% for message in get_flashed_messages() %}
        {{ message }}
    {% endfor %}

</form>

<hr>
{#先遍历作者，在作者里遍历书籍#}
<ul>
    {% for author in authors %}
        <li>{{ author.name }}<a href="{{
url_for('delete_author',author_id=author.id) }}">删除</a></li>
        <ul>
        {% for book in author.books %}
```

```
            <li>{{ book.name }}<a href="{{
url_for('delete_book',book_id=book.id) }}">删除</a> </li>
            {% else %}
            <li>无</li>
        {% endfor %}
        </ul>
    {% endfor %}

</ul>
</body>
</html>
```