**Flask-SQLAlchemy的使用\_对SQLAlchemy进行了封装和优化：**

Flask-SQLAlchemy是一个插件，

Flask-SQLAlchemy是对SQLAlchemy进行了一个简单的封装的一个插件，

使得我们在flask中使用sqlalchemy更加的简单。

**1.安装：**

pip install flask-sqlalchemy

**2.Flask-SQLAlchemy的使用要点：**

**2.1 数据库连接**

数据库初始化不再是通过create\_engine。

**1.** 跟sqlalchemy一样，定义好数据库连接字符串DB\_URI。

**2.** 将这个定义好的数据库连接字符串DB\_URI，通过`SQLALCHEMY\_DATABASE\_URI`这个key名配置到`app.config`中。

代码：app.config["SQLALCHEMY\_DATABASE\_URI"] = DB\_URI

**3.** 使用`flask\_sqlalchemy.SQLAlchemy`这个类定义一个对象，并将`app`传入进去。

代码：db = SQLAlchemy(app)

**2.2 创建ORM模型类**

之前都是通过Base = declarative\_base()来初始化一个基类，然后再继承，在Flask-SQLAlchemy中更加简单了。

**1.** 还是跟使用sqlalchemy一样，定义模型。现在不再是需要使用`delarative\_base`来创建一个基类。而是使用`db.Model`来作为基类。

**2.** 在模型类中，`Column`、`String`、`Integer`以及`relationship`等，都不需要导入了，直接使用`db`下面相应的属性名就可以了。

**3.** 在定义模型的时候，可以不写`\_\_tablename\_\_`，那么`flask\_sqlalchemy`会默认使用当前的模型的名字转换成小写来作为表的名字，

并且如果这个模型的名字用到了多个单词并且使用了驼峰命名法，那么会在多个单词之间使用下划线来进行连接，

虽然flask\_sqlalchemy给我们提供了这个特性，但是不推荐使用。（增强代码可读性，提高团队合作效率）

**2.3 将ORM模型映射到数据库表**

写完模型类后，要将模型映射到数据库的表中，使用以下代码即可

**1.** 删除数据库表：db.drop\_all()

**2.** 创建数据库表：db.create\_all()

**2.4 session的使用**

以后session也不需要使用`sessionmaker`来创建了，

直接使用`db.session`就可以了，

操作这个session的时候就跟之前的`sqlalchemy`的`session`是一样一样的。

**2.5 添加数据**

这时候就可以在数据库中看到已经生成了对应表了

添加数据和之前的没有区别，只是session成为了一个db的属性

**2.6 查询数据：**

**1.**单表查询

查询数据不再是之前的session.query方法了，而是将query属性放在了db.Model上，

所以查询就是通过“模型名.query”的方式进行查询了，`query`就跟之前的sqlalchemy中的query方法是一样用的。

**2.**多表查询

如果查找数据涉及多个模型，只能使用db.session.query(模型名).all() 这种方式

**2.7 修改数据：**

修改数据和之前的没有区别，只是session成为了一个db的属性

**2.8 删除数据：**

删除数据跟添加数据和修改数据类似，只不过session是db的一个属性而已

**3.代码演示:**

|  |
| --- |
| **from** flask **import** Flask  **from** flask\_sqlalchemy **import** SQLAlchemy  app = Flask(\_\_name\_\_)  HOSTNAME = **'127.0.0.1'**  PORT = **'3306'**  DATABASE = **'first\_sqlalchemy'**  USERNAME = **'root'**  PASSWORD = **'root'**  DB\_URI =**"mysql+pymysql://{username}:{password}@{host}:{port}/{db}?charset=utf8"**.format(username=USERNAME,password=PASSWORD,host=HOSTNAME,port=PORT,db=DATABASE)  app.config[**'SQLALCHEMY\_DATABASE\_URI'**] = DB\_URI  app.config[**'SQLALCHEMY\_TRACK\_MODIFICATIONS'**] = **False**  #1.连接数据库  db = SQLAlchemy(app)  #2.创建ORM模型  **class** User(db.Model):  \_\_tablename\_\_ = **'user'**  id = db.Column(db.Integer,primary\_key=**True**,autoincrement=**True**)  uname = db.Column(db.String(50),nullable=**False**)  **def** \_\_repr\_\_(self):  **return "<User(uname: %s)>"** % self.uname  **class** Article(db.Model):  \_\_tablename\_\_ = **'article'**  id = db.Column(db.Integer,primary\_key=**True**,autoincrement=**True**)  title = db.Column(db.String(50),nullable=**False**)  uid = db.Column(db.Integer,db.ForeignKey(**"user.id"**))  author = db.relationship(**"User"**,backref=**"articles"**)  #3.删除表  db.drop\_all()  #4.创建表  db.create\_all()  #5.添加数据  user = User(uname=**'莫莫'**)  article = Article(title=**'华为5G 算法突破了，俄罗斯小伙突破的'**)  article.author = user  db.session.add(article)  db.session.commit()  #6.查询数据  # users = User.query.all() #等价于 db.session.query(User).all()  # print(users)  #在query属性之后 可以用 order\_by 、 filter、filter\_by、group\_by、having等方法进行更复杂的单表查询  #若要进行更复杂的多表查询，只能使用db.session.query(User).all() 这种方式  #如 order\_by  users = User.query.order\_by(User.id.desc()).all()  print(users)  #7.修改数据  user = User.query.filter(User.uname==**'露露'**).first()  user.uname = **'探探'**  db.session.commit()  #8.删除数据  user = User.query.filter(User.uname==**'探探'**).first()  db.session.delete(user)  db.session.commit()  @app.route(**'/'**)  **def** hello\_world():  **return 'Hello World!'**  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  app.run() |