**12.3代码生成·Flask的Db First和代码生成器**

上文中我们知道Code First是可以根据ORM模型转成数据库表的，而本文的Db First则是反过来的，就是将数据库表映射成ORM模型。而这个关键的技术就是代码生成器——通过读取数据库元数据，然后定制模板，使用模板引擎输出自定义页面或文件的工具。

# 一些说明

SQLAlchemy周边是有一个工具可以将数据库表转成ORM模型的，叫sqlacodegen，本工程并不打算讲这个工具，因为这个工具也有自己的局限性：

1. 不能自定义模板

我们有自己的模型基类，有自己的模型样例模板

1. 不能新增自定义模板

我们在生成ORM模型的同时，还需要生成validators/services/controller层的代码。

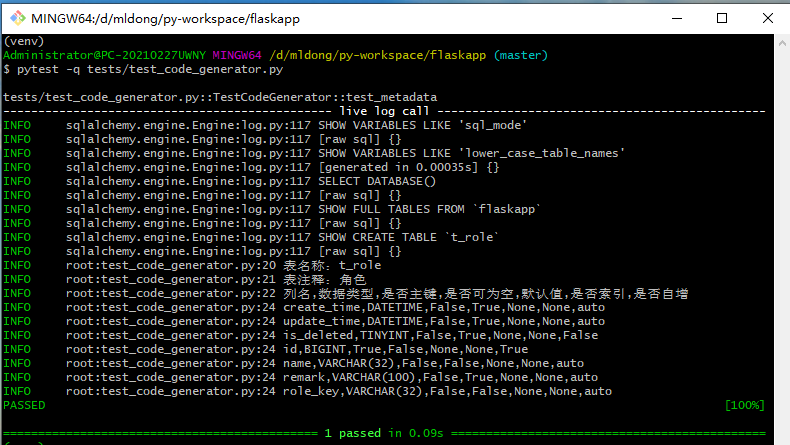
# 获取元数据的方式

## 新建单元测试类文件test\_code\_generator.py

import logging  
  
from sqlalchemy import ｍｅｔａData  
  
  
class TestCodeGenerator:  
 """  
 代码生成器单元测试类  
 """  
 def test\_ｍｅｔａdata(self, m\_db):  
 """  
 简单的测试获取元数据  
 :param m\_db:  
 :return:  
 """  
 ｍｅｔａdata\_obj = ｍｅｔａData()  
 ｍｅｔａdata\_obj.reflect(bind=m\_db.engine, only=["t\_role"])  
 tables = ｍｅｔａdata\_obj.tables  
 for table in tables.values():  
 logging.info(f"表名称：{table.name}")  
 logging.info(f"表注释：{table.comment}")  
 logging.info(f"列名,数据类型,是否主键,是否可为空,默认值,是否索引,是否自增")  
 for col in table.columns:  
 logging.info(f"{col.name},{col.type},{col.primary\_key},"  
 f"{col.nullable},{col.default},"  
 f"{col.index},{col.autoincrement}")

## 执行测试命令

pytest -q tests/test\_code\_generator.py



## 一些说明

* ｍｅｔａdata\_obj.reflect(bind=m\_db.engine, only=["t\_role"])

1. bind为执行引擎
2. only为要查询的表名称数组

* ｍｅｔａdata\_obj.tables为表元数据信息集合
* table.name 表名称
* table.comment 表注释
* table.columns 表列的元数据信息集合

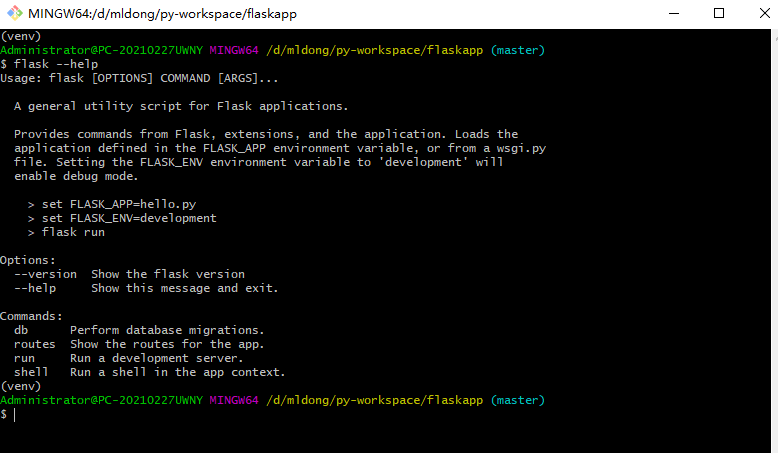
1. col.name 列名
2. col.type 列数据类型
3. col.primary\_key 是否主键
4. col.nullable 是否可为空
5. col.default 默认值
6. col.index 索引
7. col.autoincrement是否自增

# 关于Flask-cli自定义命令

我们运行Flask服务，使用的是flask run、flask db命令，这里都是内置命令，如果我们要自定义命令，要如何做呢？下面进行一个小小的例子。

## 查看Flask-cli默认命令

flask --help



## 创建目录generator

mkdir generator

## 新增文件generator/\_\_init\_\_.py

import click  
from flask.cli import AppGroup  
  
  
class CodeGenerator:  
 """  
 代码生成器  
 """  
 gen\_cli = AppGroup('code', help="代码生成相关命令")  
  
 def \_\_init\_\_(self, app=None):  
 self.app = None  
 if app:  
 self.init\_app(app)  
  
 def init\_app(self, app):  
 self.app = app  
 self.init\_cli()  
  
 def init\_cli(self):  
 """  
 注册代码生成相关命令  
 :return:  
 """  
  
 @self.gen\_cli.command('gen', help="代码生成")  
 @click.option('-t', required=True, help="要代码生成的表")  
 def gen(t):  
 print(f"gen:{t}")  
  
 @self.gen\_cli.command('show', help="查看元数据")  
 @click.option('-t', required=True, help="要查看元数据的表")  
 def show(t):  
 print(f"show:{t}")  
  
 self.app.cli.add\_command(self.gen\_cli)

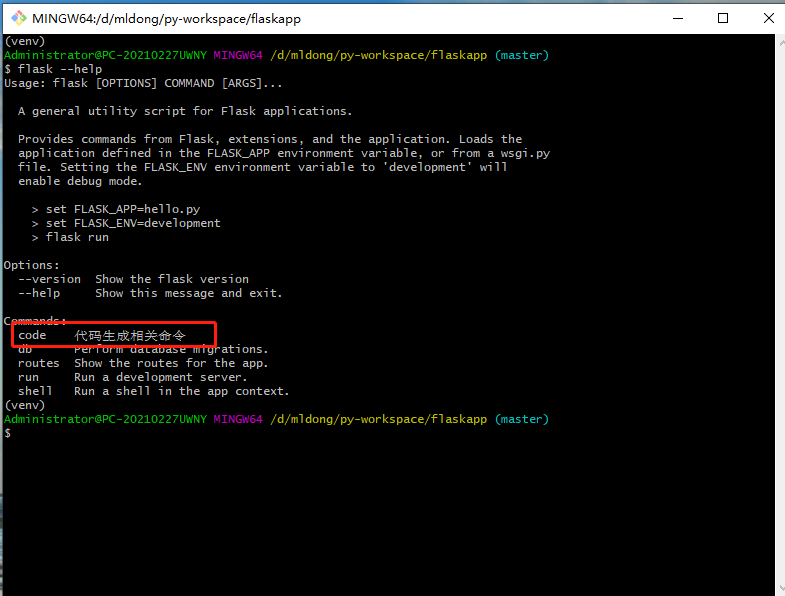
## 修改app.py

注册代码生成器相关命令

from flask import Flask  
from flask\_migrate import Migrate  
  
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)  
# 执行数据库迁移相关操作。flask db指令不能直接使用，需要获取Migrate实例，这里注册Migrate实例  
migrate = Migrate(app, db)  
# 配置代码生成器-仅开发环境配置  
if app.config.get("ENV") == "dev":  
 from generator import CodeGenerator  
 CodeGenerator(app)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(host="0.0.0.0")

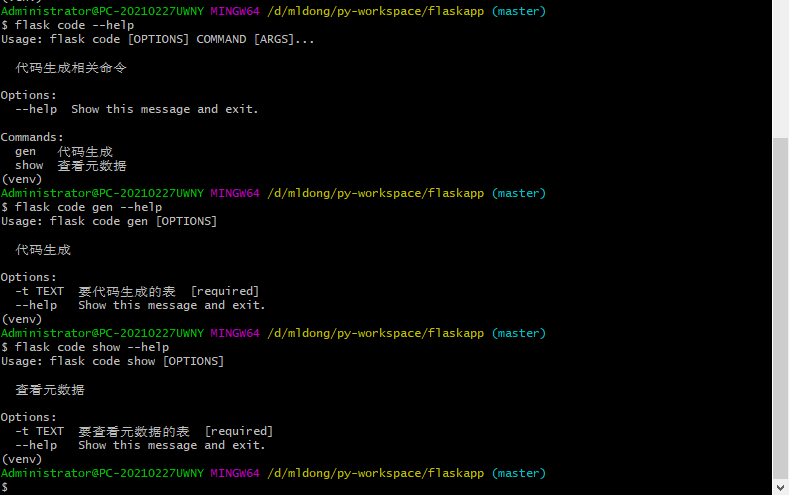
## 执行flask帮助命令

flask --help



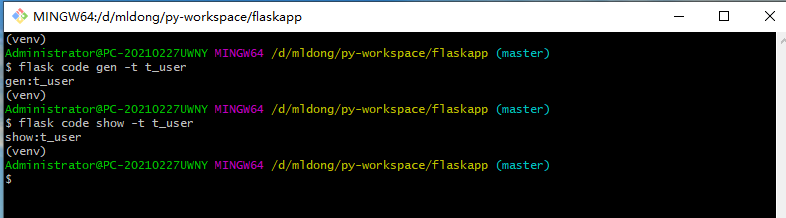
## 进一步查看命令

flask code --help  
flask code gen --help  
flask code show --help



## 执行具体的命令

flask code gen -t t\_user  
flask code show -t t\_user



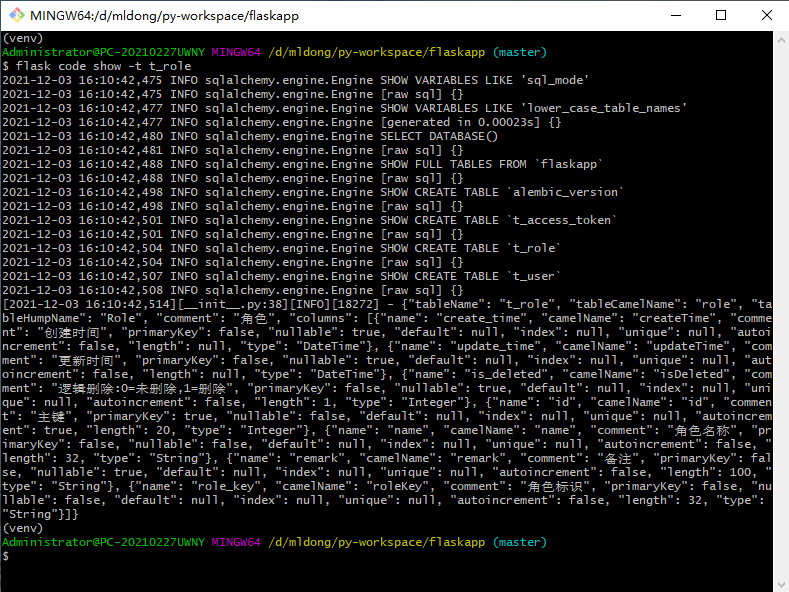
# 获取并重组元数据

## 修改generator/\_\_init\_\_.py

import json  
import re  
  
import click  
from flask.cli import AppGroup  
from flask\_sqlalchemy import SQLAlchemy  
from sqlalchemy import ｍｅｔａData  
  
from tools import underline\_to\_camel, underline\_to\_hump  
  
  
class CodeGenerator:  
 """  
 代码生成器  
 """  
 db = None  
 engine = None  
 table\_list = []  
 gen\_cli = AppGroup('code', help="代码生成相关命令")  
  
 def \_\_init\_\_(self, app=None):  
 self.app = None  
 if app:  
 self.init\_app(app)  
  
 def init\_app(self, app):  
 self.app = app  
 self.db = SQLAlchemy(app)  
 self.engine = self.db.engine  
 self.init\_cli()  
  
 def print\_table(self):  
 """  
 打印元数据信息  
 :return:  
 """  
 for table in self.table\_list:  
 self.app.logger.info(json.dumps(table, sort\_keys=False, ensure\_ascii=False))  
  
 def build\_table(self, tableName: str = "", ignorePrefix: str = "t\_"):  
 """  
 根据表名称构建元数据-重新组织元数据  
 :param tableName: 表名称  
 :param ignorePrefix: 转小驼峰和大驼峰时要忽略的表前辍，默认t\_  
 :return:  
 """  
 tableName = tableName.replace("%", "")  
 ｍｅｔａdata\_obj = ｍｅｔａData()  
 ｍｅｔａdata\_obj.reflect(bind=self.engine)  
 tables = ｍｅｔａdata\_obj.tables  
 self.table\_list = []  
 for table in tables.values():  
 if not table.name.startswith(tableName):  
 continue  
 columns = []  
 m\_table = {  
 "tableName": table.name,  
 "tableCamelName": underline\_to\_camel(table.name.replace(ignorePrefix, "")),  
 "tableHumpName": underline\_to\_hump(table.name.replace(ignorePrefix, "")),  
 "comment": table.comment,  
 "columns": columns  
 }  
 for col in table.columns:  
 m\_type = str(col.type.get\_dbapi\_type)  
 # <bound method Integer.get\_dbapi\_type of INTEGER(display\_width=11)>  
 # <bound method String.get\_dbapi\_type of VARCHAR(length=32)>  
 # <bound method DateTime.get\_dbapi\_type of DATETIME()>  
 r = re.findall('.+method (.+)\.get\_dbapi\_type', m\_type)  
 m\_type = r[0]  
 columns.append({  
 "name": col.name,  
 "camelName": underline\_to\_camel(col.name),  
 "comment": col.comment,  
 "primaryKey": col.primary\_key,  
 "nullable": col.nullable,  
 "default": col.default,  
 "index": col.index,  
 "unique": col.unique,  
 "autoincrement": col.autoincrement is True,  
 "length": getattr(col.type, 'display\_width', getattr(col.type, 'length', None)),  
 "type": m\_type  
 })  
 self.table\_list.append(m\_table)  
 return m\_table  
  
 def init\_cli(self):  
 """  
 注册代码生成相关命令  
 :return:  
 """  
  
 @self.gen\_cli.command('gen', help="代码生成")  
 @click.option('-t', required=True, help="要代码生成的表")  
 @click.option('-ignore\_prefix', default="t\_", required=False, help="忽略的表前辍")  
 def gen(t, ignore\_prefix):  
 print(f"gen{t},{ignore\_prefix}")  
  
 @self.gen\_cli.command('show', help="查看元数据")  
 @click.option('-t', required=True, help="要查看元数据的表")  
 @click.option('-ignore\_prefix', default="t\_", help="忽略的表前辍")  
 def show(t, ignore\_prefix):  
 self.build\_table(t, ignorePrefix=ignore\_prefix)  
 self.print\_table()  
  
 self.app.cli.add\_command(self.gen\_cli)

## 执行查看元数据命令

flask code show -t t\_role



## 完整的元数据样例

{  
 "tableName": "t\_role",  
 "tableCamelName": "role",  
 "tableHumpName": "Role",  
 "comment": "角色",  
 "columns": [{  
 "name": "create\_time",  
 "camelName": "createTime",  
 "comment": "创建时间 ",  
 "primaryKey ": false,  
 "nullable ": true,  
 "default ": null,  
 "index ": null,  
 "unique ": null,  
 "autoincrement ": false,  
 "length ": null,  
 "type ": "DateTime "  
 }, {  
 "name ": "update\_time ",  
 "camelName ": "updateTime ",  
 "comment ": "更新时间 ",  
 "primaryKey ": false,  
 "nullable ": true,  
 "default ": null,  
 "index ": null,  
 "unique ": null,  
 "autoincrement ": false,  
 "length ": null,  
 "type ": "DateTime "  
 }, {  
 "name ": "is\_deleted ",  
 "camelName ": "isDeleted ",  
 "comment ": "逻辑删除: 0 = 未删除,1 = 删除 ",  
 "primaryKey ": false,  
 "nullable ": true,  
 "default ": null,  
 "index ": null,  
 "unique ": null,  
 "autoincrement ": false,  
 "length ": 1,  
 "type ": "Integer "  
 }, {  
 "name ": "id ",  
 "camelName ": "id ",  
 "comment ": "主键 ",  
 "primaryKey ": true,  
 "nullable ": false,  
 "default": null,  
 "index": null,  
 "unique": null,  
 "autoincrement": true,  
 "length": 20,  
 "type": "Integer"  
 }, {  
 "name": "name",  
 "camelName": "name",  
 "comment": "角色名称",  
 "primaryKey": false,  
 "nullable": false,  
 "default": null,  
 "index": null,  
 "unique ": null,  
 "autoincrement ": false,  
 "length ": 32,  
 "type ": "String "  
 }, {  
 "name ": "remark ",  
 "camelName ": "remark ",  
 "comment ": "备注 ",  
 "primaryKey ": false,  
 "nullable ": true,  
 "default ": null,  
 "index ": null,  
 "unique ": null,  
 "autoincrement ": false,  
 "length ": 100,  
 "type ": "String "  
 }, {  
 "name ": "role\_key",  
 "camelName ": "roleKey ",  
 "comment ": "角色标识 ",  
 "primaryKey ": false,  
 "nullable ": false,  
 "default ": null,  
 "index ": null,  
 "unique ": null,  
 "autoincrement ": false,  
 "length": 32,  
 "type": "String"  
 }]  
}

## 数据结构说明

### 表说明

|  |  |  |
| --- | --- | --- |
| 名称 | 样例 | 描述 |
| tableName | t\_role | 表名 |
| tableCamelName | role | 表去前辍后的小驼峰 |
| tableHumpName | Role | 表去前辍后的大驼峰 |
| comment | 角色 | 表注释 |
| columns |  | 表字段集合 |

### 表字段

|  |  |  |
| --- | --- | --- |
| 名称 | 样例 | 描述 |
| name | role\_key | 字段名 |
| camelName | roleKey | 字段小驼峰 |
| comment | 角色标识 | 字段注释 |
| primaryKey | false | 是否主键 |
| nullable | false | 是否可为空 |
| default | null | 默认值 |
| index | null | 是否索引 |
| unique | null | 是否唯一 |
| autoincrement | false | 是否自增 |
| length | 32 | 字段长度 |
| type | String | SQLAlchemy对应的数据类型 |

# 生成代码模块

这里使用到jinja2模板引擎生成代码

## 修改generator/\_\_init\_\_.py

新增gen\_code方法

import json  
import os  
import re  
  
import click  
from flask.cli import AppGroup  
from flask\_sqlalchemy import SQLAlchemy  
from jinja2 import Environment, FileSystemLoader  
from sqlalchemy import ｍｅｔａData  
  
from tools import underline\_to\_camel, underline\_to\_hump  
  
  
class CodeGenerator:  
 """  
 代码生成器  
 """  
 db = None  
 engine = None  
 table\_list = []  
 gen\_cli = AppGroup('code', help="代码生成相关命令")  
 config = None  
  
 def \_\_init\_\_(self, app=None):  
 self.app = None  
 if app:  
 self.init\_app(app)  
  
 def init\_app(self, app):  
 self.app = app  
 self.init\_config()  
 self.db = SQLAlchemy(app)  
 self.engine = self.db.engine  
 self.init\_cli()  
  
 def print\_table(self):  
 """  
 打印元数据信息  
 :return:  
 """  
 for table in self.table\_list:  
 self.app.logger.info(json.dumps(table, sort\_keys=False, ensure\_ascii=False))  
  
 def build\_table(self, tableName: str = "", ignorePrefix: str = "t\_"):  
 """  
 根据表名称构建元数据-重新组织元数据  
 :param tableName: 表名称  
 :param ignorePrefix: 转小驼峰和大驼峰时要忽略的表前辍，默认t\_  
 :return:  
 """  
 tableName = tableName.replace("%", "")  
 ｍｅｔａdata\_obj = ｍｅｔａData()  
 ｍｅｔａdata\_obj.reflect(bind=self.engine)  
 tables = ｍｅｔａdata\_obj.tables  
 self.table\_list = []  
 for table in tables.values():  
 if not table.name.startswith(tableName):  
 continue  
 columns = []  
 m\_table = {  
 "tableName": table.name,  
 "tableCamelName": underline\_to\_camel(table.name.replace(ignorePrefix, "")),  
 "tableHumpName": underline\_to\_hump(table.name.replace(ignorePrefix, "")),  
 "comment": table.comment,  
 "columns": columns  
 }  
 for col in table.columns:  
 m\_type = str(col.type.get\_dbapi\_type)  
 # <bound method Integer.get\_dbapi\_type of INTEGER(display\_width=11)>  
 # <bound method String.get\_dbapi\_type of VARCHAR(length=32)>  
 # <bound method DateTime.get\_dbapi\_type of DATETIME()>  
 r = re.findall('.+method (.+)\.get\_dbapi\_type', m\_type)  
 m\_type = r[0]  
 columns.append({  
 "name": col.name,  
 "camelName": underline\_to\_camel(col.name),  
 "comment": col.comment,  
 "primaryKey": col.primary\_key,  
 "nullable": col.nullable,  
 "default": col.default,  
 "index": col.index,  
 "unique": col.unique,  
 "autoincrement": col.autoincrement is True,  
 "length": getattr(col.type, 'display\_width', getattr(col.type, 'length', None)),  
 "type": m\_type  
 })  
 self.table\_list.append(m\_table)  
 return m\_table  
  
 def init\_cli(self):  
 """  
 注册代码生成相关命令  
 :return:  
 """  
  
 @self.gen\_cli.command('gen', help="代码生成")  
 @click.option('-t', required=True, help="要代码生成的表")  
 @click.option('-ignore\_prefix', default="t\_", required=False, help="忽略的表前辍")  
 def gen(t, ignore\_prefix):  
 self.build\_table(t, ignorePrefix=ignore\_prefix)  
 self.gen\_code()  
  
 @self.gen\_cli.command('show', help="查看元数据")  
 @click.option('-t', required=True, help="要查看元数据的表")  
 @click.option('-ignore\_prefix', default="t\_", help="忽略的表前辍")  
 def show(t, ignore\_prefix):  
 self.build\_table(t, ignorePrefix=ignore\_prefix)  
 self.print\_table()  
  
 self.app.cli.add\_command(self.gen\_cli)  
  
 def init\_config(self):  
 """  
 加载配置文件  
 :return:  
 """  
 with open("generator/config.json", "r") as f:  
 self.config = json.load(f)  
  
 def gen\_code(self):  
 """  
 生成代码  
 """  
 env = Environment(loader=FileSystemLoader('generator/templates'))  
 templates = self.config['templates']  
 for table in self.table\_list:  
 templateData = {}  
 templateData.update(self.config)  
 templateData.update({"table": table})  
 for item in templates:  
 if not item.get("selected"):  
 break  
 template = env.get\_template(item['templateFile'])  
 path = self.config['targetProject'] + item['targetPath']  
 # 配置参数替换-模板引擎  
 path = env.from\_string(path).render(templateData)  
 # 替换包名为目录  
 path = path.replace(".", "/")  
 if not os.path.exists(path):  
 os.makedirs(path)  
 # 配置参数替换-模板引擎  
 targetFileName = env.from\_string(item['targetFileName']).render(templateData)  
 dist = path + targetFileName  
  
 if os.path.exists(dist):  
 if item.get("covered"):  
 with open(dist, 'w', encoding=item.get("encoding")) as f:  
 html = template.render(templateData)  
 f.write(html)  
 print(f"{templateData['table']['tableName']}表代码生成成功-覆盖：{dist}")  
 else:  
 print(f"{dist}文件已存在，不覆盖")  
 else:  
 with open(dist, 'w', encoding=item['encoding']) as f:  
 html = template.render(templateData)  
 f.write(html)  
 print(f"{templateData['table']['tableName']}表代码生成成功-新生成：{dist}")  
 return 1

## 新增配置文件generator/config.json

{  
 "targetProject": "",  
 "templates": [  
 {  
 "templateFile": "model.ftl",  
 "targetPath": "models/",  
 "targetFileName": "{{table.tableCamelName}}.py",  
 "selected": true,  
 "covered": true,  
 "encoding": "utf-8"  
 }  
 ]  
}

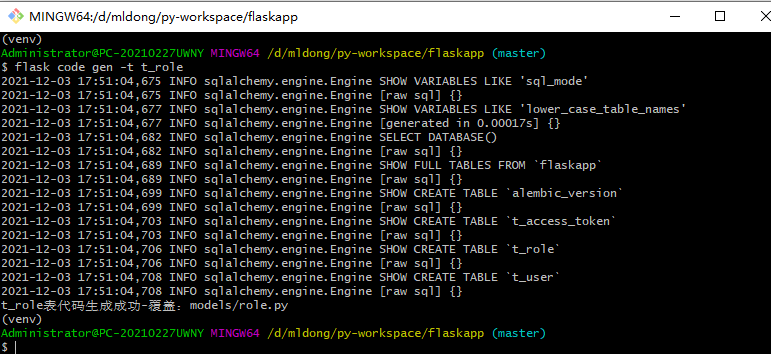
## 新增模板文件generator/template/model.ftl

这里暂时只创建orm模型的模板文件

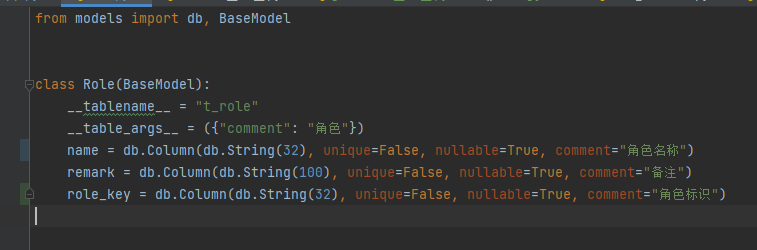
from models import db, BaseModel  
  
  
class {{table.tableHumpName}}(BaseModel):  
 \_\_tablename\_\_ = "{{table.tableName}}"  
 \_\_table\_args\_\_ = ({"comment": "{{table.comment}}"}){% for column in table.columns %}{%if column.name not in ['id','create\_time','update\_time','is\_deleted']%}  
 {{column.name}} = db.Column(db.{{column.type}}({% if column.length is not none%}{{column.length}}{%endif%}), unique={% if column.unique is not none%}True{% else %}False{%endif%}, nullable={% if column.nullable is not none%}True{% else %}False{%endif%}, comment="{{column.comment}}"){%endif%}{% endfor %}

## 执行命令生成代码

flask code gen -t t\_role



生成效果如下：



# 最后

至此，基于数据库表生成模型结束，目前这里只创建一个代码生成模板，后面我们再制作validators/services/controller层模板。