数据模型

一 模型关系

- 一对多(使用最多)
  - 一：学生(Student)
    - 添加反向引用
  - 多：文章(Article)
    - 添加外键关联
- 一对一
  - 一：学生(Student)
    - 添加反向引用(在一对多的基础上多添加属性:`uselist=False`)
  - 一：详情(Profile)
    - 添加外键关联
- 多对多
  - 多：学生(Student)
    - 需要添加反向引用
    - 需要使用`secondary`指定中间关联表
    - 设置反向查询数据的加载时机，需要使用:`db.backref`
  - 多：课程(Course)
  - 中间关联表：此表不需要用户维护
    - 表名
    - 关联外键

http://www.pythondoc.com/flask-sqlalchemy/models.html

```
# 学生模型
class Student(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(20), unique=True)
    ''' 添加反向引用（无需迁移）
    参数1：关联的模型名
    backref：反向引用的字段名
```

```
    lazy：关联数据加载时机
        'select'/True：首次使用时自动查询，是默认选项
        'joined'/False：关联查询时使用
        'subquery'：子查询时使用
        'dynamic'：不加载数据，提供了关联数据的查询(不能用在一的一侧)
    '''
    articles = db.relationship('Article', backref='stu',
lazy='dynamic')
    # 添加一对一的反向引用，需要设置：uselist=False
    profile = db.relationship('Profile', backref='stu',
uselist=False)
    # 添加多对多的反向引用，需要使用secondary指定中间关联表
    courses = db.relationship('Course',
secondary='xuankebiao',backref=db.backref('students',
lazy='dynamic'), lazy='dynamic')

# 课程模型
class Course(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(20), unique=True)


# 学生选课表(中间关联模型)
sc = db.Table('sc',
    db.Column('student_id', db.Integer, db.ForeignKey('student.id')),
    db.Column('course_id', db.Integer, db.ForeignKey('course.id'))
)
# 详情模型
class Profile(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    realname = db.Column(db.String(20), unique=True)
    # 添加关联外键
    sid = db.Column(db.Integer, db.ForeignKey('student.id'))


# 文章模型
class Article(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(32), unique=True)
    content = db.Column(db.Text)
    # 添加外键关联，需要指定关联的'表名.字段'
    sid = db.Column(db.Integer, db.ForeignKey('student.id'))
```

二 模型总结

- 关联查询

```
data = db.session.query(xuankebiao).join(Student).all()
```

- 组合查询

```
data1 = Student.query.filter(Student.id == 1)
data2 = Student.query.filter(Student.id == 2)
data = data1.union(data2).all()
```

- 原生sql查询

```
db.session.execute("select * from student").fetchall()
db.session.execute('insert into users(name)
values(:value)',params={"value":'wupeiqi'})
 db.session.commit()
```

三 数据缓存

- 说明：

  因为数据库的速度是一个web应用性能的瓶颈，因此，为了提高访问效率，尽可能的减少数据库的操作。可以将经常访问的数据缓存起来，再次使用时直接从缓存中获取，而不是每次都操作数据库。

- flask-cache：专门负责数据缓存的扩展。
- 安装：`pip install flask-cache`
- 使用：

```
from flask_cache import Cache

# 配置
# 缓存类型
app.config['CACHE_TYPE'] = 'redis'
# redis主机
app.config['CACHE_REDIS_HOST'] = '127.0.0.1'
# redis端口
```

```
app.config['CACHE_REDIS_PORT'] = 6379
# redis数据库
app.config['CACHE_REDIS_DB'] = 1

# 创建对象
cache = Cache(app, with_jinja2_ext=False)
```

- 缓存视图函数

```
# timeout：有效期，默认为300s
# key_prefix：键前缀
@cache.cached(timeout=100, key_prefix='index')
def index():
    print('查询数据库')
    return '数据缓存'
```

- 清除缓存

```
@app.route('/delete/')
def delete():
    # 指定删除
    # cache.delete('index')
    # 清空全部
    cache.clear()
    return '缓存已删除'
```

- 缓存普通函数

```
# 缓存普通函数时最好指定key_prefix参数
# 因为不指定时，缓存的键前缀默认是调用的视图函数所在路由
@cache.cached(timeout=10, key_prefix='aaa')
def aaa():
    print('查询数据库')
    return 'hello world'

# 缓存普通函数
@app.route('/common/')
def common():
    return aaa()
```

- 自定义缓存

```python
@app.route('/test/')
def test():
    # 先从缓存中获取数据
    data = cache.get('test_data')
    if data:
        # 有缓存，直接返回
        return data
    # 没有缓存
    print('读取数据库')
    data = '123456'
    # 将数据缓存起来
    cache.set('test_data', data, timeout=20)
    return data
```