数据连接

- 讲师：潘sir
- python连接mysql数据库的各个方式总结

MySQLdb

即：MySQL-python，只支持python2，弃用

- 衍生库：mysqlclient；安装方法：`pip install mysqlclient`

```
import mysqlclient

db = mysqlclient.connect(
     host="localhost",    # 主机名
     user="dcs",          # 用户名
     passwd="123",  # 密码
     db="db")         # 数据库名称

# 查询前，必须先获取游标
cur = db.cursor()

# 执行的都是原生SQL语句
cur.execute("SELECT * FROM YOUR_TABLE_NAME")

for row in cur.fetchall():
    print(row[0])

db.close()
```

PyMySQL

纯python驱动，速度相对慢，还兼容MySQLdb。安装方法：`pip install PyMySQL`

核心参数

```
import pymysql
mysql_config = {
    'host': '10.211.55.5',
    'port': 3306,
    'user': 'root',
    'password': '123456',
    'db': 'flaskblog',
    'charset': 'utf8mb4',
```

```
}
# 获取数据库连接对象
db = pymysql.connect(**mysql_config)
sql = '''select * from sms_sign limit 5'''
# 获取游标
cursor = db.cursor()
# 执行sql
cursor.execute(sql)

# 1. 获取单条数据
a = cursor.fetchone()
# 2. 获取前N条数据
b = cursor.fetchmany(3)
# 3. 获取所有数据
c = cursor.fetchall()

# 关闭连接
cursor = db.close()
print(a)
# print(b)
# print(c)
```

```
(26, '签名', '深圳', 'tu1', datetime.datetime(2019, 6, 30, 23, 8, 55), 4,
'reviewing', datetime.datetime(2019, 6, 30, 8, 47, 7), '', 1)
```

```
# 案例
import pymysql
# 不带编码可能查询数据丢失
# conn = pymysql.connect(host='10.211.55.5', user='root', passwd="123456",
db='flaskblog')
conn = pymysql.connect(host='10.211.55.5', user='root', passwd="123456",
db='flaskblog', charset='utf8mb4')
cur = conn.cursor()
cur.execute("select * from sms_sign limit 5")
for r in cur:
    print(r)
cur.close()
conn.close()
```

```
(26, '签名', '深圳', 'tu1', datetime.datetime(2019, 6, 30, 23, 8, 55), 4,
'reviewing', datetime.datetime(2019, 6, 30, 8, 47, 7), '', 1)
(27, '签名', '深圳', 'tu1', datetime.datetime(2019, 6, 30, 23, 5, 13), 4,
'reviewing', datetime.datetime(2019, 6, 30, 8, 59, 28), None, 1)
(28, '推送一切', '北京', 'tu1', datetime.datetime(2019, 7, 3, 13, 18, 40), 4,
'passed', datetime.datetime(2019, 6, 30, 9, 8, 4), '', 0)
(29, '推送一切', '北京', 'tu1', datetime.datetime(2019, 7, 12, 15, 3, 17), 4,
'passed', datetime.datetime(2019, 6, 30, 13, 57, 24), '', 0)
(30, '签名', '深圳', 'tu1', datetime.datetime(2019, 7, 3, 5, 17, 44), 4,
'reviewing', datetime.datetime(2019, 7, 3, 5, 17, 44), None, 1)
```

Records

K神

Kenneth Reitz，自学python，22岁开发了requests库，requests库的下载量超过3亿次，github上python排名世界第二，作品：

1. requests：神作，爬虫/接口自动化的根本
2. requests-html：爬虫框架，完全支持js，css，xpath，伪装浏览器，自动翻页等
3. pipenv：轻量级虚拟环境管理工具，比较推荐
    - 补充说明：虚拟环境工具有很多，比如：virtualenv，aconda，venv，pyenv，pycharm，其中aconda比较笨重，占空间1.5G，一键安装完90%一般人这一辈子会用到的Python套件，除非专门做数据挖掘/分析，一般不推荐aconda
4. records：
    - 支持多种数据库
    - 用法更简单，不用游标
    - 支持数据库事物
    - 轻松导出为 json, yaml, xls, xlsx, pandas, html 等多种数据格式

简单查询

```python
import records

# 获取数据库，数据库类型 + db-api + 账号:密码 + 地址 + 库名 + 编码
# mysql+pymysql://root:123456@10.211.55.5:3306/flaskblog?charset=utf8
db = records.Database('mysql+pymysql://root:123456@10.211.55.5:3306/flaskblog?
charset=utf8')
# 查询
rows = db.query('select * from sms_sign limit 5')
print(rows)
print(list(rows))
print(list(map(dict, list(rows))))
# for i in rows:
#     print(dict(i))
```

创建表格

```
import records

db = records.Database('mysql+pymysql://root:123456@10.211.55.5:3306/flaskblog?charset=utf8')
# 直接传入sql
sql_create_table = """CREATE TABLE IF NOT EXISTS dcs_user (name varchar(20),age int) DEFAULT CHARSET=utf8;"""
db.query(sql_create_table)
```

```
<RecordCollection size=0 pending=True>
```

插入单条数据

两种方法：

1. 通过占位符构造sql字符串
2. records提供了特殊语法

```
import records

db = records.Database('mysql+pymysql://root:123456@10.211.55.5:3306/flaskblog?charset=utf8')

# 1. 通过占位符去实现：insert into dcs_user(name, age) values ("dcs", 18);
# insert_sql = '''INSERT INTO dcs_user(name,age) values ("%s", %s);''' % ('houhc', 18)
# db.query(insert_sql)

# 2. 通过records自带语法实现
user = {"name": "xiep", "age": 19}
db.query('INSERT INTO dcs_user(name,age) values (:name, :age)', **user)
```

```
<RecordCollection size=0 pending=True>
```

- 练习：连接数据库，创建一个表格，然后自己插入一条数据

插入多条数据

```
import records

db = records.Database('mysql+pymysql://root:123456@10.211.55.5:3306/flaskblog?
charset=utf8')

users = [
    {"name":"dcs1", "age": 13},
    {"name":"dcs2", "age": 15},
    {"name":"dcs3", "age": 16}
]
# 一次性插入多条数据，接收一个列表，每个元素为一个字典
a = db.bulk_query('INSERT INTO dcs_user(name,age) values (:name, :age)',
users)
print(a, type(a))
```

```
None <class 'NoneType'>
```

- 练习：以自己的姓名创建一个表格，并插入多条数据，然后在数据库检查真实结果

数据查询

```
import records

db = records.Database('mysql+pymysql://root:123456@10.211.55.5:3306/flaskblog?
charset=utf8')

rows = db.query('SELECT * FROM dcs_user;')
# # 得到所有数据，注意是records对象
# print(rows.all())
# # 返回列表，字典形式展示
# print(rows.all(as_dict=True))
# # 获取第一个，是records对象
# print(rows.first())
# # 以字典形式获取第一个
# print(rows.first(as_dict=True))
# # 排序字典
# print(rows.first(as_ordereddict=True))
# # 查询唯一的一个，必须唯一一个记录，才能执行通过
print(rows.one())
```

```
<Record {"name": "dcs", "age": 18}>
```

导出json数据

超级强大，省略很多数据处理的过程，records支持将数据以json格式导出。还包括：yaml, xls, csv, html

```
import records

db = records.Database('mysql+pymysql://root:123456@10.211.55.5:3306/flaskblog?
charset=utf8')
rows = db.query('SELECT * FROM dcs_user;')
# 转成json格式
json_rows = rows.export('json')
# 转成yaml格式
yaml_rows = rows.export('yaml')
# 转换成html
html_rows = rows.export('html')
# 转换成html
xls_rows = rows.export('xls')
# 转换成html
csv_rows = rows.export('csv')
# 转成xml格式
print(csv_rows)
```

```
name,age
dcs,18
```

直接导出到excel文件

直接将导出内容，写入到excel文件即可

```
import records

db = records.Database('mysql+pymysql://root:123456@10.211.55.5:3306/flaskblog?
charset=utf8')
rows = db.query('SELECT * FROM sms_sign where sign_user_id = 25;')
with open('users.xlsx', 'wb') as f:
    f.write(rows.export('xlsx'))
print("ok")
```

数据库事务性

- 原子性：一个事务包含多个操作，**这些操作要么全部执行，要么全都不执行**。支持回滚操作，在某个操作失败后，回滚到事务执行之前的状态
- 一致性：一致性是指事务使得系统从一个一致的状态转换到另一个一致状态
- 隔离性：并发事务之间互相影响的程度，比如一个事务会不会读取到另一个未提交的事务修改的数据
- 持久性：事务提交后，对系统的影响是永久的
- 场景案例：A给B转账，1. 从A读取余额；2. A扣除300；3. B读取余额；4. B增加300。必须保证这4个步骤要么同时成功，要么同时失败。否则账面将有出入

```
import records

db = records.Database('mysql+pymysql://root:123456@10.211.55.5:3306/flaskblog?charset=utf8')

# records天然支持数据库事物
# 通过transaction获取一个事物对象
with db.transaction() as tx:
    user = {"name": "captain", "age": 18}
    tx.query('INSERT INTO dcs_user(name,age) values (:name, :age)', **user)
    # 下面是错误的 sql 语句，有错误，则上面的 sql 语句不会成功执行。
    print("hello")
    tx.query('what?')
    print("ok")
```

```
hello
```

优点总结：

1. 支持多种类型数据库连接
2. 经过二次封装，用法及其简单
3. 支持数据库事物操作，不用自己实现事务一致性
4. 支持多种格式数据导出，不用自己进行数据处理

练习

1. 封装一个方法：`def select_sql(self, table, *fields, **kwargs):` 接收参数：表名，查询字段，查询条件(key=value)，返回结果为列表

```
    def select_sql(self, sql):
        try:
            return list(map(dict, list(self.db.query(sql))))
        except Exception as e:
            logging.error(e)

    def tsms_select(self, table, *fields, **kwargs):
        """select *fields from table where **kwargs"""
        if kwargs:
            options = 'where '
        else:
            options = ''
        for k, v in kwargs.items():
            if isinstance(v, str):
                v = '\"' + v + '\"'
            options += k + "=" + str(v) + " and "
        if kwargs:
            options = options[:-4]
```

```
        query_fields = ','.join(fields)
        sql = '''select {0} from {1} {2};'''.format(query_fields, table,
options)
        logging.info("[now execute sql is]: {}".format(sql))
        return self.select_sql(sql)
```

```
from tsms.tsms_db import TsmsDB
db = TsmsDB()
a = db.tsms_sql("sms_sign", "sign_id,signature", "source", "audit_status",
sign_id=1289)
print(a)
```

2. 完成一个用例，申请一个签名，校验数据库数据是否落地。包含字段：签名id，签名内容，来源，资质证明，审核状态

```
from tsms.tsms_web import TsmsWeb
from tsms.tsms_base import Tsmstest
from tsms.tsms_db import TsmsDB
import unittest
from time import sleep
from tenacity import retry, stop_after_attempt, wait_exponential
import logging
logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')


class TestWeb(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.ts = Tsmstest()
        cls.db = TsmsDB()

    @classmethod
    def tearDownClass(cls):
        pass

    def test_add_sign(self):
        # 调接口
        phone = self.ts.gen_phones(1)
        data = {"sign_id": 424, "temp_id": 180, "mobiles": phone}
        self.ts.req_post('message', data)
        assert self.ts.status_code == 200
        assert isinstance(self.ts.json["uuid"], str)
        # 查询数据库
        sleep(2)
        result = self.db.tsms_select("sms_send", "content,status,consume",
uuid=self.ts.json["uuid"])
        logging.info("[数据库查询结果]: {}".format(result))
```

```
        assert result[0]["status"] == "success"
        assert result[0]["consume"] == 1


if __name__ == '__main__':
    # 执行本suite
    unittest.main(argv=['ignored', '-v'], exit=False)
```

- 扩展部分：

    1. 引入失败重试
    2. 对发送内容进行校验。思路：通过接口去获取模版内容，签名内容，然后拼接出预期结果

```
from tsms.tsms_db import TsmsDB
from tsms.tsms_base import Tsmstest
import unittest
from time import sleep
from tenacity import retry, stop_after_attempt, wait_exponential
import logging
logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')


class TestWeb(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.ts = Tsmstest()
        cls.db = TsmsDB()
        cls.sign_table = "sms_sign"
        cls.temp_table = "sms_template"

    @classmethod
    def tearDownClass(cls):
        pass

    def get_sign_name(self, sign_id):
        """从数据库查询签名内容"""
        res = self.db.tsms_select(self.sign_table, "signature",
sign_id=sign_id)
        logging.info("[查询的签名内容是]: {}".format(res))
        sign_name = res[0]["signature"]
        return sign_name

    def get_temp_name(self, temp_id):
        res = self.db.tsms_select(self.temp_table, "template",
temp_id=temp_id)
        logging.info("[查询的签名内容是]: {}".format(res))
        sign_name = res[0]["template"]
        return sign_name
```

```
    def get_send_content(self, sign_id, temp_id):
        sign = self.get_sign_name(sign_id)
        temp = self.get_temp_name(temp_id)
        send = "【%s】%s" % (sign, temp)
        logging.info("[即将推送的内容是]: {}".format(send))
        return send

    @retry(stop=stop_after_attempt(5), wait=wait_exponential(1, max=10))
    def check_send_ok(self, sign_id, temp_id):
        """校验数据库结果"""
        result = self.db.tsms_select("sms_send", "content,status,consume",
uuid=self.ts.json["uuid"])
        logging.info("[数据库查询结果]: {}".format(result))
        assert result[0]["status"] == "success"
        assert result[0]["consume"] == 1
        assert result[0]["content"] == self.get_send_content(sign_id, temp_id)

    def test_add_sign(self):
        # 构造数据
        sign_id = 424
        temp_id = 180
        phone = self.ts.gen_phones(1)
        data = {"sign_id": sign_id, "temp_id": temp_id, "mobiles": phone}
        # 调接口发送
        self.ts.req_post('message', data)
        assert self.ts.status_code == 200
        assert isinstance(self.ts.json["uuid"], str)
        # 校验数据库
        self.check_send_ok(sign_id, temp_id)

if __name__ == '__main__':
    # 执行本suite
    unittest.main(argv=['ignored', '-v'], exit=False)
```

3. 完成一个方法，该方法可以修改指定表的指定字段的值
    - 要求：
        1. 只允许修改 audit_status 字段
        2. 参考定义：`def tsms_update(self, table, field, value, **kwargs):`
    - 思路：先写一条sql，然后从调用者的角度去思考怎么剥离变量

```
# update sms_sign audit_status="reviewing" where sign_id=1289;

    def tsms_update(self, table, field, value, **kwargs):
        """更新指定的字段"""
        if kwargs:
            options = 'where '
            for k, v in kwargs.items():
```

```
                if isinstance(v, str):
                    v = '\"' + v + '\"'
                options += k + "=" + str(v) + " and "
            options = options[:-4]
        else:
            return
        sql = '''update {0} set {1}="{2}" {3};'''.format(table, field, value,
options)
        logging.info("[now execute sql is]: {}".format(sql))
        try:
            self.db.query(sql)
        except:
            logging.ERROR("[数据库更新失败 sql 是]: {}".format(sql))
```

课后扩展

自己编写方法：

1. 根据指定的条件删除指定的数据
2. 编写一个方法，可以插入多条数据，需要考虑的点：
    1. 插入的数据必须是动态的
    2. 插入方式选择批量插入(一次100条) + for循环
    3. 数据构造的方法，不仅是接口测试的必备技能，性能测试更常用到

逻辑删除/业务删除

- 逻辑删除：真正的删除数据库记录
- 业务删除：记录保留，但是通过特定的字段标识是否展示该记录，如：is_delete

```
    def tsms_delete(self, table, **kwargs):
        """DELETE FROM `dcs_user` WHERE xxx=dcs"""
        if kwargs:
            options = 'where '
        else:
            options = ''
        for k, v in kwargs.items():
            if isinstance(v, str):
                v = '\"' + v + '\"'
            options += k + "=" + str(v) + " and "
        if kwargs:
            options = options[:-4]
        sql = '''delete from {0} {1};'''.format(table, options)
        logging.info("[now execute sql is]: {}".format(sql))
        return self.db.query(sql)

    def tsms_record_del(self, table, **kwargs):
        # tsms_update(self, table, field, value, **kwargs):
        self.tsms_update(table, "is_delete", 1, **kwargs)
```

结合业务编写用例

- 改造用例：选一条之前通过前端断言的用例，改造成由db结果进行断言
- 改造用例：发送一条消息，但是从mysql中查询数据后，进行校验；注意：需要进行db查询错误重试

```
from tsms.tsms_db import TsmsDB
from tsms.tsms_base import Tsmstest
import unittest
from time import sleep
from tenacity import retry, stop_after_attempt, wait_exponential
import logging
logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')


class TestSend(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.db = TsmsDB()
        cls.ts = Tsmstest()

    @classmethod
    def tearDownClass(cls):
        pass

    @retry(stop=stop_after_attempt(5), wait=wait_exponential(multiplier=1,
max=10))
    def check_db(self, phone):
        real_res = self.db.tsms_select("sms_send", "mobile,status,consume",
uuid=self.ts.json["uuid"])[0]
        assert real_res["mobile"] == phone
        assert real_res["status"] == "success"
        assert real_res["consume"] == 1

    def test_send_one(self):
        # 调接口
        phone = self.ts.gen_phones(1)
        data = {"sign_id": 424, "temp_id": 180, "mobiles": phone}
        self.ts.req_post('message', data)
        assert self.ts.status_code == 200
        assert isinstance(self.ts.json["uuid"], str)
        # 查数据库
        self.check_db(phone[0])

if __name__ == '__main__':
    # 执行本suite
    unittest.main(argv=['ignored', '-v'], exit=False)
```

SQLALchemy

支持原生sql，又支持ORM。安装：`pip install SQLAlchemy`

ORM

Object-Relational Mapping：把关系型数据库的表结构映射到对象上，通过操作对象，来操作数据库

常见ORM库

- SQLObject：ActiveRecord 模式，库比较简单
- Storm：轻量的API
- Django-ORM：学习成本低，但紧密和Django集成，不好处理复杂的查询，在Django环境外不能使用
- peewee：Django式的API
- SQLALchemy：企业级 API，使得代码有健壮性和适应性，灵活的设计，使得能轻松写复杂查询；但属于重量级 API，导致长学习曲线

```
from sqlalchemy import Column, String, create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base
# 创建对象到基类
Base = declarative_base()
# 继承基类
class Students(Base):
    __tablename__ = 'students'
    id = Column(String(20), primary_key=True)
    name = Column(String(20))

# 初始化数据库连接：'数据库类型+数据库驱动名称://用户名:口令@机器地址:端口号/数据库名'
engine =
create_engine('mysql+pymysql://root:123456@10.211.55.5:3306/flaskblog?
charset=utf8')
# 创建DBSession类型
DBSession = sessionmaker(bind=engine)

# 创建session对象
session = DBSession()
new_student = Students(id=1, name='dcs')
session.add(new_student)
session.commit()
session.close()
```

总结：

1. 相比其他的ORM， SQLAlchemy更专注工作单元开发
2. DB Session 比较难理解和使用，但是这复杂性可以有效的减少bug
3. 每个DB session 都限定了一个数据库连接，数据库交互代码很容易调试