Python数据库连接方式

python中的数据库连接有两种方式:

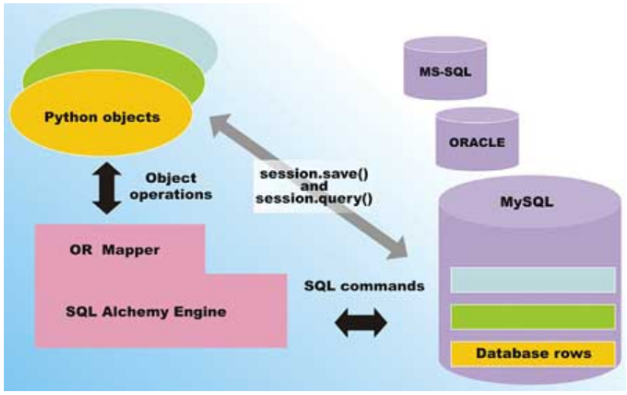
- `Python`标准数据库接口,使用`SQL`语句正常操作数据库,`e.g:`

  `MySQL`关系型数据库的`pymsql`模块

- 用`ORM`来进行数据库连接,`Python`中使用`sqlalchemy`, flask中典型的`flask_sqlalchemy`,已面向对象的方式进行数据库的连接与操作

- `ORM`是什么?有什么优势?

`ORM`，即`Object-Relational Mapping`（对象关系映射），它的作用是在关系型数据库和业务实体对象之间作一个映射，这样，我们在具体的操作业务对象的时候，就**不需要再去和复杂的`SQL`语句打交道**，只需**简单的操作对象的属性和方法。**



什么是Flask-SQLAlchemy?

Flask-SQLAlchemy 是一个 Flask 扩展,简化了在 Flask 程序中使用 SQLAlchemy 的操作。SQLAlchemy 是一个很强大的关系型数据库框架,支持多种数据库后台。 SQLAlchemy 提供了高层 ORM,也提供了使用数据库原生 SQL 的低层功能。

如何配置数据库?

安装第三方模块

```
pip install flask-sqlalchemy
```

数据库配置

官方配置信息网址

```
# app.py文件
from flask_sqlalchemy import SQLAlchemy
# 获取当前绝对路径
basedir = os.path.abspath(os.path.dirname(__file__))
app = Flask(__name__)
```

```
# SQLALCHEMY_DATABASE_URI: 用于连接数据的数据库。
app.config['SQLALCHEMY_DATABASE_URI'] =\
            'sqlite:///' + os.path.join(basedir, 'data.sqlite')

# sqlchemy将会追踪对象的修改并且发送信号
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

# 每次请求结束之后都会提交数据库的变动
app.config['SQLALCHEMY_COMMIT_ON_TEARDOWN'] = True

# 数据库与app项目关联，返回SQLAlchemy实例对象，后面可以使用对象调用数据
db = SQLAlchemy(app)
```

- 流行的数据库引擎采用的数据库 URL格式

数据库引擎	URL
MySQL	*mysql://username:password@hostname/database*
Postgres	*postgresql://username:password@hostname/database*
SQLite（Unix）	*sqlite:////absolute/path/to/database*
SQLite（Windows）	*sqlite:///c:/absolute/path/to/database*

- 连接`mysql`数据库报错解决

```
import pymysql
pymysql.install_as_MySQLdb()
```

定义模型

- 模型(Model)这个术语表示程序使用的持久化实体。
- 模型列类型
- 模型列属性

```
class Role(db.Model):
    # __tablename__类变量定义数据库中表的名称。不指定默认为模型名称。
    # Flask-SQLAlchemy需要给所有的模型定义主键列,通常命名为id。
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=True)

    # repr()显示一个可读字符串,用于调试和测试
    def __repr__(self):
        return '<Role %r>' % self.name

class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64), unique=True, index=True)

    def __repr__(self):
        return '<User %r>' % self.username
```

模型列类型

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

模型列属性

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

数据查询

- 查询过滤器

过滤器	说明
filter()	把过滤器添加到原查询，返回新查询
filter_by()	把等值过滤器添加到原查询，返回新查询
limit()	使用指定值限制原查询返回的结果数量，返回新查询
offset()	偏移原查询返回的结果，返回新查询
order_by()	排序返回结果，返回新查询
groupby()	原查询分组，返回新查询

- 查询执行函数

方法	说明
all()	以列表形式返回结果
first()	返回第一个结果，如果没有返回None
first_or_404()	返回第一个结果，如果没有抛出404异常
get()	返回主键对应记录，没有则返回None
get_or_404()	返回主键对应记录，如果没有抛出404异常
count()	返回查询结果数量
paginate()	返回paginate对象，此对象用于分页

分页实现

- 分页对象拥有的属性

属　性	说　明
items	当前页面中的记录
query	分页的源查询
page	当前页数
prev_num	上一页的页数
next_num	下一页的页数
has_next	如果有下一页，返回 True
has_prev	如果有上一页，返回 True
pages	查询得到的总页数
per_page	每页显示的记录数量
total	查询返回的记录总数

- 分页对象拥有的方法

方　法	说　明
iter_pages (left_edge=2, left_current=2, right_current=5, right_edge=2)	一个迭代器，返回一个在分页导航中显示的页数列表。这个列表的最左边显示 left_edge 页，当前页的左边显示 left_current 页，当前页的右边显示 right_current 页，最右边显示 right_edge 页。例如，在一个 100 页的列表中，当前页为第 50 页，使用默认配置，这个方法会返回以下页数：1、2、None、48、49、50、51、52、53、54、55、None、99、100。None 表示页数之间的间隔
prev()	上一页的分页对象
next()	下一页的分页对象

数据库关系

数据库实体间有三种对应关系：一对一，一对多，多对多。

一对一关系

一个学生对应一个学生档案材料，或者每个人都有唯一的身份证编号。

模型定义

```
# one_to_one.py文件

from app import db
import pymysql
pymysql.install_as_MySQLdb()


class People(db.Model):
    __tablename__ = 'peoples'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(20), nullable=False)
    """
    关系使用 relationship() 函数表示;
          'Card': People类对应的表与Card类对应的表关联;
          backref='people': 反向引用, 在Card类中添加新属性people;
          uselist=False: 默认为True, 如果为False代表一对一关系;
          lazy='select': 决定了 SQLAlchemy 什么时候从数据库中加载数据; 还可以设
定:'joined', 'subquery' , 'dynamic'
    可能报错: SAWarning: Multiple rows returned with uselist=False for lazily-
loaded
    """
    card = db.relationship('Card', backref='people', uselist=False)

    def __init__(self, name):
        self.name = name

    def __repr__(self):
        return '<People: %s>' % (self.name)


class Card(db.Model):
    __tablename__ = 'cards'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    cardID = db.Column(db.String(30), nullable=False)
    # 外键必须用类 sqlalchemy.schema.ForeignKey 来单独声明;
    people_id = db.Column(db.Integer, db.ForeignKey('peoples.id'))

    def __repr__(self):
        return "<Card: %s>" % (self.cardID)
```

基本操作

```
# one_to_one.py文件
```

```
if __name__ == '__main__':
    db.drop_all()
    db.create_all()

    # 增元素
    p1 = People(name="西部开源")
    p2 = People(name="粉条")

    db.session.add_all([p1, p2])
    db.session.commit()

    card1 = Card(cardID='001', people_id=p1.id)
    card2 = Card(cardID='002', people_id=p2.id)

    db.session.add_all([card1, card2])
    db.session.commit()

    # 查找
    card_people1 = Card.query.filter_by(cardID='001').first().people
    people_card1 = People.query.filter_by(id=1).first().card
    print(card_people1)
    print(people_card1)

    # 删除
    print("删除前: ", Card.query.all(), People.query.all())
    card = Card.query.filter_by(cardID='001').first()
    db.session.delete(card)
    db.session.commit()
    print("删除后: ", Card.query.all(), People.query.all())
```

一对多关系

一个学生只属于一个班，但是一个班级有多名学生。

- 设计数据库表：只需在 学生表 中多添加一个班级号的ID；
- 通过添加主外键约束，避免删除数据时造成数据混乱！

模型定义

```
# one_to_many.py文件

from app import db
import  pymysql
pymysql.install_as_MySQLdb()
```

```python
# 一对多： 外键写在多的一端; Grade:Stdudents== 1:n
class Student(db.Model):
    __tablename__ = 'students'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(20), nullable=False)
    # 外键
    grade_id = db.Column(db.Integer, db.ForeignKey('grades.id'))

    def __repr__(self):
        return "<Student: %s %s %s>" % (self.id, self.name, self.grade_id)

class Grade(db.Model):
    __tablename__ = 'grades'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(20), nullable=False)
    # 反向引用, 让Student类中添加属性grade;
    students = db.relationship('Student', backref='grade')

    def __repr__(self):
        return "<Grade: %s %s %s>" % (self.id, self.name, self.students)
```

基本操作

```python
if __name__ == '__main__':
    db.drop_all()
    db.create_all()

    # 增元素
    s1 = Student(name="西部开源")
    s2 = Student(name="粉条")

    db.session.add_all([s1, s2])
    db.session.commit()

    grade1 =Grade(name='Python开发')
    grade1.students.append(s1)
    grade1.students.append(s2)

    db.session.add_all([grade1])
    db.session.commit()
    print("添加成功: ", Student.query.all(), Grade.query.all())

    # 查找
    student_grade1 = Student.query.filter_by(name='粉条').first().grade
    grade_students = Grade.query.filter_by(name="Python开发").first().students
    print(student_grade1)
    print(grade_students)

    # 删除
    print("删除前: ", Student.query.all(), Grade.query.all())
    grade1 = Grade.query.filter_by(name='Python开发').first()
```

```
    db.session.delete(grade1)
    db.session.commit()
    print("删除后: ", Student.query.all(), Grade.query.all())
```

多对多关系

一个学生可以选择多门课，一门课也有多名学生。

- 对于多对多表，通过关系表就建立起了两张表的联系！多对多表时建立主外键后，要先删除约束表内容再删除主表内容。

模型定义

```
# many_to_many.py文件
from app import db
import  pymysql
pymysql.install_as_MySQLdb()

# 第三方表
tags=db.Table('tags',

 db.Column('student_id',db.Integer,db.ForeignKey('math_students.id')),
              db.Column('course_id',db.Integer,db.ForeignKey('courses.id')))

class MathStudent(db.Model):
    __tablename__ = 'math_students'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(20), nullable=False)
    courses = db.relationship('Course', secondary=tags)

    def __repr__(self):
        return "<MathStudent: %s >" % (self.name)

class Course(db.Model):
```

```
    __tablename__ = 'courses'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(20), nullable=False)
    students = db.relationship('MathStudent', secondary=tags)
    def __repr__(self):
        return "<Course: %s >" % (self.name)
```

基本操作

```
# many_to_many.py文件
if __name__ == '__main__':
    db.drop_all()
    db.create_all()

    # 创建
    s1 = MathStudent(name="拉格朗日")
    s2 = MathStudent(name="麦克劳林")
    s3 = MathStudent(name="罗尔")

    course1 = Course(name="高等数学")
    course2 = Course(name="线性代数")

    s1.courses.append(course1)
    s1.courses.append(course2)
    s2.courses.append(course1)
    s3.courses.append(course2)

    db.session.add_all([s1, s2, s3, course1, course2])
    db.session.commit()


    # 删除
    courseObj = Course.query.filter_by(name="线性代数").first()
    studentObj = MathStudent.query.filter_by(name="罗尔").first()
    print("删除前选择线性代数的学生: ", courseObj.students)
    courseObj.students.remove(studentObj)
    print("删除后择线性代数的学生: ", courseObj.students)

    # 其他操作相同
```

实战案例