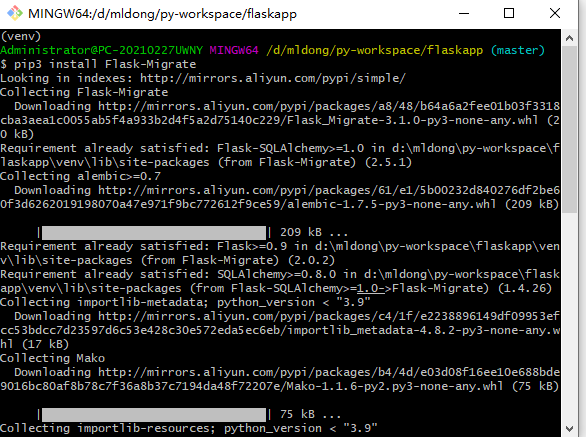
**12.2代码生成·Flask的Code First**

这里的Code就是我们ORM模型，Code First优先就是我们在开发时候，优先设计和编写ORM的模型类。然后再使用一些工具，将我们的ORM模型映射到数据库中。在这里简单的演示该过程。

# 安装Flask-Migrate

Flask-Migrate可以十分方便的进行数据库的迁移与映射，将我们修改过的ORM模型映射到数据库中。

# 安装最新版本  
pip3 install Flask-Migrate  
# 或指定安装版本  
pip3 install Flask-Migrate==3.1.0



# 新增models/role.py

这里新增一个角色模型，映射的表名称为t\_role

from models import db, BaseModel  
  
  
class Role(BaseModel):  
 \_\_tablename\_\_ = "t\_role"  
 \_\_table\_args\_\_ = ({"comment": "角色"})  
 id = db.Column(db.BigInteger, primary\_key=True, comment="主键")  
 name = db.Column(db.String(32), unique=False, nullable=False, comment="角色名称")  
 remark = db.Column(db.String(100), unique=False, nullable=True, comment="备注")

# 新建services/role\_service.py

因为Flask-Migrate迁移命令要求模型在视图层有导入动作，才能完成迁移，所以这里就完整的新建role\_service.py和role\_controller.py文件

from services import BaseService  
  
  
class RoleService(BaseService):  
 """  
 角色模块业务处理类  
 """

# 修改controllers/role\_controller.py

from flask import Blueprint  
  
from controllers import R  
from models.role import Role  
from services.role\_service import RoleService  
  
role = Blueprint('role', \_\_name\_\_, url\_prefix="/role")  
  
role\_service = RoleService(model=Role)  
  
  
@role.route("/get")  
def role\_get():  
 return R.success()  
  
  
@role.route("/list", methods=['POST'])  
def role\_list():  
 return R.success()  
  
  
@role.route("/save", methods=['POST'])  
def role\_save():  
 return R.success()  
  
  
@role.route("/update", methods=['POST'])  
def role\_update():  
 return R.success()  
  
  
@role.route("/delete", methods=['POST'])  
def role\_delete():  
 return R.success()

# 修改app.py

注册Migrate实例

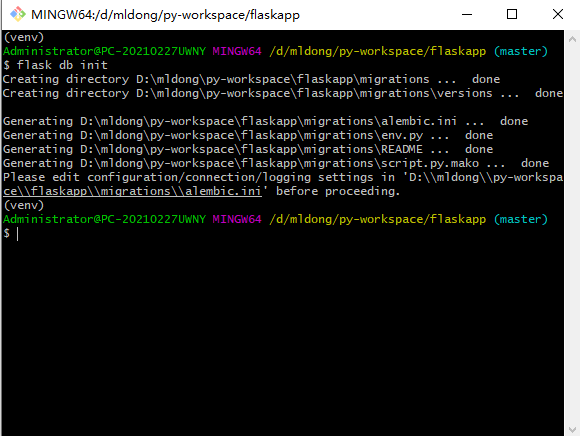
from flask import Flask  
from flask\_migrate import Migrate  
  
from config import getConfig  
from controllers.role\_controller import role  
from controllers.user\_controller import user  
from mauth.mtoken import redis\_client  
from mexception import ExceptionConfig  
from mlogging import LoggingConfig  
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)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(host="0.0.0.0")

# 执行迁移与映射相关命令

## 初始化迁移存储库

和git仓库类似，只需要初始化一次

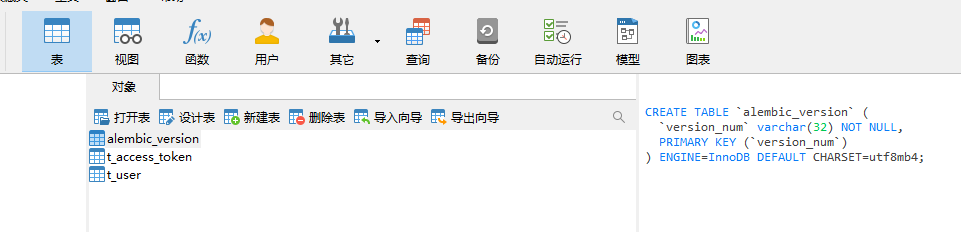
flask db init

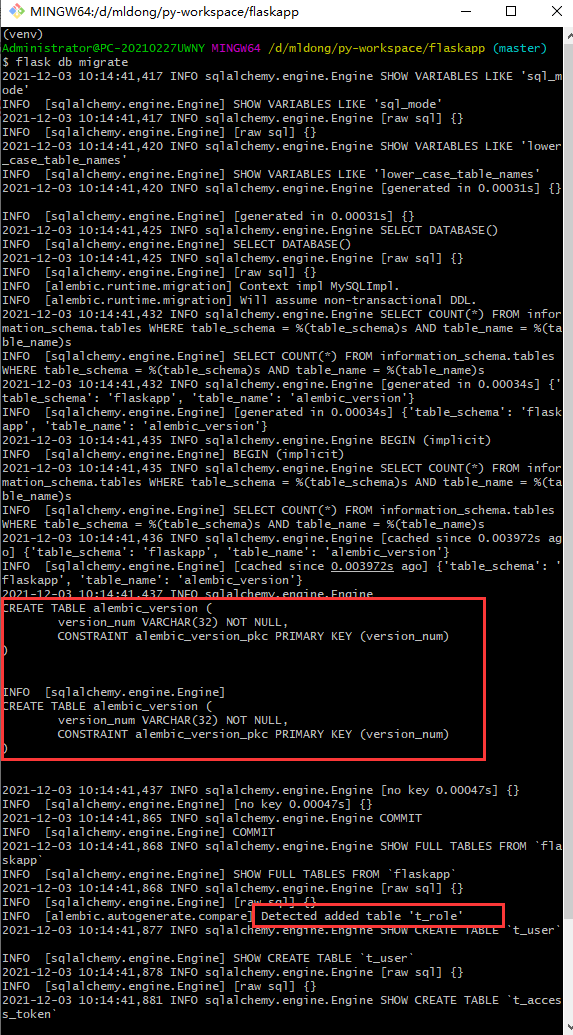


## 生成初始迁移文件

flask db migrate

执行该命令只是生成迁移文件，并未真正进行迁移，其中迁移文件包括：

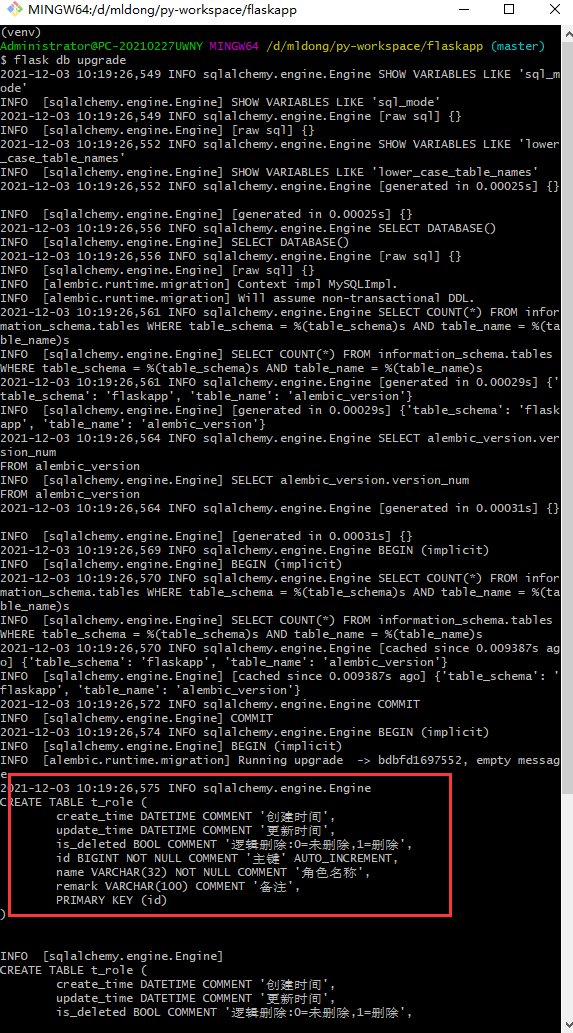


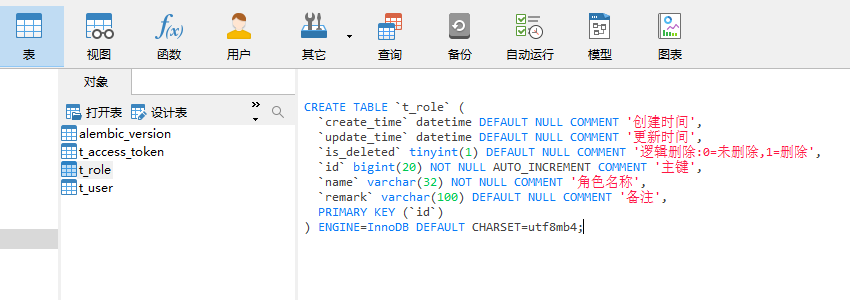


## 同步模型表结构到数据库

生成表t\_role

flask db upgrade





# 修改models/role.py

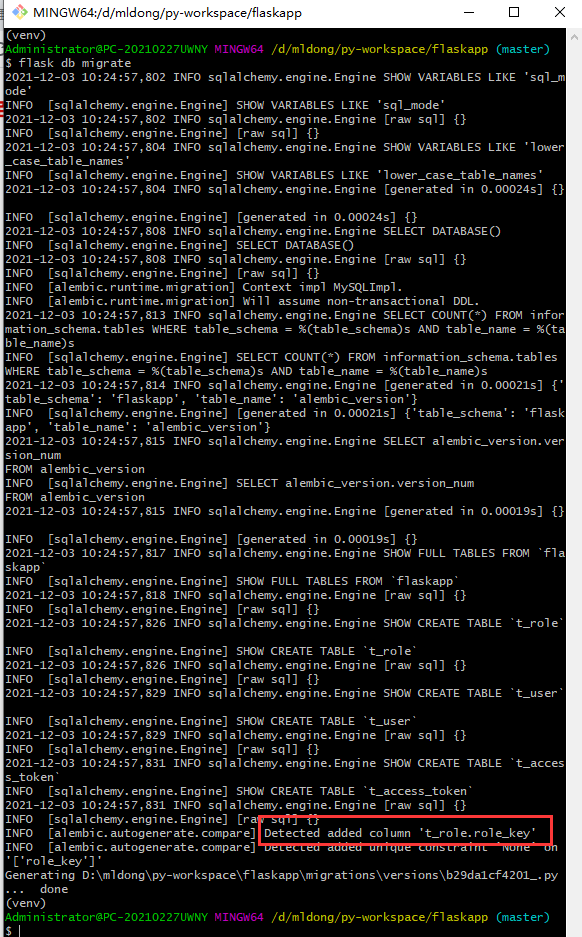
新增role\_key字段

from models import db, BaseModel  
  
  
class Role(BaseModel):  
 \_\_tablename\_\_ = "t\_role"  
 \_\_table\_args\_\_ = ({"comment": "角色"})  
 id = db.Column(db.BigInteger, primary\_key=True, comment="主键")  
 name = db.Column(db.String(32), unique=False, nullable=False, comment="角色名称")  
 role\_key = db.Column(db.String(32), unique=True, nullable=False, comment="角色标识")  
 remark = db.Column(db.String(100), unique=False, nullable=True, comment="备注")

# 再次执行迁移与映射相关命令

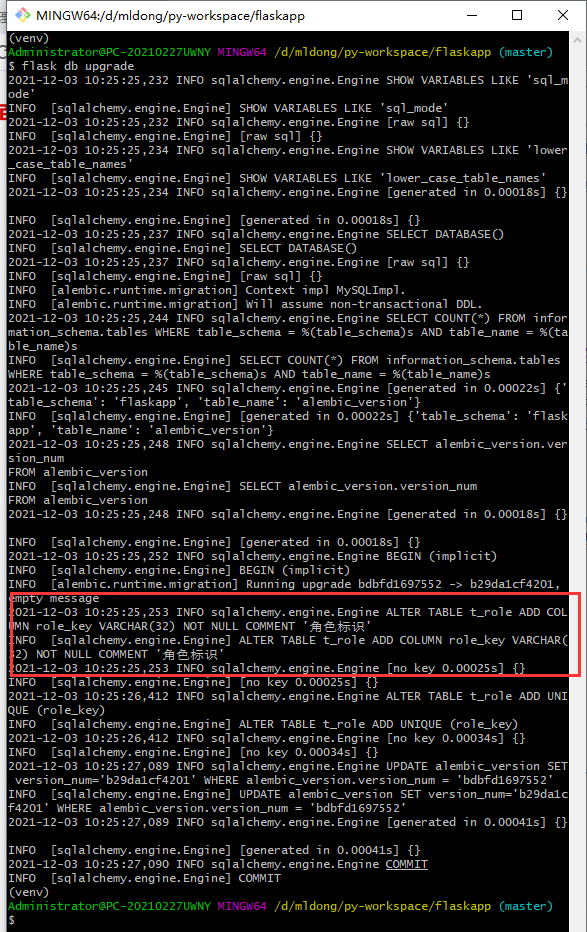
## 生成初始迁移文件

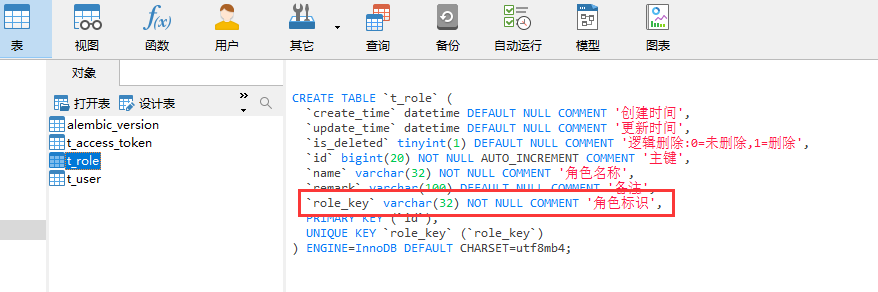
flask db migrate



## 同步模型表结构到数据库

flask db upgrade





# 最后

这就是Code First的开发体验，在我们完成代码编写后，我们可以使用Flask-Migrate相关命令，将我们的模型实体关联的表结构同步到数据库上，这样其实就是省去一些人工建表的环节，而且也不用过多地去关注数据库层的实现，比如数据类型等。不同的数据库有不同的实现，我们在这里可以只使用SQLAlchemy提供的数据类型，由其帮我们进行映射转换即可。