小白学Flask第十一天| flask-sqlalchemy数据库扩展包(一)

原创JAP君 Python进击者
2019-10-02原文

主要内容：

1. 数据库的设置
2. 定义模型
3. 关系

数据库的设置

学习过web开发的人也许都知道，在web开发中最常用的数据库就是关系模型数据库，关系型数据库把所有的数据都存储在表中，表用来给应用的实体建模，表的列数是固定的，行数是可变的。查询的语句也是结构化的语言。

关系型数据库的列定义了表中表示的实体的数据属性。比如：商品表里有name、price、number等。

Flask本身不限定数据库的选择，你可以选择SQL或NOSQL的任何一种。也可以选择更方便的SQLALchemy，类似于Django的ORM。SQLALchemy实际上是对数据库的抽象，让开发者不用直接和SQL语句打交道，而是通过Python对象来操作数据库，在舍弃一些性能开销的同时，换来的是开发效率的较大提升。

说类这么多，我们今天的主角就是SQLAlchemy。SQLAlchemy是一个关系型数据库框架，它提供了高层的ORM和底层的原生数据库的操作。flask-sqlalchemy是一个简化了SQLAlchemy操作的flask扩展。

前面做了很多铺垫，那么直接进入到今天的主题。

首先关于数据库的安装，我相信在这里不必多说，这里使用的是mysql数据库，如何安装？请大家自行百度。

在前面我也提到了flask-sqlalchemy这个扩展。首先第一步就是去安装这个扩展：

```
pip install flask-sqlalchemy
```

简单粗暴，直接pip一下就ok了。

但是，除了这一个当然是不够的，因为我们需要链接到mysql数据库，所以还得安装下面的库：

```
pip install flask-mysqldb
```

使用Flask-SQLAlchemy扩展操作数据库，首先需要建立数据库连接。数据库连接通过URL指定，而且程序使用的数据库必须保存到Flask配置对象的SQLALCHEMY_DATABASE_URI键中，就例如下面这样：

```
app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql://root:mysql@127.0.0.1:3306/test3'
```

说到这里，我们来对比一下在django是如何配置数据库：

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'test2',
        'USER': 'root',
        'PASSWORD': 'mysql',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

可以看到两者是完全不相同的。

关于配置，这里给出详细一点的代码：

```
#设置连接数据库的URL
app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql://root:mysql@127.0.0.1:3306/Flask_test'


#设置每次请求结束后会自动提交数据库中的改动
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = True


#查询时会显示原始SQL语句
app.config['SQLALCHEMY_ECHO'] = True
db = SQLAlchemy(app)
```

有关于SQLALCHEMY_TRACK_MODIFICATIONS键，flask-sqlachemy官方文档建议设置为Flase，以便于在不需要跟踪对象变化时降低内存消耗。

定义模型

模型是表示应用使用的持久化实体，在ORM中，模型一般是一个Python类，类中的属性就是数据库表中的列。

在这里我们来创建两个模型，分别是Role和User

```
class Role(db.Model):
    # 定义表名
    __tablename__ = 'roles'
    # 定义列对象
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=True)

    #repr()方法显示一个可读字符串
    def __repr__(self):
        return 'Role:%s'% self.name

class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
```

```
    name = db.Column(db.String(64), unique=True, index=True)
    email = db.Column(db.String(64),unique=True)
    pswd = db.Column(db.String(64))

    def __repr__(self):
        return 'User:%s'%self.name
```

接触过数据库的朋友应该大体能看懂是什么意思。

__tablename__代表着数据库表的名称

下面的代码就是创建来一个整型的列id，以及一个字符串类的列name，并且id设置为主键

```
    # 定义列对象
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=True)
```

db.Column类构造函数的第一个参数是数据库列和模型属性的类型。这里为给大家准备了一份常用的SQLAlchemy列类型：

类型名	python中类型	说明
Integer	int	普通整数，一般是32位
SmallInteger	int	取值范围小的整数，一般是16位
BigInteger	int或long	不限制精度的整数
Float	float	浮点数
Numeric	decimal.Decimal	普通整数，一般是32位

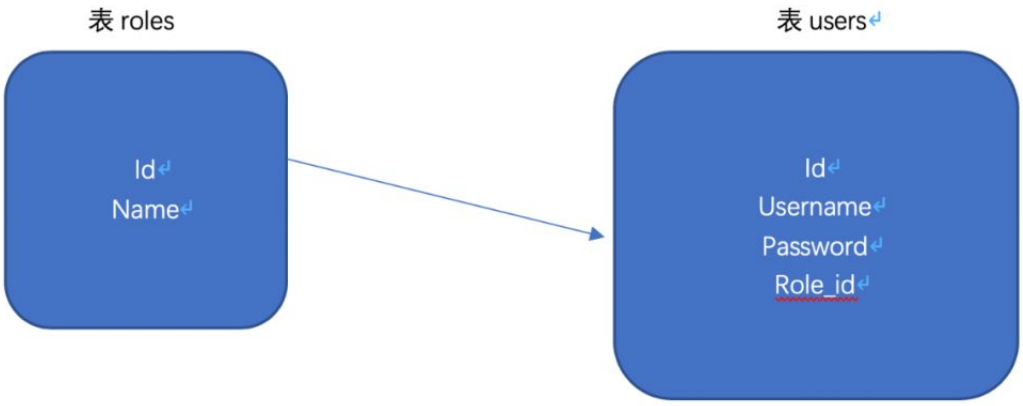
类型名	python中类型	说明
String	str	变长字符串
Text	str	变长字符串，对较长或不限长度的字符串做了优化
Unicode	unicode	变长Unicode字符串
UnicodeText	unicode	变长Unicode字符串，对较长或不限长度的字符串做了优化
Boolean	bool	布尔值
Date	datetime.date	时间
Time	datetime.datetime	日期和时间
LargeBinary	str	二进制文件

同时也给出SQLAlchemy常用的列选项：

选项名	说明
primary_key	如果为True，代表表的主键
unique	如果为True，代表这列不允许出现重复的值
index	如果为True，为这列创建索引，提高查询效率
nullable	如果为True，允许有空值，如果为False，不允许有空值
default	为这列定义默认值

关系

关系型数据库当然得说说关系这个词，关系型数据库就是使用关系把不同表中的行联系在一起。



上图就是一个一对多的关系。

那么如何通过代码来实现这种关系呢？

```
class Role(db.Model):

    #...

    users = db.relationship('User', backref='role')


class User(db.Model):

    # ...

    role_id = db.Column(db.Integer, db.ForeignKey('role_id'))
```

添加到user模型中的role_id列被定义成外键，就是这个外键建立起列关系。传给db.ForeignKey()的参数'role.id'表明，这列的值是roles表中的相应行的id值。

从“一”那一端可知，添加到Role模型中的users属性代表这个关系的面向对象吃的视角。对于一个Role实例，其users属性将返回和角色相关联的用户组成的列表（也就是“多”那一端）。

db.relationship()的第一个参数表明这个关系的另一端是哪个模型。backref参数向User模型中添加一个role属性，从而定义反向关系。通过User实例的这个属性可以获得对应的Role模型对象，而不用再通过role_id外键获取。

这里给出常用的SQLAlchemy关系选项：

选项名	说明
backref	在关系的另一模型中添加反向引用
primary join	明确指定两个模型之间使用的联结条件
uselist	如果为False，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	指定多对多中记录的排序方式
secondary join	在SQLAlchemy中无法自行决定时，指定多对多关系中的二级联结条件

Flask系列文章：

小白学Flask第八天| Flask上下文和请求钩子

小白学Flask第九天| 看看模板的那些事（一）

小白学Flask第十天| 宏、继承、包含、特殊变量

持续更新中...

精选留言

暂无...