03-模型基础

模型Model

Flask模型

```
Flask默认并没有提供任何数据库操作的API
我们可以选择任何适合自己项目的数据库来使用
Flask中可以自己的选择用原生语句实现功能，也可以选择ORM（SQLAlchemy，MongoEngine）

原生SQL缺点
  代码利用率低，条件复杂代码语句越长，有很多相似语句
  一些SQL是在业务逻辑中拼出来的，修改需要了解业务逻辑
  直接写SQL容易忽视SQL问题
```

ORM

```
  Flask通过Model操作数据库，不管你数据库的类型是MySql或者Sqlite，Flask自动帮你生成相应数据库类型的SQL语句，所以不需要关注SQL语句和类型，对数据的操作Flask帮我们自动完成。只要会写Model就可以了。

    Flask使用对象关系映射（Object Relational Mapping，简称ORM）框架去操控数据库。

  ORM(Object Relational Mapping)对象关系映射，是一种程序技术，用于实现面向对象编程语言里不同类型系统的数据之间的转换。

将对对象的操作转换为原生SQL
优点
  易用性，可以有效减少重复SQL
  性能损耗少
  设计灵活，可以轻松实现复杂查询
  移植性好
```

Flask的ORM

```
Flask使用Python自带的ORM：SQLAlchemy

针对于Flask的支持,安装插件 flask-sqlalchemy

pip install flask-sqlalchemy
```

连接SQLite

```
SQLite连接的URI：
  DB_URI = sqlite:///sqlite3.db
```

连接MySQL

```
USERNAME='root'
PASSWORD='root'
HOSTNAME = 'localhost'
PORT='3306'
DATABASE='HelloFlask'

格式：  mysql+pymysql://USERNAME:PASSWORD@HOSTNAME:PORT/DATABASE

# 配置URL
DB_URI='mysql+pymysql://{}:{}@{}:{}/{}'.format(
  USERNAME,
  PASSWORD,
  HOSTNAME,
  PORT,
  DATABASE
)
```

在Flask中使用ORM

```
连接数据库需要指定配置
  app.config['SQLALCHEMY_DATABASE_URI'] = DB_URI  # 配置连接数据库路径DB_URI
  app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False  # 禁止对象追踪修改

SQLite数据库连接不需要额外驱动，也不需要用户名和密码

# 在Flask项目中使用
db = SQLAlchemy()
db.init_app(app)
```

创建模型

```
class Person(db.Model):
  __tablename__='person'
  id = db.Column(db.Integer,primary_key=True)
  name = db.Column(db.String(16),unique=True)

字段类型
    Integer
    Float
    String

常用约束
  primary_key
```

```
    autoincrement
    unique
    default

数据简单操作
    创建数据库，表
        db.create_all()
    删除表
        db.drop_all()

    在事务中处理，数据插入
        db.session.add(object)
        db.session.commit()

    获取所有数据
      Person.query.all()
```

数据迁移

```
安装
  pip install flask-migrate

初始化
  使用app和db进行migrate对象初始化
      from flask_migrate import Migrate
    migrate = Migrate()
      migrate.init_app(app=app, db=db)

数据迁移命令：
  在cmd或Terminal先进入项目目录：
  然后输入命令：
  flask db init  创建迁移文件夹migrates，只调用一次
    flask db migrate  生成迁移文件
    flask db upgrade  执行迁移文件中的升级
    flask db downgrade  执行迁移文件中的降级
```

模型操作

创建模型

```python
# 模型：类
class Person(db.Model):
    __tablename__ = 'person'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(20), unique=True)
    age = db.Column(db.Integer, default=1)
```

字段类型

类型名	Python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	定点数
String	str	变长字符串
Text	str	长字符串,对较长或不限长度的字符串做了优化
Unicode	unicode	变长 Unicode 字符串
UnicodeText	unicode	变长 Unicode 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	日期
Time	datetime.time	时间
DateTime	datetime.datetime	日期和时间
Interval	datetime.timedelta	时间间隔
LargeBinary	str	二进制文件

常用约束

选项名	说 明
primary_key	如果设为 True ,这列就是表的主键
unique	如果设为 True ,这列不允许出现重复的值
index	如果设为 True ,为这列创建索引,提升查询效率
nullable	如果设为 True ,这列允许使用空值;如果设为 False ,这列不允许使用空值
default	为这列定义默认值

模型操作

单表操作

增加数据

a. 一次增加一条数据：

```
p = Person()
p.name = '小明'
p.age = 22

try:
    db.session.add(p)
    db.session.commit()
except:
    # 回滚
    db.session.rollback()
    db.session.flush()
```

b. 一次添加多条数据

```
persons = []
for i in range(10,30):
    p = Person()
    p.name = '宝强' + str(i)
    p.age = i
    persons.append(p)
db.session.add_all(persons)
db.session.commit()
```

删除数据

```
p = Person.query.first()  # 获取第一条数据
db.session.delete(p)
db.session.commit()
```

修改数据

```
p = Person.query.first()
p.age = 100
db.session.commit()
```

查询数据

过滤器

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() 使用指定的值限制原查询返回的结果数量,返回一个新查询

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

常用查询

all() 以列表形式返回查询的所有结果,返回列表

first() 返回查询的第一个结果,如果没有结果,则返回 None

first_or_404() 返回查询的第一个结果,如果没有结果,则终止请求,返回 404 错误响应

get() 返回指定主键对应的行,如果没有对应的行,则返回 None

get_or_404() 返回指定主键对应的行,如果没找到指定的主键,则终止请求,返回 404 错误响应

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

查询属性

contains

startswith

endswith

in_

__gt__

__ge__

__lt__

__le__

逻辑运算

与 and_

filter(and_(条件),条件…)

或 or_

filter(or_(条件),条件…)

非 not_

filter(not_(条件),条件…)

示例:

查询:

```
persons = Person.query.all()  # 获取所有
persons = Person.query.filter(Person.age>22)

# filter功能比filter_by强大
persons = Person.query.filter(Person.age==22)  # filter(类.属性==值)
persons = Person.query.filter_by(age=22) # filter_by(属性=值)

persons = Person.query.filter(Person.age.__lt__(22)) # <
```

```
            persons = Person.query.filter(Person.age.__le__(22)) # <=
            persons = Person.query.filter(Person.age.__gt__(22)) # >
            persons = Person.query.filter(Person.age.__ge__(22)) # >=

            persons = Person.query.filter(Person.age.startswith('宝'))  # 开头匹配
            persons = Person.query.filter(Person.age.endswith('宝'))  # 结尾匹配
            persons = Person.query.filter(Person.age.contains('宝'))  # 包含
            persons = Person.query.filter(Person.age.in_([11,12,22]))  # in_

            persons = Person.query.filter(Person.age>=20, Person.age<30)  # and_
            persons = Person.query.filter(and_(Person.age>=20, Person.age<30))  # and_
            persons = Person.query.filter(or_(Person.age>=30, Person.age<20))  # or_
            persons = Person.query.filter(not_(Person.age<30))  # not_
```

排序：

```
            persons = Person.query.order_by('age')  # 升序
            persons = Person.query.order_by(desc('age'))  # 降序
```

分页：

```
            persons = Person.query.limit(5)  # 取前5个
            persons = Person.query.offset(5)  # 跳过前5个

            # 获取页码page和每页数量num
            page = int(request.args.get('page'))
            per_page = int(request.args.get('per_page'))

            # 手动做分页
            persons = Person.query.offset((page-1) * per_page).limit(per_page)

            # 使用paginate做分页
            persons = Person.query.paginate(page=page, per_page=per_page,
error_out=False).items
```

paginate对象的属性：

- items：返回当前页的内容列表
- has_next：是否还有下一页
- has_prev：是否还有上一页
- next(error_out=False)：返回下一页的Pagination对象
- prev(error_out=False)：返回上一页的Pagination对象
- page：当前页的页码（从1开始）
- pages：总页数
- per_page：每页显示的数量
- prev_num：上一页页码数
- next_num：下一页页码数
- total：查询返回的记录总数

掌握

1. 熟练掌握Flask模型使用
2. 掌握数据迁移
3. 掌握模型常用的字段类型和常用约束
4. 掌握模型单表操作
5. 创建User表，字段:id，name, passwd, age
    1. 实现注册功能
    2. 实现登录功能