# 课程介绍

* 1. Flask-SQLAlchemy的使用;（掌握）
* 2. 常用查询;（掌握）
* 3. 表的关系；（了解）

1. Flask-SQLAlchemy
   1. Flask-SQLAlchemy是什么

Flask里面没有原生的orm，需要用到第三方的库，大名顶顶的SQLALchemy正是一类实现ORM的库，而flask-sqlalchemy是在SQLAlchemy基础上封装的一套更方便Flask框架使用的ORM系统。

使用ORM的好处：可以让我们操作数据库跟操作对象是一样的，非常方便。因为一个表就抽象成一个类，一条数据就抽象成该类的一个对象。

学习时，建议配合官网手册进行查询。

* + 1. 安装

依然使用pip来进行安装：

pip install Flask\_SQLAlchemy

可以看到安装了2个库：Flask-SQLAlchemy-2.3.2 SQLAlchemy-1.2.9

* + 1. 使用

实例化SQLAlchmey对象，把该对象注册到app里即可。

如果小项目，就可以使用官网的例子的效果，在一个py文件里把数据库对象和app对象写在一个文件，然后进行注册绑定。大型项目，建议还是分层处理。

from flask import Flask

from flask\_sqlalchemy import SQLAlchemy

app = Flask(\_\_name\_\_)

db = SQLAlchemy(app=app)

在实例化时，必须加入app的配置。

* + 1. 配置数据库引擎

所有ORM的操作，都需要告知数据库引擎，才能对数据库进行表的映射操作。SQLAlchemy可以支持很多数据库，具体的数据库配置可以参考：

http://flask-sqlalchemy.pocoo.org/2.3/config/

他的通用模板类似：

dialect+driver://username:password@host:port/database

dialect数据库实现 ：

postgresql mysql mariadb oracle sqlite

例如sqlite数据库的配置：



其他数据库需要配置driver：

根据不同的数据库，选择相关的数据库驱动即可，由于各个系统在兼容性上容易出现问题，建议根据错误信息进行定位，查找适合的驱动程序。

例如mysql，驱动程序可选：

mysqlclient

cymysql

若非MysqlDB方式的话，需要在driver中配置：



* + 1. 同步映射表

一旦配置成功了数据库驱动引擎，并且也注册到app后，就可以直接调用create\_all()方法对数据库进行同步映射表。

在shell中导入db对象，然后就可以调用db.create\_all方法了。

此时会看到数据库里有对应的表产生了。

>>> from yourapplication import db

>>> db.create\_all()

* + 1. 数据的增

可以在shell中先熟悉SQLAlchemy的接口操作，这样后面在代码中，操作是一样的。

增加：

>>> from yourapplication import User

>>> u1 = User(username='jim)

>>> db.session.add(u1)

>>> db.session.commit()

增加的流程：

* + - 1. 产生一个python对象，ORM的类对象
      2. 把他加入到数据库的session里
      3. 提交这个数据库session;
    1. 数据的查

每个数据模型都有query接口可以用来查询模型所对应的表的记录。

比如，查询”user”表中的所有记录：

users = User.query.all()

返回的users是一个列表，其中每个元素都是一个User类型的对象，对应于”user”表中的一条记录。

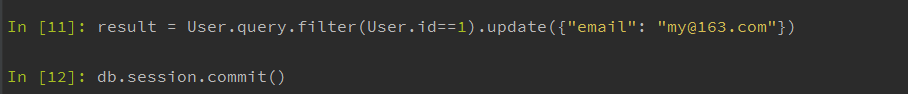
query接口还支持“filter()”，“filter\_by()”等条件操作。后面来学习。

* + 1. 删除和更新

更新数据：

先找到记录，然后直接操作属性后，commit提交。

同时也提供了update方法来批量更新记录。



同样也可以调用.delete()方法进行批量删除，只是最后不要忘记commit就可以。

如果想删除一条记录，那么可以使用

>>> db.session.delete(me)

>>> db.session.commit()

* 1. 定义模型

SQLAlchemy使用Column类来定义表格的字段。

下面列出常用的类型名，在设计数据库时可以查表：

**常用的SQLAlchemy列类型：**

|  |  |  |
| --- | --- | --- |
| 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 | 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 | 日期 |
| Time | datetime.time | 时间 |
| DateTime | datetime.datetime | 日期和时间 |
| Interval | datetime.timedelta | 时间间隔 |
| Enum | str | 一组字符串 |
| PickleType | 任何 Python 对象 | 自动使用 Pickle 序列化 |
| LargeBinary | str | 二进制文件 |

db.Column 中其余的参数指定属性的配置选项。下面列出了一些可用选项。

**最常使用的SQLAlchemy列选项**

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

注意事项：

Flask-SQLAlchemy 要求每个模型都要定义主键，这一列经常命名为 id。

* 1. 使用Alembic进行数据库的迁移

在开发程序的过程中，往往会有修改数据库模型的需求，而且修改之后还需要更新数据库。而Flask-SQLAlchemy只有在数据库表不存在时才会根据模型进行创建。因此，更新表的唯一方式就是先删除旧表，然后重新进行create\_all()的操作，非常的麻烦，而且数据还会丢失。

SQLAlchemy 的主力开发人员编写了一个迁移框架，称为 Alembic，我们不会直接使用Alembic，而是使用Flask-Migrate插件，他要和Flask-Script一起使用。

* + 1. 安装插件

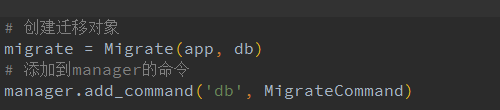
pip install Flask-Migrate

* + 1. 添加数据迁移的命令

在Flask-Migrate的参数中，提供了一个Migrate和MigrateCommand类。这个包的MigrateCommand中包含了所有和数据库相关的命令。

利用Migrate实例化迁移对象，把app和db进行初始化。

from flask\_migrate import Migrate,MigrateCommand



* + 1. 初始化迁移仓库

在维护数据库迁移之前，要使用 init 子命令创建迁移仓库：



这个命令会创建 migrations 文件夹，所有迁移脚本都存放其中。

* + 1. 创建迁移文件

将模型生成迁移文件，只要模型更改了，就需要执行一遍这个命令。migrate子命令用来自动创建迁移脚本：



在使用时，建议跟-m”xxx”参数将迁移记录描述清楚，方便以后版本回溯。

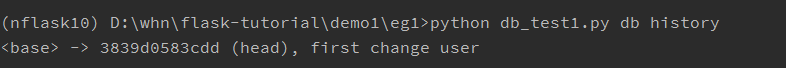
这里注意-m后面没有空格，同时必须是双引号。

* + 1. 更新迁移文件

将迁移文件真正的映射到数据库中。每次运行了`migrate`命令后，就记得要运行这个命令。



* + 1. 历史记录



1. 常用查询
   1. filter和filter\_by

“filter\_by()”方法，对查询结果过滤，参数必须是键值对”key=value”。

# 查询name='Tom'

users = User.query.filter\_by(name='Tom')

# 查询name='Tom' AND age=17

users = User.query.filter\_by(name='Jane', age=17)

效果相当于使用了WHERE子句，多个键值对用逗号分割。

query.filter\_by只有在你确切知道要查找的值时，才能工作。使用query.filter则可以避免这个问题，可以接收任何python的比较表达式。（如==, >, <, >=, <=）

“filter()”方法，对查询结果过滤

# 查询 age<20

users = User.query.filter(User.age<20)

# 查询name LIKE 'J%' AND age<20

users = User.query.filter(User.name.startswith('J'), User.age<20)

* 1. filter常用表达式

通用的查询语句：

equals:

query.filter(User.name == 'ed')

not equals:

query.filter(User.name != 'ed')

LIKE:

query.filter(User.name.like('%ed%'))

IN:

query.filter(User.name.in\_(['ed', 'wendy', 'jack']))

# 也适合查询对象:

query.filter(User.name.in\_( session.query(User.name).filter(User.name.like('%ed%'))))

NOT IN:

query.filter(~User.name.in\_(['ed', 'wendy', 'jack']))

IS NULL:

query.filter(User.name == None)

query.filter(User.name.is\_(None))

IS NOT NULL:

query.filter(User.name != None)

query.filter(User.name.isnot(None))

AND:

# 用 and\_()

from sqlalchemy import and\_

query.filter(and\_(User.name == 'ed', User.fullname == 'Ed Jones'))

# 或者以逗号分隔多个表达式

query.filter(User.name == 'ed', User.fullname == 'Ed Jones')

# 或者使用链式调用

query.filter(User.name == 'ed').filter(User.fullname == 'Ed Jones')

OR:

from sqlalchemy import or\_

query.filter(or\_(User.name == 'ed', User.name == 'wendy'))

MATCH:

query.filter(User.name.match('wendy'))

还有一些快捷方式：

contains ： 包含特定字符

startswith： 开头

endswith： 结尾

* 1. 数据筛选

“first()”方法，取返回列表中的第一个元素，当我们只查询一条记录时非常有用

user = User.query.filter\_by(name='Michael').first()

“order\_by()”方法，排序

from sqlalchemy import desc

# ORDER BY name

user = User.query.order\_by(User.name)

# ORDER BY age DESC, name

user = User.query.order\_by(desc(User.age), User.name)

“limit()”和”offset()”方法，分页

# LIMIT 10 OFFSET 10

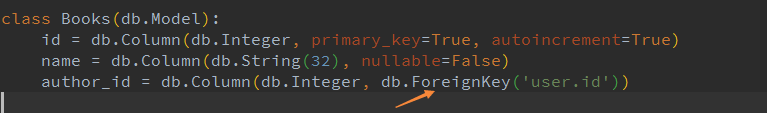
user = User.query.limit(10).offset(10)

等同于MySQL中的LIMIT和OFFSET，上例中我们从第11条记录开始取，并最多只取10条。

1. 表的关系
   1. 一对多

假设一本书只能有一个作者的话，那么书和作者之间的关系是什么那？

每本书属于一个作者，那么应该在书的表中，增加一个作者的ID列，而这个作者必须属于用户表里，这样书里添加一个字段，增加外键约束即可。



使用db.ForeignKey()增加外部的约束，里面跟表名.列名。

通过book怎么查询到作者信息那，这个属于Flask里比较特殊的，比Django要复杂。在一表中，增加一个隐含属性：



db.relationship() 的第一个参数表明这个关系的另一端是哪个模型。如果模型类尚未定义，可使用字符串形式指定。

db.relationship() 中的 backref 参数向Books模型中添加一个author属性，从而定义反向关系。这一属性可替代author\_id访问Books模型，此时获取的是模型对象，而不是外键的值。

lazy则是加载方式，默认是select，是在查询Books归属的User的时候就自动查询并加载books，也就是一次查询所有数据，这个例子中，设为dynamic是生成一个Query对象，但是还没有进行数据查询。

1. 课程总结
   1. 重点
      * 1. Flask-SQLAlchemy的接口
        2. 查询使用
        3. Flask-Migrate的使用
   2. 难点
      * 1. 一对多的外键使用
   3. 如何掌握？
      * 1. 使用shell来操作调试
        2. 可以先用单文件来测试
2. 课后练习
   * + 1. 定义作者和文章的数据表结构
       2. 添加2个作者，若干文章
3. 面试题
   * + 1. 谈下Flask的ORM和Django的ORM区别
4. 扩展知识或课外阅读推荐（可选）