1.字段类型

SQLAlchemy 创建模型类的时候，跟mysql使用sql语句去创建表很流程很相似。

1、id主键列需要自己定义
2、需要指定字段类型
3、需要指定约束条件
4、表关系

2.常用的SQLAlchemy字段类型

类型名	python中类型	说明
Integer	int	普通整数，一般是32位
SmallIntegeger	int	小整数
BigInteger	int或long	大整数
Float	float	浮点数

类型名	python中类型	说明
Numeric	decimal.Decimal	普通整数，一般是32位
String	str	变长字符串
Text	str	变长字符串，对较长或不限长度的字符串做了优化
Unicode	unicode	变长Unicode字符串
UnicodeText	unicode	变长Unicode字符串，对较长或不限长度的字符串做了优化
Boolean	bool	布尔值
Date	datetime.date	时间
Time	datetime.datetime	日期和时间
LargeBinary	str	二进制文件

3.约束条件

约束	说明

约束	说明
primary_key	主键
unique	唯一约束，True不允许重复
index	索引，如果为True，为这列创建索引，提高查询效率
nullable	空值，如果为True，允许有空值，如果为False，不允许有空值
default	默认值

4.关系

关系	说明
One To Many (一对多)	表示一对多的关系时，在子表类中通过 foreign key (外键)引用父表类，在父表类中通过relationship() 方法来引用子表的类

关系	说明
One To One (一对一)	一对一是两张表之间本质上的双向关系。只需要在一对多关系基础上的父表中使用uselist=False 参数来表示
Many To Many (多对多)	多对多关系会在两个类之间增加一个关联的表,连个关系表中使用 relationship() 方法中通过secondary 来引用关联表，关联表通过 MetaData 对象来与声明基类关联，使用ForeignKey连接接来定位到远程的表

选项名	说明
backref	在关系的另一个模型中添加反向引用
primaryjoin	明确指定两个模型之间使用的联结条件。只在模棱两可的关系中需要指定.
lazy	指定如何加载相关记录。可选值如下：
	select（首次访问时按需加载）
	immediate（源对象加载后就加载）
	joined（加载记录，但使用联结）
	subquery（立即加载，但使用子查询）
	noload（永不加载）
	dynamic（不加载记录，但提供加载记录的查询）
uselist	如果设为 Fales，不使用列表，而使用标量值
order_by	指定关系中记录的排序方式
secondary	指定多对多关系中关系表的名字
secondaryjoin	SQLAlchemy 无法自行决定时，指定多对多关系中的二级联结条件

5.创建模型类

一对多关系：

```
class Father(db.Model):
    """创建一个父亲模型类"""
    # 表名
    __tablename__ = 'father'
    # id主键列，整数类型，自增
    id = db.Column(db.Integer,primary_key=True)
    # name，可变长字符串类型
```

```
    name = db.Column(db.String(20))
    # 关系字段，不是数据库中真实存在的字段，而是为了方便查询添加的
属性
    son = db.relationship('Son', backref='father')


class Son(db.Model):
    # 表名
    __tablename__ = 'son'
    # id主键列，整数类型，自增
    id = db.Column(db.Integer,primary_key=True)
    # name，可变长字符串类型
    name = db.Column(db.String(20))
    # 外键，需要指定字段类型，以及关联那个表的那个字段
    father_id = db.Column(db.Integer,
db.ForeignKey('father.id'))
```

一对一关系：

```
class Father(db.Model):
    """创建一个父亲模型类"""
    # 表名
    __tablename__ = 'father'
    # id主键列，整数类型，自增
    id = db.Column(db.Integer,primary_key=True)
    # name，可变长字符串类型
    name = db.Column(db.String(20))
    # 关系字段，不是数据库中真实存在的字段，而是为了方便查询添加的
属性
    son = db.relationship('Son',
backref='father',uselist=False)


class Son(db.Model):
    # 表名
    __tablename__ = 'son'
    # id主键列，整数类型，自增
    id = db.Column(db.Integer,primary_key=True)
```

```
    # name，可变长字符串类型
    name = db.Column(db.String(20))
    # 外键，需要指定字段类型，以及关联那个表的那个字段
    father_id = db.Column(db.Integer,
db.ForeignKey('father.id'))
```

多对多关系：

1.有关backref和back_populates的作用，都是在一对多或多对多查询的时候。设置一个值，这个值用来从一个表格对象指向到另一个对象，只是用法稍微有点差别。
2.backref是老式的方法，back_populates是新式的方法，但是目前都可以使用

```
# 多对多关系中的两个表之间的一个关联表
association_table = db.Table('association', db.metadata,
    db.Column('author_id', db.Integer,
db.ForeignKey('author.id')),
    db.Column('book_id', db.Integer, db.ForeignKey('book.id'))
)


class Author(db.Model):
    """ 创建作者模型类"""
    # 指定表名，如果没有指定将默认使用模型类的名称
    __tablename__ = 'author'
    # id 类型是整数 主键列
    id = db.Column(db.Integer, primary_key=True)
    # name 类型是可变长字符串，唯一
    name = db.Column(db.String(20), unique=True)
    # 关联中间表，不是数据库中真实存在的字段，而是为了方便查询添加
的属性
    book = db.relationship('Book', back_populates='author',
secondary=association_table)


class Book(db.Model):
    """创建书模型类"""
    # 指定表名，如果不指定将使用模型类名称作为表名
    __tablename__ = 'book'
    id = db.Column(db.Integer,primary_key=True)
```

```
    name = db.Column(db.String(20))
    # 关联中间表，不是数据库中真实存在的字段，而是为了方便查询添加的属性
    author = db.relationship("Author", secondary=association_table, back_populates="book")
```