FLASK

1. 蓝图和类视图

```python
from flask_restful import Resource


class UserResource(Resource):
    def get(self):
        return {'method': 'get'}

    def post(self):
        return {'method': 'post'}
```

只需要定会类视图

```python
# 1. 创建蓝图对象 Blueprint(...)
# 2. 通过蓝图对象创建对应的组件对象 Api(蓝图对象)
# 3. 组件对象添加类视图 api.add_resource(类视图, 路径)

from flask import Blueprint
from flask_restful import Api

from user.views import UserResource

user_blue = Blueprint('user', __name__, url_prefix='/user')

user_api = Api(user_blue)
user_api.add_resource(UserResource, '/index')  # /user/index
```

2. 请求参数解析

```python
def post(self):
    # 创建请求解析器
    parser = RequestParser()
    # 添加参数规则
    # 解析 name 和 age
    parser.add_argument('name')
    parser.add_argument('age')
    # 执行解析
    args = parser.parse_args()

    # 读取请求数据
    print(args.name)
    print(args.age)
    return {'method': 'get'}
```

2.1 请求参数解析细节

```python
# TODO:
# 1. 创建请求解析器 parser = RequestParser()
# 2. 添加参数规则 parser.add_argument(参数名, default=默认值, required=是否必传, location=参数从什么地方读取)
# 3. 执行解析  args = parser.parse_args()
#       默认会从 查询字符串/post键值对/post-json数据 进行参数提取
#       如果参数解析失败，会自动返回 400 错误
#       否则返回解析出来后的数据
# 4. 获取参数 args.参数名

# 5. 自定义参数校验函数
#     这个函数需要接受一个参数，这个参数是原始的请求参数
#     如果参数校验成功，应当返回一个值
#     如果参数校验失败，应当抛出一个 ValueError 异常

def convert_datetime(value):
    # "2010-10-10 10:10:10"  化为 python datetime 对象,
    try:
        return datetime.strptime(value, "%Y-%m-%d %H:%M:%S")
    except Exception:
        raise ValueError('参数格式不正确，应当是: %Y-%m-%d %H:%M:%S')
```

3. sqlalchemy

4. 介绍

5. 基本配置和组件初始化

- 数据库URI(连接地址)格式: `协议名://用户名:密码@数据库IP:端口号/数据库名` , 如:

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql://root:123456@127.0.0.1:3306/flask'
```

- 注意点:
  - 如果数据库驱动使用的是 **pymysql**, 则协议名需要修改为 `mysql+pymysql://xxxxxxx`
  - `sqlalchemy` 支持多种关系型数据库, 其他数据库的URI可以查阅 官方文档

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)

# 应用配置
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql://root:123456@127.0.0.1:3306/flask'
app.config['SQLALCHEMY_ECHO'] = True

# 方式1: 初始化组件对象, 直接关联Flask应用
db = SQLAlchemy(app)
```

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy


# 方式2: 初始化组件对象, 延后关联Flask应用
db = SQLAlchemy()


def create_app(config_type):
    """工厂函数"""

    # 创建应用
    flask_app = Flask(__name__)
    # 加载配置
    config_class = config_dict[config_type]
    flask_app.config.from_object(config_class)

    # 关联flask应用
    db.init_app(app)

    return flask_app
```

7. 模型类定义

```
# TODO:
# 1. 配置数据库连接
# 相关配置
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://root:123456@127.0.0.1:3306/flask'
app.config['SQLALCHEMY_ECHO'] = True

# 2. 实例化 SQLAlchemy 对象 db = SQLAlchemy(app)

db = SQLAlchemy(app)
```

```
# 3. 定义模型类 class XXX(db.Model)
class User(db.Model):  # user 表名
    #     - __tablename__ = 't_user'  # 设置表名, 表名默认为类名小写
    __tablename__ = 'tb_user'
    id = Column(Integer, primary_key=True)  # 设置 id 为主键
    name = Column(String(32))
    age = Column(Integer, default=0)


#     - 字段名 = Column(字段类型,
```

```
if __name__ == '__main__':
    # 删除所有继承自db.Model的表 db.drop_all()
    db.drop_all()
    # 创建所有继承自db.Model的表 db.create_all()
    db.create_all()
    # 启动 app
    app.run(host='0.0.0.0', port=8000, debug=True)
```

8. 增加数据

9. 查询数据介绍

```
def create_user():
    #    1. 模型类实例化
    user = User(name='zhangsan', age=100)
    user1 = User(name='lisi', age=1000)
    db.session.add_all([user, user1])
    #    2. db.session.add(模型类对象)
    # db.session.add(user)
    #       db.session.add_all(模型类对象列表) 可以一次性添加多条数据
    #    3. 提交事务 db.session.commit()
    db.session.commit()
    return 'create user'
```

sqlalchemy 会自动开启事务，在事务中执行 sql 语句

10. 查询数据演示

11. 数据更新

```python
# 4. 服务启动时创建用于测试的数据
# 5. 定义接口
#    1. 查询数据
#    2. 更新属性值
#    3. 提交事务
@app.route('/update/goods')
def update_goods():
    goods = Goods.query.filter(Goods.name == '方便面').first()1 # 可能为 None
    if goods:
        goods.count = goods.count - 1
        db.session.commit()2

    return 'update goods'


# 定义接口，查询数据同时进行更新
# db.session.query(模型类).filter(过滤条件).update({要更新的字典: 更新的值})
# db.session.commit()
@app.route('/update_goods')
def update_goods_2():
    db.session.query(Goods).filter(Goods.name == '方便面').update({'count': Goods.count - 1})
    db.session.commit()
    return 'update goods 2'
```

12. 删除数据

```python
# TODO:
# 定义接口：先查后删除
#    1. 查询要删除的数据
#    2. 调用 db.session.delete(数据) 表示要删除数据
#    3. 提交会话 db.session.commit()

@app.route('/delete/goods')
def delete_goods():
    goods = db.session.query(Goods).filter(Goods.name == '方便面').first()1
    if goods:
        db.session.delete(goods)2
        db.session.commit()3

    return 'delete goods'


# TODO:
# 定义接口：基于过滤条件的删除
#     1. db.session.query(模型类).filter(过滤条件).delete()
#     2. 提交会话
@app.route('/delete_goods')
def delete_goods_2():
    db.session.query(Goods).filter(Goods.name == '方便面').delete()4
    db.session.commit()5
    return 'delete goods 2'
```

0. 练习题

自定义手机号解析函数

需求:

- 通过 RequestParser 校验请求参数
- 客户端传递一个 mobile 参数，通过自定义解析函数，解析这个参数

定义商品模型类

需求:

- 定义多个字段
    - name: str
    - price: float
    - description: str
    - count: int
    - tag: str
- 创建 30 条商品信息
    - 每一条数据的 tag 在 手机、电脑、硬盘、网卡 中随机选择一个
    - 10 条数据 price 在 1 到 10000 之间随机
    - 10 条数据 count 在 -10 到 1000 之间随机
    - 10 条数据 description 包含"手机"
- 查询 price 大于 1000 的商品
- 找出 price 最大的商品和最小的商品
- 将 count 为负数的数据更新为 0
- 按照 tag 分组统计每一 tag 的商品数量
- 找出 description 包含手机的商品