头条项目仓库地址：使用git clone

`http://git.meiduo.site/sy_python28/toutiao-backend.git`

1、数据库设计

- 需求：后台开发人员需要写接口，**写接口本质是增删改查**，需要有数据库，表的设计和定义。
  - **根据项目产品原型图**，分析项目数据库和表的数据存储。
  - 数据存储：关系型数据库的数据存储，**mysql中存储的数据，长期存储的数据**。
- 如何设计？如下四个角度：
  - **表结构**
  - **字段类型、是否允许为null、是否有默认值**
  - **索引设计**
  - **数据库引擎的选择**
- 根据黑马头条**前台**产品原型图中**用户端**的部分，进行数据库设计。

2、头条项目前台产品原型图

- 使用工具分析数据库表，excel、画图软件等。
- 思想：**词性分析法，名词体现表或字段，动词体现关系。**
- 主键id：唯一、不变、自增，不建议使用业务数据，比如手机号、身份证号码等。
  - 建议-整型值
- 表和表之间的关系：自问自答！！！文章和评论，一对多的关系。
  - 1、一个文章，有几个评论？多个。
  - 2、一个评论，能属于几个文章？一个。
- 文章图片存储：无图、单图、多图，**JSON类型**
  - mysql5.x版本之后，新有json数据类型，多条key/value形式存**不经常变化的数据**，本质是text文本。
  - 不需要再定义一张表，存储文章和图片的关系。
  - 目的：通过文章表的一个字段，保存文章图片，节省一张表。
- 用户关注：只有2张表，自关联多对多。一般的多对多有3张表，用户表、频道表、用户频道表。
  - 一个人可以关注多人
  - 一个人可以被多人关注
- 三范式：数据库设计的范式，发布文章数量、关注数量、粉丝数量、获赞数量都属于冗余字段。
  - 1、表中的字段是不可分割，原子性。
  - 2、满足第一范式的基础上，有主键依赖。
  - 3、满足第二范式的基础上，非主属性之间，没有依赖关系。
- 反范式设计：本质是减少查询。
  - 利用冗余字段，存储数据，空间换时间。

    - 节省数据库查询时间。
- 美多商城项目：订单表 = 订单id + 商品id + 商品数量 + 商品单价 + 商品总价...
- 实名认证：
    - 需要保存的身份证正面、反面、手持；
    - 活体认证；后台会根据摄像头传输的视频流，会至少保存1张图片。
- 索引设计：**提高查询效率的字段，经常出现在where后面**，手机号(index=True)
    - 数据类型比较复杂，不单一。
    - 主键、外键、唯一；
    - key/value形式的数据，提高查询的字段；
    - 外键的作用：
        - 1、表示表关联，
        - 2、帮助维护表中数据的完整性。
        - 3、数据量大了以后，影响数据的修改、插入、删除的效率；
        - 4、项目初期会用，中后期随着数据量的增加，建议移除外键；
    - 头条项目中没有外键；
- 整型值大小问题：
    - 示例：创建表时，字段的数据类型 = int(10) , int(20) ,int(5)
    - 大小是一样的。
    - int的长度并不影响数据的存储精度，长度只和显示有关
- char和varchar如何选择？示例：手机号和昵称
    - 1、手机号用char还是varchar？在于查询效率，select *** from user where mobile=13012345678，char比varchar查询效率高；
    - 2、昵称用varchar，可变长，不知道昵称有多大。
    - 总结：char查询效率高、但是，浪费空间，varchar节省空间，查询效率低。
- 引擎：核心、发动机、驱动，决定数据存储和数据操作的方式。
    - 1、InnoDB默认的，支持事务、外键，查询效率相对较低。
    - 2、MyISAM，不支持事务，全文索引，查询效率相对较高。
    - 3、引擎是针对表，不同的表，可以有不同的引擎。

3ORM

- 概念：对象关系映射，object----relation-----mapping
    - 定义模型类、属性

```
类-----表
属性----字段
对象----数据(记录)
class Person():
  pass

p1 = Person()
p2 = Person()
```

- 特点：
  - 1、提高开发效率，
  - 2、避免sql注入，
  - 3、兼容不同的数据库，
  - 4、降低查询速度，
- 使用方式：
  - 1、Django中，先定义模型类，数据库迁移，生成表，操作数据；
  - 2、头条项目中，先create table创建表，再定义模型类，操作数据；
  - **头条项目采用编写原生SQL创建表，之后再编写模型类进行映射的方式。**

```
# Flask是支持数据库迁移，需要安装扩展包
flask-script:提供脚本命令，
flask-migrate：提供迁移命令，init/migrate/upgrade/downgrade
```

4.flask-sqlalchemy扩展包

- alchemy：炼金术，orm本身把对象的操作，转成sql让数据库执行。
- 安装：pip install flask-sqlalchemy
- 使用：

```
# django中是通过配置文件settings.py
host/port/user/password/database
# 使用类似于url地址的形式，来连接数据
SQLALCHEMY_DATABASE_URI = 'mysql://用户名:密码@主机和端口/数据库'
# 如果，使用本机数据库
SQLALCHEMY_DATABASE_URI = 'mysql://root:mysql@127.0.0.1:3306/toutiao'
# 简写方式
SQLALCHEMY_DATABASE_URI = 'mysql://root:mysql@localhost/toutiao'

# 代码执行过程中，会展示sql语句
# 生产环境下，不能配置
SQLALCHEMY_ECHO = True
# 动态追踪修改，如果不配置，会提示警告信息，不影响代码的执行，
# 设置True或False都可以关闭警告，True会对计算机性能产生一定的影响，
```

```
# 建议设置False
SQLALCHEMY_TRACK_MODIFICATIONS = False/True
```

- 关系选项：backref表示反向引用

```
人和手机号的关系?
一个人可以有多个手机号!
每个手机号只能属于一个人!
从人查询他的手机号，叫正向查询。
从手机号查询属于谁，叫反向查询。
```

5.定义模型

- 实例化sqlalchemy的两种方式：

```
# 第一种，单个文件的使用方式
app = Flask(__name__)
db = SQLAlchemy(app)

# 第二种，项目使用的方式
db = SQLAlchemy()
# app = Flask(__name__)
def create_app():
  app = Flask(__name__)
  db.init_app(app)
  return app
# 工厂函数一执行，db即和app初始化完成
app = create_app()
```

- 垂直拆分，表的字段太多，把不是经常使用的字段，拆分出去。

6.SQLAlchemy基本操作

- 查询
    - 1、all表示查询所有

```
SQL:
select user_id,mobile,user_name from user_basic;
ORM:
User.query.all()
```

    - 2、first表示查询第一个

```
SQL:
select user_id,mobile,user_name from user_basic limit 1;
ORM:
User.query.first()
```

- 3、get查询，参数为主键

```
SQL:
select user_id,mobile,user_name from user_basic where user_id=1;
ORM:
User.query.get(1)
```

- SQLAlchemy的查询方式

```
db.session.query(User).all()
db.session.query(User).first()
db.session.query(User).get(2)
```

- filter_by表示过滤查询，first/all表示执行器
  - 参数只能使用赋值操作，使用属性名即可

```
u = User.query.filter_by(id=1)
u.first()# 可行
u.mobile # 不行
# <flask_sqlalchemy.BaseQuery object at 0x7fc3681b8a20>
u = User.query.filter_by(id=1).first()
u.mobile # 可行 # <User 1>
User.query.filter_by(id=1).all()# [<User 1>]
User.query.filter_by().all() # 查询所有
```

- filter表示过滤查询，first/all
  - 可以使用丰富的运算符，==,!=
  - 使用类名.属性名

```
User.query.filter(User.id==1)
# 条件可以不写，默认查询所有
# 如果有条件，必须类名.属性名，使用运算符
User.query.filter(User.id==1).all()

# 多个条件
User.query.filter(User.id>=30,User.mobile.startswith('18')).all()
```

    - 逻辑运算符：`not_/and_/or_`需要导入使用from sqlalchemy import or。。。
    - offset和limit，两个函数的先后顺序，不影响执行结果

```
User.query.limit(1)
User.query.offset(1).limit(1).all()
```

- 新增
- 修改
- 删除