**7.2业务逻辑层·事务处理**

当我们需要对多张表进行增、删、改操作时，就需要用到事务处理，这里贴一下百度百科对事务处理的定义：

事务处理（TRANSACTION）是由一个或多个SQL语句序列结合在一起所形成的一个逻辑处理单元。事务处理中的每个语句都是完成整个任务的一部分工作，所有的语句组织在一起能够完成某一特定的任务。DBMS在对事务处理中的语句进行处理时，是按照下面的约定来进行的，这就是“事务处理中的所有语句被作为一个原子工作单位，所有的语句既可成功地被执行，也可以没有任何一个语句被执行”。DBMS负责完成这种约定，即使在事务处理中应用程序异常退出，或者是硬件出现故障等各种意外情况下，也是如此。在任何意外情况下，DBMS都负责确保在系统恢复正常后，数据库内容决不会出现“部分事务处理中的语句被执行完”的情况。

# 举个例子

## 银行转账业务

银行给用户A转账给用户B时，有两个操作，从A里面减100，然后给B加100。这两个操作必须放在一个事务里面才行，否是就会出现钱扣了，对方又没到账的情况。通过session.begin 方法可以获取事务对象，执行完sql后，要把事务commit提交，如果出现异常，则把事务rollback。

## 批量插入

共10记录，一次插入一条，要求要么全部成功，要么全部失败。

# 修改services/user\_service.py

这里先演示批量插入用户，新增批量save\_batch与save\_batch\_no\_trans方法

from models.user import User  
from services import BaseService  
  
  
class UserService(BaseService):  
 """  
 用户模块业务处理类  
 """  
  
 def login(self, user\_name, password):  
 """  
 用户密码登录  
 :param user\_name:  
 :param password:  
 :return:  
 """  
 u = self.db.session.query(User).filter(User.user\_name == user\_name).first()  
 if u is None:  
 raise Exception("用户名或密码错误")  
 if u.password != password:  
 raise Exception("用户名或密码错误")  
 res = u.to\_dict(camel=True)  
 del res['password']  
 del res['createTime']  
 del res['updateTime']  
 del res['isDeleted']  
 return res  
  
 def save\_batch(self, form\_list):  
 """  
 批量插入用户-开启事务  
 :param form\_list:  
 :return:  
 """  
 # 使用with的话，会自动执行session.commit()，如异常自动执行session.rollback()  
 with self.db.session.begin(): # 开启事物  
 for form in form\_list:  
 model = User(\*\*form)  
 self.db.session.add(model)  
 # flush会将session中的数据刷到数据库中，使数据库主键自增；但不会写到磁盘里  
 self.db.session.flush()  
  
 def save\_batch\_no\_trans(self, form\_list):  
 """  
 批量插入用户-未开启事务  
 :param form\_list:  
 :return:  
 """  
 for form in form\_list:  
 model = User(\*\*form)  
 self.db.session.add(model)  
 self.db.session.commit()

# 修改controllers/user\_controllers.py

新增/user/saveBatch与/user/saveBatchNoTrans接口

from flask import Blueprint, request  
  
from controllers import R  
from models.user import User  
from services.user\_service import UserService  
from validators import BasePageForm  
from validators.id\_validator import IdForm, IdsForm  
from validators.user\_validator import UserForm, LoginForm  
  
# 声明一个蓝图  
user = Blueprint('user', \_\_name\_\_, url\_prefix="/user")  
  
# 声明一个用户业务服务  
user\_service = UserService(model=User)  
  
  
@user.route("/get", methods=['POST'])  
def user\_get():  
 """  
 通过id获取用户信息  
 :return:  
 """  
 form = IdForm()  
 form.validate\_for\_api()  
 # 可通过form.data获取所有提交参数  
 # 或者直接拿id值 id=form.id.data  
 # u = User.query.filter\_by(id=form.id.data).first()  
 # 通过主键查询  
 u = user\_service.get(form)  
 if u is not None:  
 return R.data(u.to\_dict(camel=True))  
 else:  
 return R.fail("该记录不存在")  
  
  
@user.route("/list", methods=['POST'])  
def user\_list():  
 """  
 分页查询用户列表  
 :return:  
 """  
 form = BasePageForm()  
 form.validate\_for\_api()  
 return R.data(user\_service.list(form))  
  
  
@user.route("/save", methods=['POST'])  
def user\_save():  
 """  
 添加用户  
 :return:  
 """  
 form = UserForm()  
 form.validate\_for\_api()  
 # 可通过form.data获取所有提交参数  
 # print(form.data)  
 user\_service.save(form)  
 return R.success("添加用户成功")  
  
  
@user.route("/update", methods=['POST'])  
def user\_update():  
 """  
 修改用户  
 :return:  
 """  
 form = UserForm()  
 form.validate\_for\_api()  
 # 可通过form.data获取所有提交参数  
 # print(form.data)  
 user\_service.update(form)  
 return R.success("修改用户成功")  
  
  
@user.route("/delete", methods=['POST'])  
def user\_delete():  
 """  
 删除用户  
 :return:  
 """  
 form = IdsForm()  
 form.validate\_for\_api()  
 # 可通过form.data获取所有提交参数  
 # print(form.data)  
 user\_service.delete(form)  
 return R.success("删除用户成功")  
  
  
@user.route("/login", methods=['POST'])  
def user\_login():  
 """  
 登录  
 :return:  
 """  
 form = LoginForm()  
 form.validate\_for\_api()  
 # 可通过form.data获取所有提交参数  
 # print(form.data)  
 res = user\_service.login(form.userName.data, form.password.data)  
 return R.data(res)  
  
  
@user.route("/saveBatch", methods=['POST'])  
def user\_save\_batch():  
 """  
 批量插入用户-开启事务  
 :return:  
 """  
 # 复杂的表单校验-wtforms支持不是很好，这里先不校验  
 user\_service.save\_batch(request.get\_json())  
 return R.success("添加成功")  
  
  
@user.route("/saveBatchNoTrans", methods=['POST'])  
def user\_save\_batch\_no\_trans():  
 """  
 批量插入用户-未开启事务  
 :return:  
 """  
 # 复杂的表单校验-wtforms支持不是很好，这里先不校验  
 user\_service.save\_batch\_no\_trans(request.get\_json())  
 return R.success("添加成功")

# 运行Flask服务

flask run

# 测试前重新建表

drop table if exists t\_user;  
CREATE TABLE `t\_user` (  
 `id` bigint(20) NOT NULL AUTO\_INCREMENT COMMENT '主键',  
 `user\_name` varchar(32) NOT NULL COMMENT '用户名',  
 `real\_name` varchar(32) NOT NULL COMMENT '姓名',  
 `password` varchar(64) NOT NULL COMMENT '密码',  
 `create\_time` datetime DEFAULT NULL COMMENT '创建时间',  
 `update\_time` datetime DEFAULT NULL COMMENT '更新时间',  
 `is\_deleted` tinyint(1) DEFAULT NULL COMMENT '逻辑删除:0=未删除,1=删除',  
 PRIMARY KEY (`id`),  
 UNIQUE KEY `user\_name` (`user\_name`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='用户';

# 使用Postman接口测试工具访问

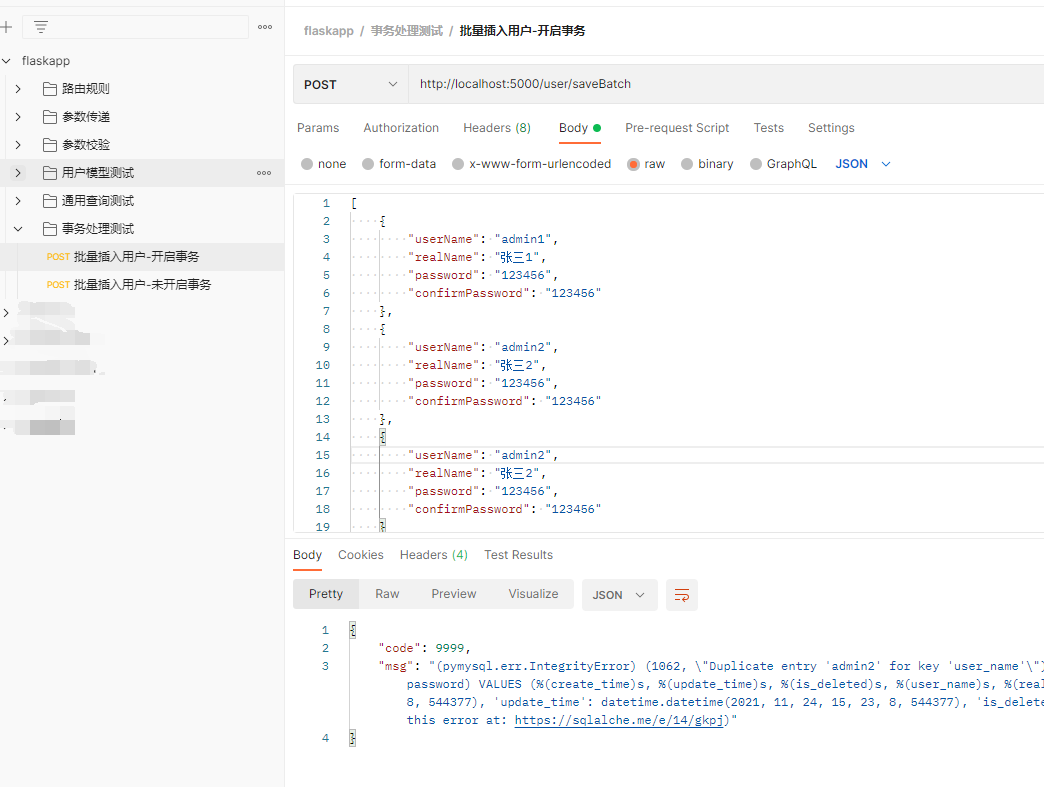
## 访问/user/saveBatch

因为user\_name设置为唯一，所以可以人为造一些用户名重复的数据

[  
 {  
 "userName": "admin1",  
 "realName": "张三1",  
 "password": "123456",  
 "confirmPassword": "123456"  
 },  
 {  
 "userName": "admin2",  
 "realName": "张三2",  
 "password": "123456",  
 "confirmPassword": "123456"  
 },  
 {  
 "userName": "admin2",  
 "realName": "张三2",  
 "password": "123456",  
 "confirmPassword": "123456"  
 }  
]

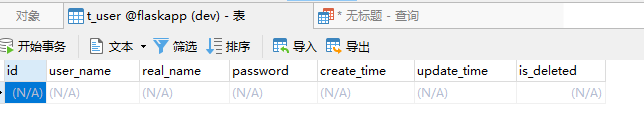
执行结果如下：

存在用户名唯一冲突



去数据库查看如下：

数据插入失败-事务生效-回滚成功



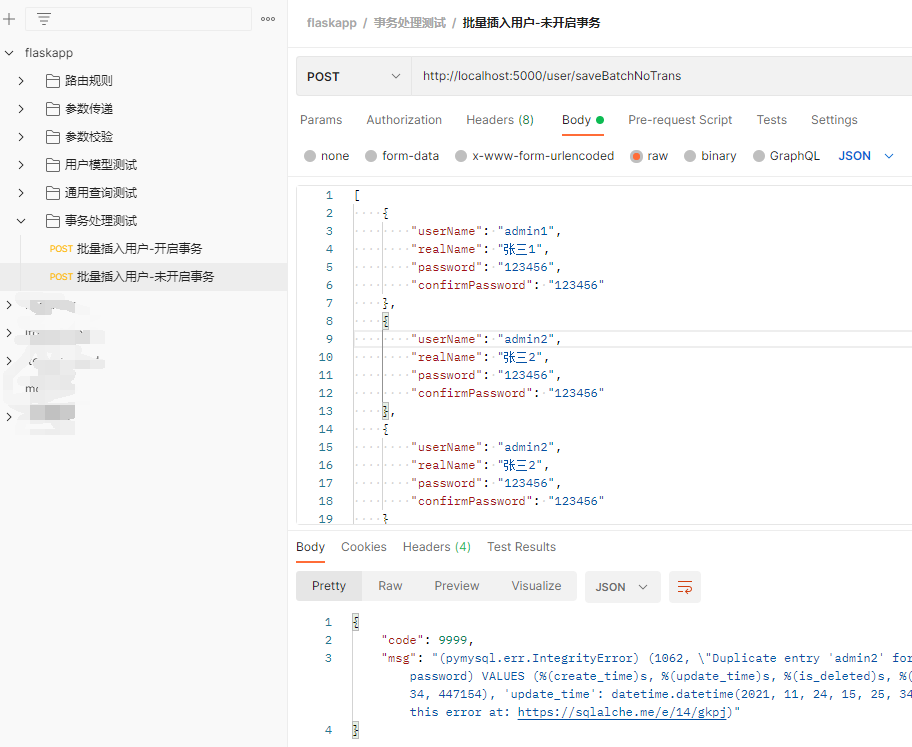
## 访问/user/saveBatchNoTrans

使用相同的数据

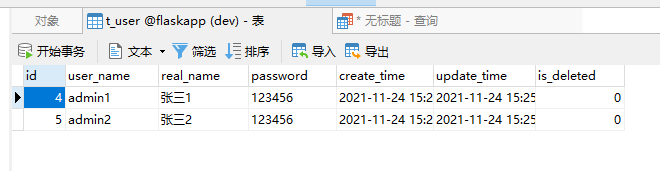
[  
 {  
 "userName": "admin1",  
 "realName": "张三1",  
 "password": "123456",  
 "confirmPassword": "123456"  
 },  
 {  
 "userName": "admin2",  
 "realName": "张三2",  
 "password": "123456",  
 "confirmPassword": "123456"  
 },  
 {  
 "userName": "admin2",  
 "realName": "张三2",  
 "password": "123456",  
 "confirmPassword": "123456"  
 }  
]

执行结果如下：

一样存在用户名冲突



但是前两条记录插入成功了-没回滚



至此，事务验证结束。

# postman导出文件

