《Django 教程》

- 讲师: 魏明择
- 时间: 2019

目录

数据库迁移的错误处理方法

- 当执行 `$ python3 manage.py makemigrations` 出现如下迁移错误时的处理方法
    - 错误信息

```
$ python3 manage.py makemigrations
You are trying to change the nullable field 'title' on book to
non-nullable without a default; we can't do that (the database
needs something to populate existing rows).
Please select a fix:
1) Provide a one-off default now (will be set on all existing
rows with a null value for this column)
2) Ignore for now, and let me handle existing rows with NULL
myself (e.g. because you added a RunPython or RunSQL operation to
handle NULL values in a previous data migration)
3) Quit, and let me add a default in models.py
Select an option:
```

    - 翻译为中文如下:

```
$ python3 manage.py makemigrations
您试图将图书上的可空字段“title”更改为非空字段(没有默认值);我们不能这样做(数
据库需要填充现有行)。
请选择修复:
1)现在提供一次性默认值(将对所有现有行设置此列的空值)
2)暂时忽略，让我自己处理空值的现有行(例如，因为您在以前的数据迁移中添加了
RunPython或RunSQL操作来处理空值)
3)退出，让我在models.py中添加一个默认值
选择一个选项:
```

    - 错误原因
        - 当将如下代码

```
class Book(models.Model):
    title = models.CharField("书名", max_length=50, null=True)
```

        - 去掉 null=True 改为如下内容时会出现上述错误

```
class Book(models.Model):
    title = models.CharField("书名", max_length=50)
```

        - 原理是 此数据库的title 字段由原来的可以为NULL改为非NULL状态,意味着原来这个字段可以不填值，现在改为必须填定一个值，那填什么值呢？此时必须添加一个缺省值。
    - 处理方法:
        1. 选择1 手动给出一个缺省值，在生成 bookstore/migrations/000x_auto_xxxxxxxx_xxxx.py 文件时自动将输入的值添加到default参数中
        2. 暂时忽略，以后用其它的命令处理缺省值问题(不推荐)
        3. 退出当前生成迁移文件的过程，自己去修改models.py, 新增加一个`default=XXX` 的缺省值(推荐使用)

- 数据库的迁移文件混乱的解决办法

    1. 删除 所有 migrations 里所有的 000?_XXXX.py (`__init__.py`除外)
    2. 删除 数据表
        - sql> drop database mywebdb;
    3. 重新创建 数据表
        - sql> create datebase mywebdb default charset...;
    4. 重新生成migrations里所有的 000?_XXXX.py
        - python3 manage.py makemigrations
    5. 重新更新数据库
        - python3 manage.py migrate

数据库的操作(CRUD操作)

- CRUD是指在做计算处理时的增加(Create)、读取查询(Read)、更新(Update)和删除(Delete)

管理器对象

- 每个继承自 models.Model 的模型类，都会有一个 objects 对象被同样继承下来。这个对象叫管理器对象
- 数据换的增删改查可以通过模型的管理器实现

```
class Entry(models.Model):
    ...
Entry.objects.create(...) # 是管理器对象
```

创建数据对象

- Django 使用一种直观的方式把数据库表中的数据表示成Python 对象
- 创建数据中每一条记录就是创建一个数据对象

    1. Entry.objects.create(属性1=值1, 属性2=值1,...)

    - 成功: 返回创建好的实体对象
    - 失败: 抛出异常
2. 创建 Entry 实体对象,并调用 save() 进行保存

```
obj = Entry(属性=值,属性=值)
obj.属性=值
obj.save()
无返回值,保存成功后,obj会被重新赋值
```

3. 示例1:

```
try:
    abook = Book.objects.create(title='Python', pub='清华大学出版
社')
    print(abook)
except:
    print('创建对象失败')
```

4. 示例2:

```
try:
    abook = Book(title='Python', pub='清华大学出版社')
    abook.save
    print(abook)
except:
    print('创建对象失败')
```

5. 示例3:

```
try:
    abook = Book()
    abook.title='Python'
    abook.pub='清华大学出版社'
    abook.save
    print(abook)
except:
    print('创建对象失败')
```

- 练习:
    - 使用以上三种方式,分别向Book和Publisher表中各增加三条数据

Django shell 的使用

- 在Django提供了一个交互式的操作项目叫`Django Shell` 它能够在交互模式用项目工程的代码执行相应的操作
- 利用Django Shell 可以代替编写View的代码来进行直接操作
- 在Django Shell 下只能进行简单的操作，不能运行远程调式
- 启动 Django shell 显示IPython风格的交互界面如下:

```
$ python3 manage.py shell
manage.py shell
Python 3.6.1 (v3.6.1:69c0db5050, Mar 21 2017, 01:21:04)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
In [2]: from bookstore import models
In [3]: models.Book.objects.create(title="Python")
Out[3]: <Book: Book object>
In [4]: book = models.Book.objects.create(title='C++')
In [4]: print(book)
Book object
```

- 练习:

```
在 bookstore/models.py 应用中添加两个model类
1. Book - 图书
    1. title - CharField 书名,非空,唯一
    2. pub - CharField 出版社,字符串,非空
    3. price - 图书价格,,
    4. market_price - 图书零售价
2. Author - 作者
    1. name - CharField 姓名,非空,唯一,加索引
    2. age - IntegerField, 年龄,非空, 缺省值为1
    3. email - EmailField, 邮箱,允许为空
```

  - 然后用Django Shell 添加如下数据
    - 图书信息

书名	定价	零售价	出版社
Python	20.00	25.00	清华大学出版社
Python3	60.00	65.00	清华大学出版社
Django	70.00	75.00	清华大学出版社
JQuery	90.00	85.00	机械工业出版社
Linux	80.00	65.00	机械工业出版社

书名	定价	零售价	出版社
Windows	50.00	35.00	机械工业出版社
HTML5	90.00	105.00	清华大学出版社

- 作者信息:

姓名	年龄	邮箱
王老师	28	wangwc@tedu.cn
吕老师	31	lvze@tedu.cn
祁老师	30	qitx@tedu.cn

查询数据

- 数据库的查询需要使用管理器对象进行
- 通过 Entry.objects 管理器方法调用查询接口

方法	说明
all()	查询全部记录,返回QuerySet查询对象
get()	查询符合条件的单一记录
filter()	查询符合条件的多条记录
exclude()	查询符合条件之外的全部记录

...

1. all()方法
    - 方法: all()
    - 用法: Entry.objects.all()
    - 作用: 查询Entry实体中所有的数据
        - 等同于
            - select * from tabel
    - 返回值: QuerySet容器对象,内部存放Entry 实例
    - 示例:

```
from bookstore import models
books = models.Book.object.all()
for book in books:
    print("书名", book.title, '出版社:', book.pub)
```

2. 在模型类中定义 `def __str__(self):` 方法可以将自定义默认的字符串

```
class Book(models.Model):
    title = ...
    def __str__(self):
```

```
        return "书名: %s, 出版社: %s, 定价: %s" % (self.title, self.pub,
self.price)
```

3. 查询返回指定列(字典表示)

    - 方法: values('列1', '列2')
    - 用法: Entry.objects.values(...)
    - 作用: 查询部分列的数据并返回
        - select 列1,列2 from xxx
    - 返回值: QuerySet
        - 返回查询结果容器，容器内存字典，每个字典代表一条数据,
        - 格式为: {'列1': 值1, '列2': 值2}
    - 示例:

```
from bookstore import models
books = models.Book.objects.values("title", "pub")
for book in books:
    print("书名", book["title"], '出版社:', book['pub'])
    print("book=", book)
```

4. 查询返回指定列（元组表示)

    - 方法:values_list('列1','列2')
    - 用法:Entry.objects.values_list(...)
    - 作用:
        - 返回元组形式的查询结果
    - 返回值: QuerySet容器对象,内部存放 元组
        - 会将查询出来的数据封装到元组中,再封装到查询集合QuerySet中
    - 示例:

```
from bookstore import models
books = models.Book.objects.values_list("title", "pub")
for book in books:
    print("book=", book)  # ('Python', '清华大学出版社')...
```

5. 排序查询

    - 方法:order_by
    - 用法:Entry.objects.order_by('-列','列')
    - 作用:
        - 与all()方法不同，它会用SQL 语句的ORDER BY 子句对查询结果进行根据某个字段选择性的进行排序
    - 说明:

        - 默认是按照升序排序,降序排序则需要在列前增加'-'表示

    - 示例:

```
from bookstore import models
books = models.Book.objects.order_by("price")
for book in books:
    print("书名:", book.title, '价格:', book.price)
```

6. 根据条件查询多条记录

    - 方法: filter(条件)
    - 语法:

```
Entry.objects.filter(属性1=值1, 属性2=值2)
```

    - 返回值:
        - QuerySet容器对象,内部存放 Entry 实例
    - 说明:
        - 当多个属性在一起时为与关系，即当`Books.objects.filter(price=20, pub="清华大学出版社")` 返回价格为20 且 出版社为"清华大学出版社"的全部图书
    - 示例:

```
# 查询书中出版社为"清华大学出版社"的图书
from bookstore import models
books = models.Book.objects.filter(pub="清华大学出版社")
for book in books:
    print("书名:", book.title)

2. 查询Author实体中id为1并且isActive为True的
    - authors=Author.objects.filter(id=1,isActive=True)
```

字段查找

- 字段查询是指如何指定SQL语句中 WHERE 子句的内容。
- 字段查询需要通过QuerySet的filter(), exclude() and get()的关键字参数指定。
- 非等值条件的构建,需要使用字段查询
- 示例:

```
# 查询作者中年龄大于30
Author.objects.filter(age__gt = 30)
# 对应
SELECT .... WHERE AGE > 35;
```

查询谓词

- 每一个查询谓词是一个独立的查询功能

1. `__exact` : 等值匹配

```
Author.objects.filter(id__exact=1)
select * from author where id = 1
```

2. `__contains` : 包含指定值

```
Author.objects.filter(name__contains='w')
select * from author where name like '%w%'
```

3. `__startswith` : 以 XXX 开始
4. `__endswith` : 以 XXX 开始
5. `__gt` : 大于指定值

```
Author.objects.filer(age__gt=50)
select * from author where age > 50
```

6. `__gte` : 大于等于
7. `__lt` : 小于
8. `__lte` : 小于等于
9. `__in` : 查找数据是否在指定范围内
    - 示例

```
Author.objects.filter(country__in=['中国','日本','韩国'])
# 等同于
select * from author where country in ('中国','日本','韩国')
```

10. `__range`: 查找数据是否在指定的区间范围内

```
# 查找年龄在某一区间内的所有作者
Author.objects.filter(age__range=(35,50))
# 等同于
SELECT ... WHERE Author BETWEEN 35 and 50;
```

11. 详细内容参见: https://yiyibooks.cn/xx/Django_1.11.6/ref/models/querysets.html

- 示例

```
Entry.objects.filter(id__gt=4)
SQL等效:

SELECT ... WHERE id > 4;
```

- 练习:

    1. 查询Author表中age大于等于85的信息
        - `Author.objects.filter(age__gte=85)`
    2. 查询Author表中姓王的人的信息
        - `Author.objects.filter(name__startswith='王')`
    3. 查询Author表中Email中包含"wc"的人的信息
        - `Author.objects.filter(email__contains='in')`

2. 不等的条件筛选

    - 语法: Entry.objects.exclude(条件)
    - 作用:
        - 返回不包含此 条件 的 作部的数据集
    - 示例:
        - 查询 清华大学出版社，价格大于50 以外的全部图书

```
books = models.Book.objects.exclude(pub=清华大学出版社,
price__lt=50)
for book in books:
    print(book)
```

3. 查询指定的一条数据

    - 语法: Entry.objects.get(条件)
    - 作用:
        - 返回满足条件的唯一一条数据
    - 返回值:
        - Entry 对象
    -
    - 说明:
        - 该方法只能返回一条数据
        - 查询结果多余一条数据则抛出,Model.MultipleObjectsReturned异常
        - 查询结果如果没有数据则抛出Model.DoesNotExist异常
    - 示例:

```
from bookstore import models
book = models.Book.object.get(id=1)
print(book.title)
```

修改数据记录

1. 修改单个实体的某些字段值
    1. 查
        - 通过 get() 得到要修改的实体对象
    2. 改
        - 通过 对象.属性 的方式修改数据
    3. 保存
        - 通过 对象.save() 保存数据
    - 如:

```
from bookstore import models
abook = models.Book.objects.get(id=10)
abook.market_price = "10.5"
abook.save()
```

2. 通过 QuerySet 批量修改 对应的全部字段
    - 直接调用QuerySet的update(属性=值) 实现批量修改
    - 如:

```
# 将 id大于3的所有图书价格定为0元
books = Book.objects.filter(id__gt=3)
books.update(price=0)
# 将所有书的零售价定为100元
books = Book.objects.all()
books.update(market_price=100)
```

删除记录

- 删除记录是指删除数据库中的一条或多条记录
- 删除单个Entry对象或删除一个查询结果集(QuerySet)中的全部对象都是调用 delete()方法

1. 删除单个对象
    - 步骤
        1. 查找查询结果对应的一个数据对象
        2. 调用这个数据对象的delete()方法实现删除
    - 示例:

```
try:
    auth = Author.objects.get(id=1)
    auth.delete()
except:
    print(删除失败)
```

2. 删除查询结果集
    - 步骤
        1. 查找查询结果集中满足条件的全部QuerySet查询集合对象
        2. 调用查询集合对象的delete()方法实现删除
    - 示例:

```
# 删除全部作者中，年龄大于65的全部信息
auths = Author.objects.filter(age__gt=65)
auths.delete()
```

聚合查询

- 聚合查询是指对一个数据表中的一个字段的数据进行部分或全部进行统计查询,查bookstore_book数据表中的全部书的平均价格，查询所有书的总个数等,都要使用聚合查询

1. 不带分组聚合
    - 不带分组的聚合查询是指导将全部数据进行集中统计查询
    - 聚合函数:
        - 定义模块: `django.db.models`
        - 用法: `from django.db.models import *`
        - 聚合函数:
            - Sum, Avg, Count, Max, Min
    - 语法:
        - Entry.objects.aggregate(结果变量名=聚合函数('列'))
    - 返回结果:
        - 由 结果变量名和值组成的字典
        - 格式为:
            - `{"结果变量名": 值}
    - 示例:

```
# 得到所有书的平均价格
from bookstore import models
from django.db.models import Count
result = models.Book.objects.aggregate(myAvg=Avg('price'))
print("平均价格是:", result['myAvg'])
print("result=", result)  # {"myAvg": 58.2}

# 得到数据表里有多少本书
from django.db.models import Count
result = models.Book.objects.aggregate(mycnt=Count('title'))
print("数据记录总个数是:", result['mycnt'])
print("result=", result)  # {"mycnt": 10}
```

2. 分组聚合

- 分组聚合是指通过计算查询结果中每一个对象所关联的对象集合，从而得出总计值(也可以是平均值或总和)，即为查询集的每一项生成聚合。
- 语法:
  - QuerySet.annotate(结果变量名=聚合函数('列'))
- 用法步骤:
  1. 通过先用查询结果Entry.object.value. 查找查询要分组聚合的列
     - Entry.object.value('列1', '列2')
     - 如:

```
pub_set = models.Book.objects.values('pub')
print(books)  # <QuerySet [{'pub': '清华大学出版社'},
{'pub': '清华大学出版社'}, {'pub_hou {'pub': '机械工业出版
社'}, {'pub': '清华大学出版社'}]>
```

  2. 通过返回结果的 QuerySet.annotate 方法分组聚合得到分组结果
     - QuerySet.annotate(名=聚合函数('列'))
     - 返回 QuerySet 结果集,内部存储结果的字典
     - 如:

```
pub_count_set = pub_set.annotate(myCount=Count('pub'))
print(pub_count_set)  # <QuerySet [{'pub': '清华大学出版
社', 'myCount': 7}, {'pub': '机械工业出版社', 'myCount':
3}]>
```

  - .values('查询列名')
- 示例:
  - 得到哪儿个出版社共出版多少本书

```
def test_annotate(request):
    - from django.db.models import Count
    from . import models

    # 得到所有出版社的查询集合QuerySet
    pub_set = models.Book.objects.values('pub')
    # 根据出版社查询分组，出版社和Count的分组聚合查询集合
    pub_count_set = pub_set.annotate(myCount=Count('pub'))  # 返回
查询集合
    for item in pub_count_set:
        print("出版社:", item['pub'], "图书有：", item['myCount'])
    return HttpResponse('请查看服务器端控制台获取结果')
```

F对象

- 一个F对象代表数据库中某个字段的信息
- F对象通常是对数据库中的字段值在不加载到内存中的情况下直接在数据库服务器端进行操作
- F对象在 数据包 django.db.models 中.使用时需要通过如下语句进行加载
    - `from django.db.models import F`

1. 作用:
    - 在执行过程中获取某列的值并对其直接进行操作
    - 当同时对数据库中两个字段的值进行比较获取 QuerySet 数据集时，可以使用F对象
2. 说明:
    - 一个 F()对象代表了一个model的字段的值
3. 使用它就可以直接参考model的field和执行数据库操作而不用再把它们（model field）查询出来放到python内存中。
4. 语法:

```
from django.db.models import F
F('列名')
```

5. 示例1
    - 更新Book实例中所有的制场价涨10元

```
models.Book.objects.all().update(market_price=F('market_price')+10)
# 以下做法好于如下代码
books = models.Book.objects.all()
for book in books:
    book.update(market_price=book.marget_price+10)
    book.save()
```

6. 示例2
    - 对数据库中两个字段的值进行比较，列出哪儿些书的零售价高于定价?

```
from django.db.models import F
from bookstore import models
books = models.Book.objects.filter(market_price__gt=F('price'))
for book in books:
    print(book.title, '定价:', book.price, '现价:', book.market_price)
```

Q对象 - Q()

- 当在获取查询结果集 使用复杂的逻辑或 | 、 逻辑非 ~ 等操作时可以借助于 Q对象进行操作
- 如: 想找出定价低于20元 或 清华大学出版社的全部书，可以写成

```
models.Book.objects.filter(Q(price__lt=20)|Q(pub="清华大学出版社"))
```

- Q对象在 数据包 django.db.models 中。需要先导入再使用
  - `from django.db.models import F`

1. 作用
   - 在条件中用来实现除 and(&) 以外的 or(|) 或 not(~) 操作
2. 运算符:
   - & 与操作
   - | 或操作
   - ~ 非操作
3. 语法

```
from django.db.models import Q
Q(条件1)|Q(条件2)  # 条件1成立或条件2成立
Q(条件1)&Q(条件2)  # 条件1和条件2同时成立
Q(条件1)&~Q(条件2)  # 条件1成立且条件2不成立
...
```

4. 示例

```
from django.db.models import Q
# 查找清华大学出版社的书或价格低于50的书
models.Book.objects.filter(Q(market_price__lt=50) | Q(pub_house='清华大学出版社'))
# 查找不是机械工业出版社的书且价格低于50的书
models.Book.objects.filter(Q(market_price__lt=50) & ~Q(pub_house='机械工业出版社'))
```

原生的数据库操作方法

使用Entry.objects.raw()进行 数据库查询操作查询

- 在django中，可以使用模型管理器的raw方法来执行select语句进行数据查询

1. 语法:
   - `Entry.objects.raw(sql语句)`
2. 用法
   - `Entry.objects.raw('sql语句')`
3. 返回值:
   - QuerySet 集合对象

4. 示例

```
books = models.Book.objects.raw('select * from bookstore_book')

for book in books:
    print(book)
```

admin 后台数据库管理

- django 提供了比较完善的后台管理数据库的接口，可供开发过程中调用和测试使用
- django 会搜集所有已注册的模型类，为这些模型类提拱数据管理界面，供开发者使用
- 使用步骤:
    1. 创建后台管理帐号:
        - 后台管理--创建管理员帐号
            - `$ python3 manage.py createsuperuser`
            - 根据提示完成注册,参考如下:

```
$ python3 manage.py createsuperuser
Username (leave blank to use 'tarena'): tarena  # 此处输入用户名
Email address: weimz@tedu.cn  # 此处输入邮箱
Password: # 此处输入密码(密码要复杂些，否则会提示密码太简单)
Password (again): # 再次输入重复密码
Superuser created successfully.
$
```

    2. 用注册的帐号登陆后台管理界面
        - 后台管理的登录地址:
            - http://127.0.0.1:8000/admin

自定义后台管理数据表

- 若要自己定义的模型类也能在 `/admin` 后台管理界中显示和管理，需要将自己的类注册到后台管理界面
- 添加自己定义模型类的后台管理数据表的,需要用`admin.site.register(自定义模型类)` 方法进行注册
    - 配置步骤如下:
        1. 在应用app中的admin.py中导入注册要管理的模型models类, 如:

```
from . import models
```

        2. 调用 admin.site.register 方法进行注册,如:

```
from django.contrib import admin
admin.site.register(自定义模型类)
```

- 如: 在 bookstore/admin.py 添加如下代码对Book类进行管理
- 示例:

```
# file: bookstore/admin.py
from django.contrib import admin
# Register your models here.

from . import models
...
admin.site.register(models.Book)  # 将Book类注册为可管理页面
```

修改后台Models的展现形式

- 在admin后台管理数据库中对自定义的数据记录都展示为 `XXXX object` 类型的记录，不便于阅读和判断
- 在用户自定义的模型类中可以重写 `def __str__(self):` 方法解决显示问题,如:
  - 在 自定义模型类中重写 **str**(self) 方法返回显示文字内容:

```
classd Bookstore(models.Model):
    ...
    def __str__(self):
        return "书名" + self.title
```

模型管理器类

- 作用:
  - 用后台管理界面添加便于操作的新功能。
- 说明:
  - 后台管理器类须继承自 `django.contrib.admin` 里的 `ModelAdmin` 类
- 模型管理器的使用方法:

1. 在 `<应用app>/admin.py` 里定义模型管理器类

```
class XXXX_Manager(admin.ModelAdmin):
    ......
```

2. 注册管理器与模型类关联

```
from django.contrib import admin
from . import models
admin.site.register(models.YYYY, XXXX_Manager) # 注册models.YYYY
模型类与 管理器类 XXXX_Manager 关联
```

- 示例:

```
# file : bookstore/admin.py
from django.contrib import admin
from . import models

class BookAdmin(admin.ModelAdmin):
    list_display = ['id', 'title', 'price', 'market_price']

admin.site.register(models.Book, BookAdmin)
```

  - 进入http://127.0.0.1:8000/admin/bookstore/book/ 查看显示方式和以前有所不同

- 模型管理器类ModelAdmin中实现的高级管理功能

1. list_display 去控制哪些字段会显示在Admin 的修改列表页面中。
2. list_display_links 可以控制list_display中的字段是否应该链接到对象的“更改”页面。
3. list_filter 设置激活激活Admin 修改列表页面右侧栏中的过滤器
4. search_fields 设置启用Admin 更改列表页面上的搜索框。
5. list_editable 设置为模型上的字段名称列表，这将允许在更改列表页面上进行编辑。
6. 其它参见https://docs.djangoproject.com/en/1.11/ref/contrib/admin/

数据库表管理

1. 修改模型类字段的显示名字
   - 模型类各字段的第一个参数为verbose_name,此字段显示的名字会在后台数据库管理页面显示
   - 通过 verbose_name 字段选项,修改显示名称示例如下:

```
title = models.CharField(
    max_length = 30,
    verbose_name='显示名称'
)
```

2. 通过Meta内嵌类 定义模型类的属性及展现形式
   - 模型类可以通过定义内部类class Meta 来重新定义当前模型类和数据表的一些属性信息
   - 用法格式如下:

```
class Book(models.Model):
    title = CharField(....)
    class Meta:
```

```
        1. db_table = '数据表名'
            - 该模型所用的数据表的名称。(设置完成后需要立马更新同步数据库)
        2. verbose_name = '单数名'
            - 给模型对象的一个易于理解的名称(单数),用于显示在/admin管理界
面中
        3. verbose_name_plural = '复数名'
            - 该对象复数形式的名称(复数),用于显示在/admin管理界面中
```

- 示例:

```
class Meta:
    db_table = 'book_table'  # 将原数据表名"bookstore_book" 换为
"book_table",请查看数据表
    verbose_name = 'booooook'
    verbose_name_plural = 'booksssssss'  # 去127.0.0.1:8000/admin
下看看哪儿变化了?
```

- 练习:
  - 将Book模型类 和 Author 模型类都加入后台管理
  - 制作一个AuthorManager管理器类，让后台管理Authors列表中显示作者的ID、姓名、年龄信息
  - 用后台管理程序 添加三条 Author 记录
  - 修改其中一条记录的年龄
  - 删除最后一条添加的记录
  - 将bookstore_author 数名表名称改为myauthor (需要重新迁移数据库)