**12.4代码生成·代码生成模板制作**

上文中做的代码生成器，使用的模板引擎是Jinja2，本文使用Jinja2制作另外几个模块的模板。

# Jinja2语法

## 变量取值

{{table.tableName}}-{{table.comment}}

## 条件判断-if

{% if table.tableName == 't\_role'%}  
 我是角色表  
{%endif%}

## 条件判断-if-elif

{% if table.tableName == 't\_user'%}  
 我是用户表  
{% elif table.tableName == 't\_role'%}  
 我是角色表  
{%endif%}

## 条件判断-if-elif-else

{% if table.tableName == 't\_user'%}  
 我是用户表  
{% elif table.tableName == 't\_access\_token'%}  
 我是token表  
{% else %}  
 我是角色表  
{%endif%}

## 遍历

{% for column in table.columns %}  
 {{column.name}}-{{column.comment}}  
{% endfor %}

## 遍历+条件判断

{% for column in table.columns %}  
 {%if column.name not in ['id','create\_time','update\_time','is\_deleted']%}  
 {{column.name}}-{{column.comment}}  
 {% endif %}  
{% endfor %}

## 过滤器

过虑器即为Jinja2内置的一些处理函数

* safe: 渲染时值不转义
* capitialize: 把值的首字母转换成大写，其他子母转换为小写
* lower: 把值转换成小写形式
* upper: 把值转换成大写形式
* title: 把值中每个单词的首字母都转换成大写
* trim: 把值的首尾空格去掉
* striptags: 渲染之前把值中所有的HTML标签都删掉
* join: 拼接多个值为字符串
* replace: 替换字符串的值
* round: 默认对数字进行四舍五入，也可以用参数进行控制
* int: 把值转换成整型

{{table.tableName}}------{{table.tableName | lower }}  
{{table.tableName}}------{{table.tableName | replace('t\_','') }}

## 自定义过滤器

除了使用内置处理函数外，我们还可以自定义一些函数，使用方式同上。

env = Environment(loader=FileSystemLoader('generator/templates'))  
# 增加自定义过滤器-my\_fun  
env.filters['my\_fun'] = self.my\_fun

# Jinja单元测试

## 修改tests/conftest.py

新增m\_table装配

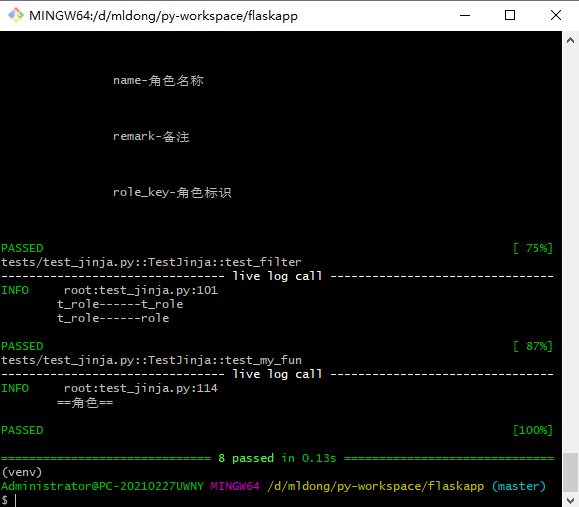
import pytest  
from flask import Flask  
  
from config import getConfig  
from controllers.role\_controller import role  
from controllers.user\_controller import user  
from generator import CodeGenerator  
from mauth.mtoken import redis\_client, TokenStrategyFactory  
from mexception import ExceptionConfig  
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  
  
  
@pytest.fixture(scope='class')  
def m\_table():  
 """  
 装配m\_table对象给test\_模型层和服务逻辑层方法使用  
 :return:  
 """  
 with app.app\_context():  
 code\_generator = CodeGenerator(app)  
 table = code\_generator.build\_table('t\_role')  
 yield table

## 新增tests/test\_jinja.py

import logging  
  
from jinja2 import Environment, FileSystemLoader  
  
  
class TestJinja:  
 def test\_value(self, m\_table):  
 env = Environment(loader=FileSystemLoader('generator/templates'))  
 template = """  
 {{table.tableName}}-{{table.comment}}  
 """  
 templateData = {  
 'table': m\_table  
 }  
 s = env.from\_string(template).render(templateData)  
 logging.info(s)  
  
 def test\_if(self, m\_table):  
 env = Environment(loader=FileSystemLoader('generator/templates'))  
 template = """  
 {% if table.tableName == 't\_role'%}  
 我是角色表  
 {%endif%}  
 """  
 templateData = {  
 'table': m\_table  
 }  
 s = env.from\_string(template).render(templateData)  
 logging.info(s)  
  
 def test\_if\_elif(self, m\_table):  
 env = Environment(loader=FileSystemLoader('generator/templates'))  
 template = """  
 {% if table.tableName == 't\_user'%}  
 我是用户表  
 {% elif table.tableName == 't\_role'%}  
 我是角色表  
 {%endif%}  
 """  
 templateData = {  
 'table': m\_table  
 }  
 s = env.from\_string(template).render(templateData)  
 logging.info(s)  
  
 def test\_if\_elif\_else(self, m\_table):  
 env = Environment(loader=FileSystemLoader('generator/templates'))  
 template = """  
 {% if table.tableName == 't\_user'%}  
 我是用户表  
 {% elif table.tableName == 't\_access\_token'%}  
 我是token表  
 {% else %}  
 我是角色表  
 {%endif%}  
 """  
 templateData = {  
 'table': m\_table  
 }  
 s = env.from\_string(template).render(templateData)  
 logging.info(s)  
  
 def test\_for(self, m\_table):  
 env = Environment(loader=FileSystemLoader('generator/templates'))  
 template = """  
 {% for column in table.columns %}  
 {{column.name}}-{{column.comment}}  
 {% endfor %}  
 """  
 templateData = {  
 'table': m\_table  
 }  
 s = env.from\_string(template).render(templateData)  
 logging.info(s)  
  
 def test\_for\_if(self, m\_table):  
 env = Environment(loader=FileSystemLoader('generator/templates'))  
 template = """  
 {% for column in table.columns %}  
 {%if column.name not in ['id','create\_time','update\_time','is\_deleted']%}  
 {{column.name}}-{{column.comment}}  
 {% endif %}  
 {% endfor %}  
 """  
 templateData = {  
 'table': m\_table  
 }  
 s = env.from\_string(template).render(templateData)  
 logging.info(s)  
  
 def test\_filter(self, m\_table):  
 env = Environment(loader=FileSystemLoader('generator/templates'))  
 template = """  
 {{table.tableName}}------{{table.tableName | lower }}  
 {{table.tableName}}------{{table.tableName | replace('t\_','') }}  
 """  
 templateData = {  
 'table': m\_table  
 }  
 s = env.from\_string(template).render(templateData)  
 logging.info(s)  
  
 def test\_my\_fun(self, m\_table):  
 env = Environment(loader=FileSystemLoader('generator/templates'))  
 # 增加自定义过滤器-my\_fun  
 env.filters['my\_fun'] = self.my\_fun  
 template = """  
 {{ table.comment | my\_fun }}  
 """  
 templateData = {  
 'table': m\_table  
 }  
 s = env.from\_string(template).render(templateData)  
 logging.info(s)  
  
 @staticmethod  
 def my\_fun(text):  
 return "=="+text+"=="

## 执行单元测试命令

pytest -q tests/test\_jinja.py



# 制作模板

## 制作validators层模板

新增generator/templates/validator.ftl文件

import wtforms  
  
from validators import BaseForm, BasePageForm  
  
  
class {{table.tableHumpName}}Form(BaseForm):  
 """  
 {{table.comment}}表单校验类  
 """  
 id = wtforms.IntegerField(){% for column in table.columns %}{%if column.name not in ['id','create\_time','update\_time','is\_deleted']%}  
 {{column.camelName}} = wtforms.StringField("{{column.comment}}"{%if not column.nullable%}, [wtforms.validators.DataRequired(message="{{column.comment}}不能为空")]{%endif%}){%endif%}{% endfor %}  
  
  
class {{table.tableHumpName}}PageForm(BasePageForm):  
 """  
 {{table.comment}}分页校验类  
 """  
 pass

## 制作services层模板

新增generator/templates/service.ftl文件

from services import BaseService  
  
  
class {{table.tableHumpName}}Service(BaseService):  
 """  
 {{table.comment}}模块业务处理类  
 """

## 制作controllers层模板

新增generator/templates/controller.ftl文件

from flask import Blueprint  
  
from controllers import R  
from mauth import HasPerm  
from models.{{table.shortTableName}} import {{table.tableHumpName}}  
from services.{{table.shortTableName}}\_service import {{table.tableHumpName}}Service  
from validators import BasePageForm  
from validators.id\_validator import IdForm, IdsForm  
from validators.{{table.shortTableName}}\_validator import {{table.tableHumpName}}Form  
  
# 声明一个蓝图  
{{table.shortTableName}} = Blueprint('{{table.shortTableName}}', \_\_name\_\_, url\_prefix="/{{table.shortTableName}}")  
  
# 声明一个{{table.comment}}业务服务  
{{table.shortTableName}}\_service = {{table.tableHumpName}}Service(model={{table.tableHumpName}})  
  
  
@{{table.shortTableName}}.route("/get", methods=['POST'])  
@HasPerm(access="{{table.shortTableName}}:get", name="通过id获取{{table.comment}}信息")  
def {{table.shortTableName}}\_get():  
 """  
 通过id获取{{table.comment}}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 = {{table.tableHumpName}}.query.filter\_by(id=form.id.data).first()  
 # 通过主键查询  
 u = {{table.shortTableName}}\_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")  
  
  
@{{table.shortTableName}}.route("/list", methods=['POST'])  
@HasPerm(access="{{table.shortTableName}}:list", name="分页查询{{table.comment}}列表")  
def {{table.shortTableName}}\_list():  
 """  
 分页查询{{table.comment}}列表  
 :return:  
 """  
 form = BasePageForm()  
 form.validate\_for\_api()  
 return R.data({{table.shortTableName}}\_service.list(form))  
  
  
@{{table.shortTableName}}.route("/save", methods=['POST'])  
@HasPerm(access="{{table.shortTableName}}:save", name="添加{{table.comment}}")  
def {{table.shortTableName}}\_save():  
 """  
 添加{{table.comment}}  
 :return:  
 """  
 form = {{table.tableHumpName}}Form()  
 form.validate\_for\_api()  
 # 可通过form.data获取所有提交参数  
 # print(form.data)  
 {{table.shortTableName}}\_service.save(form)  
 return R.success("添加{{table.comment}}成功")  
  
  
@{{table.shortTableName}}.route("/update", methods=['POST'])  
@HasPerm(access="{{table.shortTableName}}:update", name="修改{{table.comment}}")  
def {{table.shortTableName}}\_update():  
 """  
 修改{{table.comment}}  
 :return:  
 """  
 form = {{table.tableHumpName}}Form()  
 form.validate\_for\_api()  
 # 可通过form.data获取所有提交参数  
 # print(form.data)  
 {{table.shortTableName}}\_service.update(form)  
 return R.success("修改{{table.comment}}成功")  
  
  
@{{table.shortTableName}}.route("/delete", methods=['POST'])  
@HasPerm(access="{{table.shortTableName}}:delete", name="删除{{table.comment}}")  
def {{table.shortTableName}}\_delete():  
 """  
 删除{{table.comment}}  
 :return:  
 """  
 form = IdsForm()  
 form.validate\_for\_api()  
 # 可通过form.data获取所有提交参数  
 # print(form.data)  
 {{table.shortTableName}}\_service.delete(form)  
 return R.success("删除{{table.comment}}成功")

## 修改配置文件generator/config.json

{  
 "targetProject": "",  
 "templates": [  
 {  
 "templateFile": "model.ftl",  
 "targetPath": "models/",  
 "targetFileName": "{{table.shortTableName}}.py",  
 "selected": true,  
 "covered": true,  
 "encoding": "utf-8"  
 },  
 {  
 "templateFile": "validator.ftl",  
 "targetPath": "validators/",  
 "targetFileName": "{{table.shortTableName}}\_validator.py",  
 "selected": true,  
 "covered": true,  
 "encoding": "utf-8"  
 },  
 {  
 "templateFile": "service.ftl",  
 "targetPath": "services/",  
 "targetFileName": "{{table.shortTableName}}\_service.py",  
 "selected": true,  
 "covered": true,  
 "encoding": "utf-8"  
 },  
 {  
 "templateFile": "controller.ftl",  
 "targetPath": "controllers/",  
 "targetFileName": "{{table.shortTableName}}\_controller.py",  
 "selected": true,  
 "covered": true,  
 "encoding": "utf-8"  
 }  
 ]  
}

# 执行代码生成命令

flask code gen -t t\_role

