**11.2单元测试·测试Flask应用**

前面说过，我们除了可以使用Postman测试Flask服务，还能使用pytest这类单元测试工具来测试。这里先简单演示一下。

# 测试模型层

## 新增tests/conftest.py

实例化Flask，并做一些测试前的准备工作。

代码如下：

import pytest  
from flask import Flask  
  
from config import getConfig  
from controllers.role\_controller import role  
from controllers.user\_controller import user  
from mauth.mtoken import redis\_client  
from mexception import ExceptionConfig  
from mlogging import LoggingConfig  
from mlogging.request\_log import RequestLog  
from models import db  
  
app = Flask(\_\_name\_\_)  
  
# 注册用户模块  
app.register\_blueprint(user)  
# 注册角色模块  
app.register\_blueprint(role)  
  
# 从配置对象中加载  
app.config.from\_ｏｂｊｅｃｔ(getConfig())  
  
# 初始化db  
db.init\_app(app)  
# 初始化redis  
redis\_client.init\_app(app)  
# 配置日志  
LoggingConfig(app)  
# 配置请求日志  
RequestLog(app)  
# 配置异常处理  
ExceptionConfig(app)  
  
  
@pytest.fixture  
def m\_db():  
 with app.app\_context():  
 yield db

一此说明：

* 因为app.py不是模块，不能直接导入，这里只能先简单复制一份过来修改，后续再优化。
* @pytest.fixture叫固定装置，可以理解为，在执行test\_\*方法前进行一些初始化动作
* 在这里就是初始化一个变量叫m\_db
* 这里注意一点就是方法名不能与上面的变量名一样，如上面已经有db了，这里不能重名。
* app.app\_context()，不这样做的话，SQLAlchemy没办法实例化

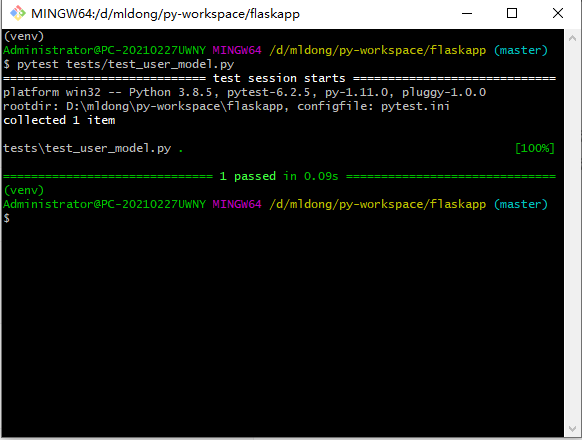
新增tests/test\_user\_model.py

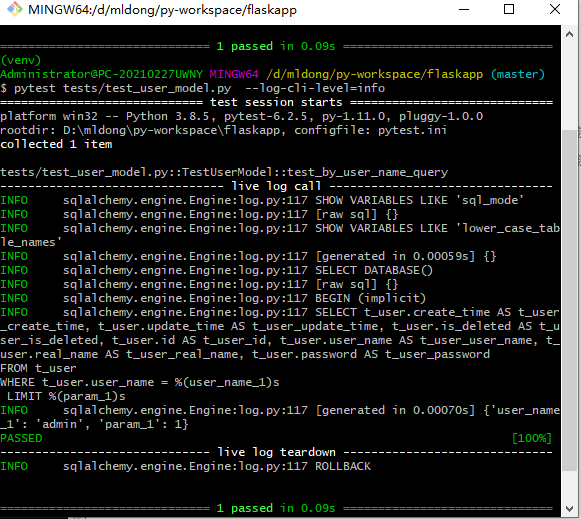
方法中的m\_db参数名要和上面@pytest.fixture定义的方法名称保持一致

from models.user import User  
  
  
class TestUserModel:  
 """  
 User模型测试  
 """  
 def test\_by\_user\_name\_query(self, m\_db):  
 """  
 通过用户名查询测试  
 :param m\_db:  
 :return:  
 """  
 u = m\_db.session.query(User).filter(User.user\_name == "admin").first()  
 assert u

## 执行测试命令

# 默认无日志输出  
pytest tests/test\_user\_model.py  
# 输出日志  
pytest tests/test\_user\_model.py --log-cli-level=info





## 根目录新增pytest.ini文件

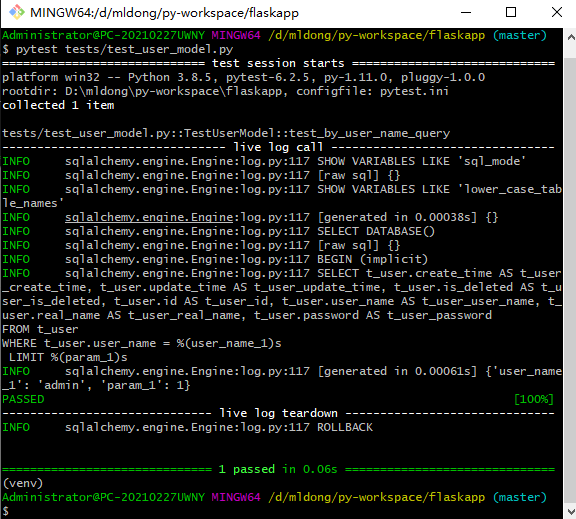
新增内容如下：

[pytest]  
log\_cli = 1  
log\_cli\_level = DEBUG

注：默认是0，所以无日志输出，需要修改成True，才会有日志输出。

## 再次执行测试命令

pytest tests/test\_user\_model.py



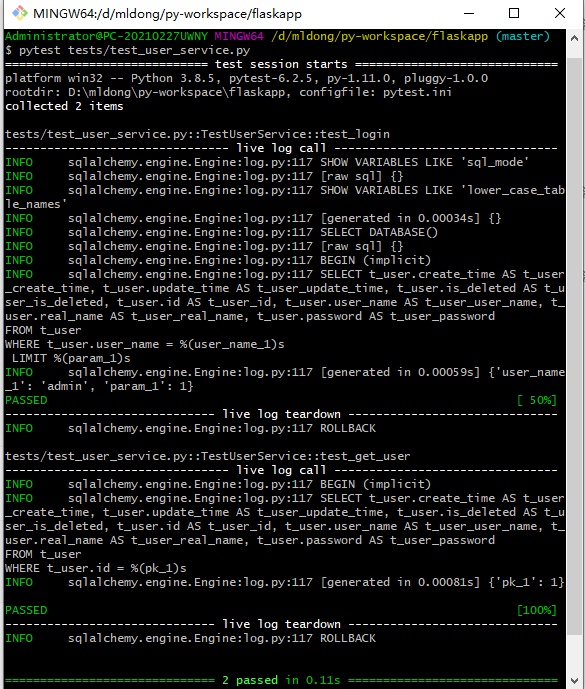
# 测试业务逻辑层

## 新增tests/test\_user\_service.py

from models.user import User  
from services.user\_service import UserService  
from validators.id\_validator import IdForm  
  
  
class TestUserService:  
 """  
 用户业务逻辑测试  
 """  
 def test\_login(self, m\_db):  
 """  
 测试登录  
 :param m\_db:  
 :return:  
 """  
 user\_service = UserService(db=m\_db, model=User)  
 assert user\_service.login("admin", "123456")  
  
 def test\_get\_user(self, m\_db):  
 """  
 测试通过id查询用户信息  
 :param m\_db:  
 :return:  
 """  
 user\_service = UserService(db=m\_db, model=User)  
 form = IdForm(data={  
 "id": 1  
 })  
 assert user\_service.get(form)

## 执行测试命令

pytest tests/test\_user\_service.py



# 测试控制层

控制层测试需要注入客户端，所以需要修改conftest.py

## 修改tests/conftest.py

import pytest  
from flask import Flask  
  
from config import getConfig  
from controllers.role\_controller import role  
from controllers.user\_controller import user  
from mauth.mtoken import redis\_client, TokenStrategyFactory  
from mexception import ExceptionConfig  
from mlogging import LoggingConfig  
from mlogging.request\_log import RequestLog  
from models import db  
  
app = Flask(\_\_name\_\_)  
  
# 注册用户模块  
app.register\_blueprint(user)  
# 注册角色模块  
app.register\_blueprint(role)  
  
# 从配置对象中加载  
app.config.from\_ｏｂｊｅｃｔ(getConfig())  
  
# 初始化db  
db.init\_app(app)  
# 初始化redis  
redis\_client.init\_app(app)  
# 配置日志  
LoggingConfig(app)  
# 配置请求日志  
RequestLog(app)  
# 配置异常处理  
ExceptionConfig(app)  
  
  
@pytest.fixture  
def client():  
 # 装备一个客户端给test\_控制层方法使用  
 with app.test\_client() as client:  
 ctx = app.app\_context()  
 ctx.push()  
 yield client # this is where the testing happens!  
 ctx.pop()  
  
  
@pytest.fixture  
def auth\_client():  
 """  
 装备一个已授权的客户端给test\_控制层方法使用  
 :return:  
 """  
 with app.test\_client() as client:  
 ctx = app.app\_context()  
 ctx.push()  
 token\_strategy = TokenStrategyFactory.get\_instance()  
 data = token\_strategy.set({  
 "userId": 1,  
 "userName": "admin"  
 })  
 # HTTP\_KEY的方式存入全局请求头  
 client.environ\_base['HTTP\_TOKEN'] = data.get('token')  
 yield client # this is where the testing happens!  
 ctx.pop()  
  
  
@pytest.fixture  
def m\_db():  
 """  
 装配db对象给test\_模型层和服务逻辑层方法使用  
 :return:  
 """  
 with app.app\_context():  
 yield db

一些说明：

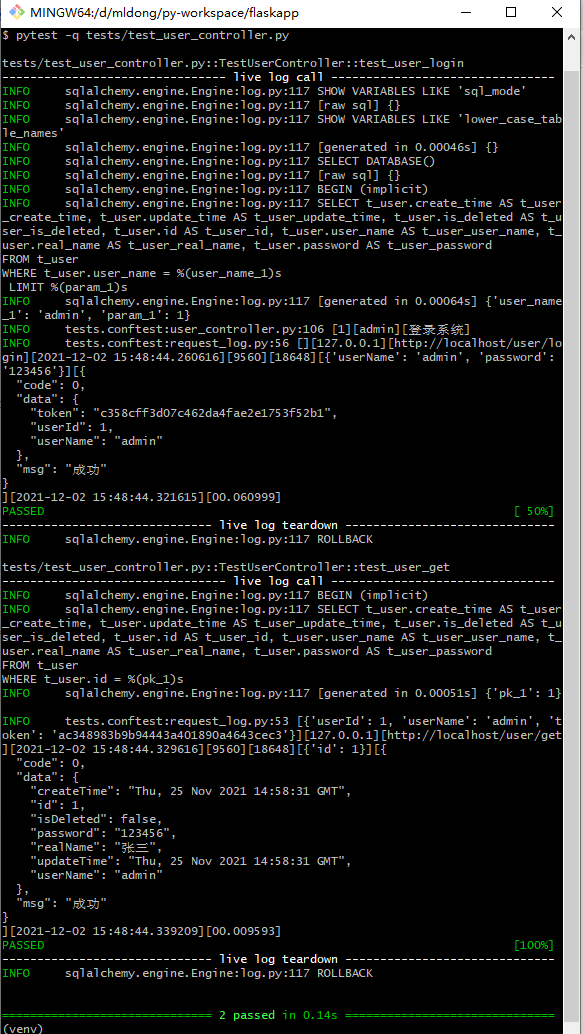
* 这里共注入了两个客户端client和auth\_client
* auth\_client将创建一个token并将token注入到请求头

## 新增tests/user\_controller.py

import json  
  
  
class TestUserController:  
 """  
 用户控制层单元测试类  
 """  
 def test\_user\_login(self, client):  
 """  
 测试登录  
 :param client:  
 :return:  
 """  
 res = client.post('/user/login', data=json.dumps({  
 "userName": "admin",  
 "password": "123456"  
 }), content\_type='application/json')  
 assert res.json.get('code') == 0  
  
 def test\_user\_get(self, auth\_client):  
 """  
 注册通过id获取用户信息  
 :param auth\_client:  
 :return:  
 """  
 res = auth\_client.post('/user/get', data=json.dumps({  
 "id": 1  
 }), content\_type='application/json')  
 assert res.json.get('code') == 0

## 执行测试命令

pytest -q tests/test\_user\_controller.py



# 关于@pytest.fixture的一些说明

### fixture的scope参数

scope参数有四种，分别是'function','module','class','session'，默认为function。

* function：每个test都运行，默认是function的scope
* class：每个class的所有test只运行一次
* module：每个module的所有test只运行一次
* session：每个session只运行一次