Flask数据库基本操作-SQLAlchemy

创建时间:2018年12月19日14时52分

**Flask数据库**  
D：使用扩展包flask-sqlalchemy来操作数据库（增删改查）  
E：通过 Python 对象来操作数据库，在舍弃一些性能开销的同时，换来的是开发效率的较大提升  
U：**常用的SQLAlchemy字段类型**

|  |  |  |
| --- | --- | --- |
| 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 |
| 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 | 时间 |
| Time | datetime.datetime | 日期和时间 |
| LargeBinary | str | 二进制文件 |

**常用的SQLAlchemy列选项**

|  |  |
| --- | --- |
| 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 |

**常用的SQLAlchemy关系选项**

|  |  |
| --- | --- |
| 选项名 | 说明 |
| backref | 在关系的另一模型中添加反向引用 |
| primary join | 明确指定两个模型之间使用的联结条件 |
| uselist | 如果为False，不使用列表，而使用标量值 |
| order\_by | 指定关系中记录的排序方式 |
| secondary | 指定多对多中记录的排序方式 |
| secondary join | 在SQLAlchemy中无法自行决定时，指定多对多关系中的二级联结条件 |

**①安装扩展包及导包**  
安装flask-sqlalchemy：pip install flask-sqlalchemy  
如果连接的是mysql数据，需安装mysqldb：pip install flask-sqlalchemy  
导包：from flask\_sqlalchemy import SQLAlchemy  
**②配置相关数据库的设置**  
#数据库信息设置  
app.config['SQLALCHEMY\_DATABASE\_URI'] = 'mysql://root:mysql@127.0.0.1:3306/数据库名'  
# 动态追踪修改设置，如未设置只会提示警告，极大影响mysql性能app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False  
**③创建连接数据库的对象**  
db = SQLAlchemy(app)  
**④定义模型类，继承db.Model**  
**定义数据库的表名：**\_\_tablename\_\_     
**设置字段：** 字段=db.Column(db.字段类型，字段选项)     
     例如:id = db.Column(db.Integer, primary\_key=True)  
      name = db.Column(db.String(64), unique=True)  
**设置关联属性：**books = db.relationship('Book', backref='author', lazy='dynamic')  
给该模型类添加一个属性，第一个参数为多类类名，通过这个属性可以查询一对多所有对象  
第二个参数backref =’该类类名小写‘，是反向给多类申明一个新属性  
第三个参数指定是lazy属性，即何时加载数据，dynamic指的是在访问属性的时候，并没有在内存中加载数据，而是返回一个query对象, 需要执行相应方法才可以获取对象，比如.all()  
**⑤无迁移式的建表和删表**  
db.drop\_all() 删除该数据库所有的表             db.create\_all() 在该数据库下创建所有模型类映射的表  
  
**增删改查**  
增：①创建对象：  b =Book（name='图书'）  
       ②把数据添加到用户会话：db.session.add(b)        
    如果多个对象，使用db.session.add\_all([b1,b2,b3])  
③提交用户会话到数据库：  db.session.commit()  
删：第一种方式：  
①查出该对象：     b=Book.query.get(1)  
②从用户会话删除该对对象：    db.session.delete(b)  
③提交用户会话：  db.session.commit()   
第二种方式：  
①查出对象直接删除：  Book.query.get(1).delete()  
②提交用户会话：  db.session.commit()   
改：①查出该对象：      b=Book.query.get(1)  
②修改对象属性：  b.name='小说'  
③提交用户会话：  db.session.commit()   
查：①无条件查询：Book.query.查询执行器  
②条件查询：Book.query.过滤器.查询执行器  
常用过滤器如下：

|  |  |  |
| --- | --- | --- |
| 过滤器 | 说明 | 示例 |
| filter(条件) | 返回符合该条件的查询集，BaseQuery对象 | Book.query.filter(Book.id==1) |
| filter\_by() | 返回符合该等值条件的查询集 | Book.query.filter\_by(id=1) |
| 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，返回一个新查询集 |  |
| order\_by() | 根据字段进行排序，默认是正序，返回新查询集,desc可以反序 | Book.query.order\_by(Book.id) Book.query.order\_by(Book.id.desc) |
| group\_by() | 根据字段进行分组，返回新查询集合 |  |

常用查询执行器如下：

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 示例 |
| all() | 以列表形式返回查询的所有结果 | Book.query.filter(Book.id==1).all() |
| first() | 返回查询的第一个结果，如果未查到，返回None | Book.query.filter(Book.id==1).first() |
| 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 | Book.query.get(1) |
| 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 | Book.query.count() |
| paginate() | 返回一个Paginate对象，它包含指定范围内的结果,参数一：第几页，参数二：每页个数，参数3：如果没有该页数返回False | Book.query.paginate(2,3,False) |

**逻辑非，逻辑与，逻辑或**  
from sqlalchemy import not\_,and\_,or\_  
示例：User.query.filter(not\_(User.name=='chen')).all()  
              User.query.filter(and\_(User.name!='wang',User.email.endswith('163.com'))).all()  
  
**一对多，多对一关联查询**  
**一对多：①先查询出一类对象，**例如author=Author.query.get(1)  
**②根据我们设置的relationship属性获取这一类对象下的全部多类对象：**  
books=author.books                  (即该作者下全部书籍)  
**多对一：①查询出多类对象，**例如book=Book.query.get(2)  
**②根据我们设置的backref反向设置的属性获取该多类对象对应的一类对象：**  
author =book.author                 (即这本书所属的作者）  
  
  
**flask数据库迁移**  
D：在数据库中建立模型类映射的数据库表，如果需要修改数据库模型，还要在修改之后更新数据库，最好的解决的方法使用数据库迁移框架Flask-Migrate  
E：建立相关数据库表，而且追踪数据库模式的变化，然后把变动应用到数据库中，还可以回退版本。  
U：**迁移步骤：**  
**①安装扩展包：**迁移扩展包：pip install flask-migrate，脚本管理器包：pip install flask-script  
**②导包：**from flask\_migrate import Migrate,MigrateCommand                      from flask\_script import Shell,Manager  
**③创建脚本管理器：**manager= Manager（app）  
**④迁移关联应用和数据库：**Migrate（app，db）  
**⑤添加迁移命令道脚本管理器：**manager.add\_command('db',MigrateCommand)  
**⑥使用脚本命令在命令行进行迁移操作：**

|  |  |
| --- | --- |
| 创建迁移文件夹 | python xxx.py db init |
| 生成当前版本迁移文件 | python xxx.py db migrate -m '版本说明' |
| 执行当前本迁移文件 | python xxx.py db upgrade |
| 回退一个迁移版本 | python xxx.py db downgrade |
| 查看迁移历史记录 | python xxx.py db history |
| 向上迁移到指定版本 | python xxx.py db upgrade 版本号 |
| 向下迁移到指定版本 | python xxx.py db downgrade 版本号 |
| 查看当前迁移版本 | python xxx.py db current |