数据库迁移方案研究

[一、 资料 1](#_Toc502076609)

[迁移：是Django用于同步你的发生改变的模型(添加一个字段，删除一个模型，等等。) 到你的数据库。 1](#_Toc502076610)

[命令集： 2](#_Toc502076611)

[注意： 2](#_Toc502076612)

[二、 基础操作 2](#_Toc502076613)

[1、 后台环境支持： 2](#_Toc502076614)

[2、 定义一个虚拟环境，用于进行测试： 2](#_Toc502076615)

[3、 数据库操作： 2](#_Toc502076616)

[4、 django迁移相关基础操作： 2](#_Toc502076617)

[三、 验证操作： 5](#_Toc502076618)

[1、 基于portal项目进行相关验证： 5](#_Toc502076619)

[2、 以t2cloud\_auth项目为例，进行迁移的版本控制： 7](#_Toc502076620)

[3、 清除指定app的迁移文件和迁移的历史，重新创建一个新的迁移文件 8](#_Toc502076621)

[四、 应用软件辅助操作： 9](#_Toc502076622)

[1、pycharm IDE下如何进行对应的迁移的操作和调试 10](#_Toc502076623)

[2、navicat 进行数据库的对应操作 10](#_Toc502076624)

[五、 扩展内容： 10](#_Toc502076625)

[1、 Setting 配置文件 10](#_Toc502076626)

[六、 迁移的过程中添加初始化的数据 11](#_Toc502076627)

[1、 有些时候，只初始化数据的表结构是不能满足需求的，还需要为指定的表添加指定的初始话数据信息： 11](#_Toc502076628)

[七、 进行数据库的实际迁移工作 12](#_Toc502076629)

[1、 目前项目的dashboard 数据库的初始化脚本文件存放在mental项目中。为了能更好的对整个dashboard进行迁移版本的控制，初步决定在mental项目中创建一个db\_migrations文件夹进行dashboard整个项目的迁移文件统一存放，实现数据迁移的版本控制。 12](#_Toc502076630)

[2、 初始化mental下面的脚本存放路径 12](#_Toc502076631)

[3、 手动编写对应的迁移文件脚本，并配置迁移的顺序。将脚本文件统一存放起来方便之后的管理操作。由于不太清楚rfr/2.2.3 和 rfr/2.2.4 两个版本的区别，所以目前先以 mental、portal、mental\_cms、auth这个四个应用进行控制操作。 12](#_Toc502076632)

[八、 引用文献： 14](#_Toc502076633)

1. 资料

迁移：是Django用于同步你的发生改变的模型(添加一个字段，删除一个模型，等等。) 到你的数据库。

命令集：

migrate ：负责应用迁移，以及取消应用并列出其状态。

makemigrations：负责基于你的模型修改创建一个新的迁移。

sqlmigrate：展示迁移的sql语句。

注意：

迁移是建立和运行在每一个app的基础上的。要是迁移正常工作，必须要初始迁移，然后进行更改。（Django会将更改与迁移文件进行比较，而不是数据库）

执行迁移命令的时候，MySQL应该提前将数据库创建好。

MySQl缺少对模式更改操作的事物支持，这意味着如果迁移无法应用，则必须手动取消配准更改才能再次尝试（不可能回滚到更早的点）。

如果数据库中已经存在表，进行初始迁移的时候要使用 python manage.py mrgrate –fake-initial。

1. 基础操作
2. 后台环境支持：

Django 1.8

MySQL

Python 2.7.11+

1. 定义一个虚拟环境，用于进行测试：

# 创建一个虚拟环境

$ mkvirtualvens django\_py2

# 切换到django\_py2环境中

$ workon django\_py2

1. 数据库操作：

# 确保数据库服务已运行

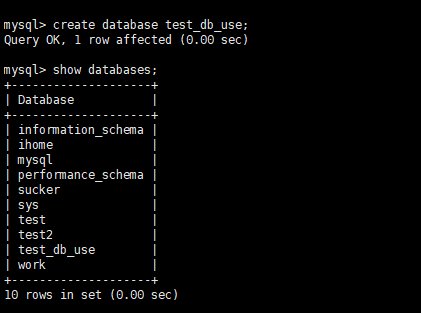
$ systemctl |grep mysql.service

# 进入mysql，并创建测试的数据库test\_db\_use

$ mysql –u root –p

>create databases test\_db\_use

>use test\_db\_use



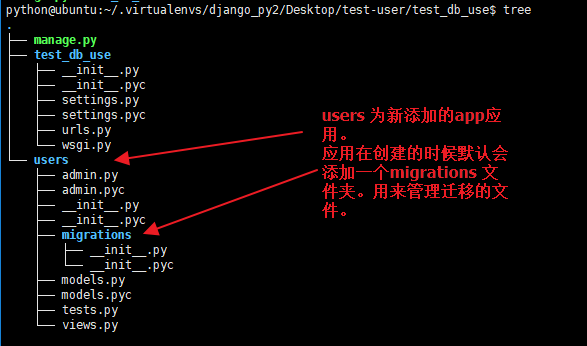
1. django迁移相关基础操作：

# 创建一个测试项目test\_db\_use

$ django-admin startproject test\_db\_use

# 添加一个users的app应用

$ python manage.py startapp users



# 修改users中的models文件，定义模型类

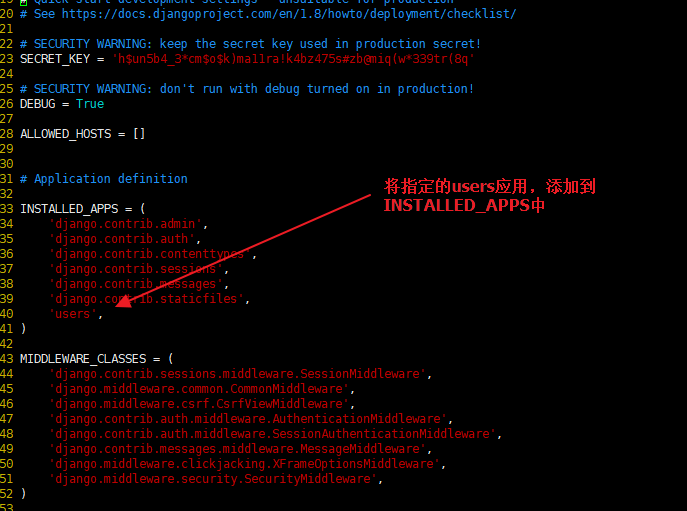
# 模型类对应数据库中的数据表：

$ vim models.py



# 激活模型，编辑setting.py文件，将users应用加入到INSTALLED\_APPS中

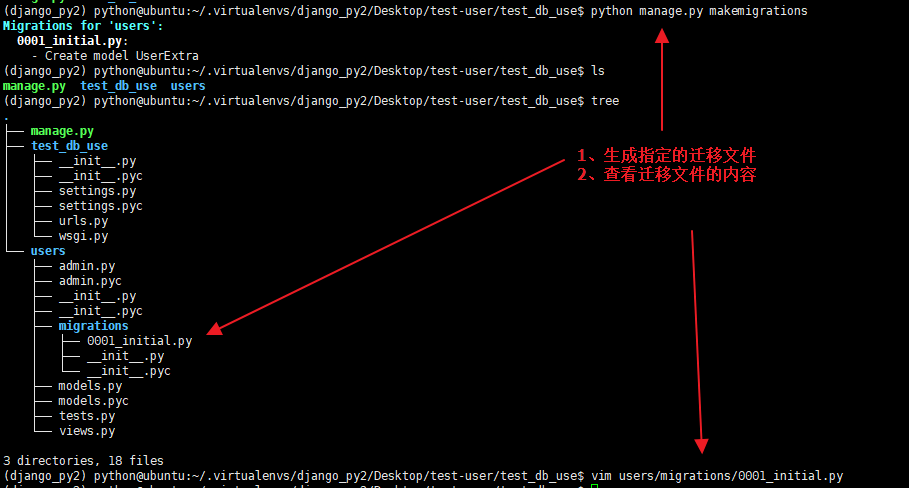
$ vim ../test\_db\_use/setting.py





# 生成迁移文件，根据模型生成sql语句

$ python manage.py makemigrations



# 如果要给migrations起一个有意义的名字而不是用自动生成的，则可以使用--name选项

$ python manage.py makemigrations --name user\_db\_1216

# 如果app应用中不存在migrations文件夹，生成迁移文件时，指定app，会在应用中创建一个migrations文件夹，并在其中生成迁移文件

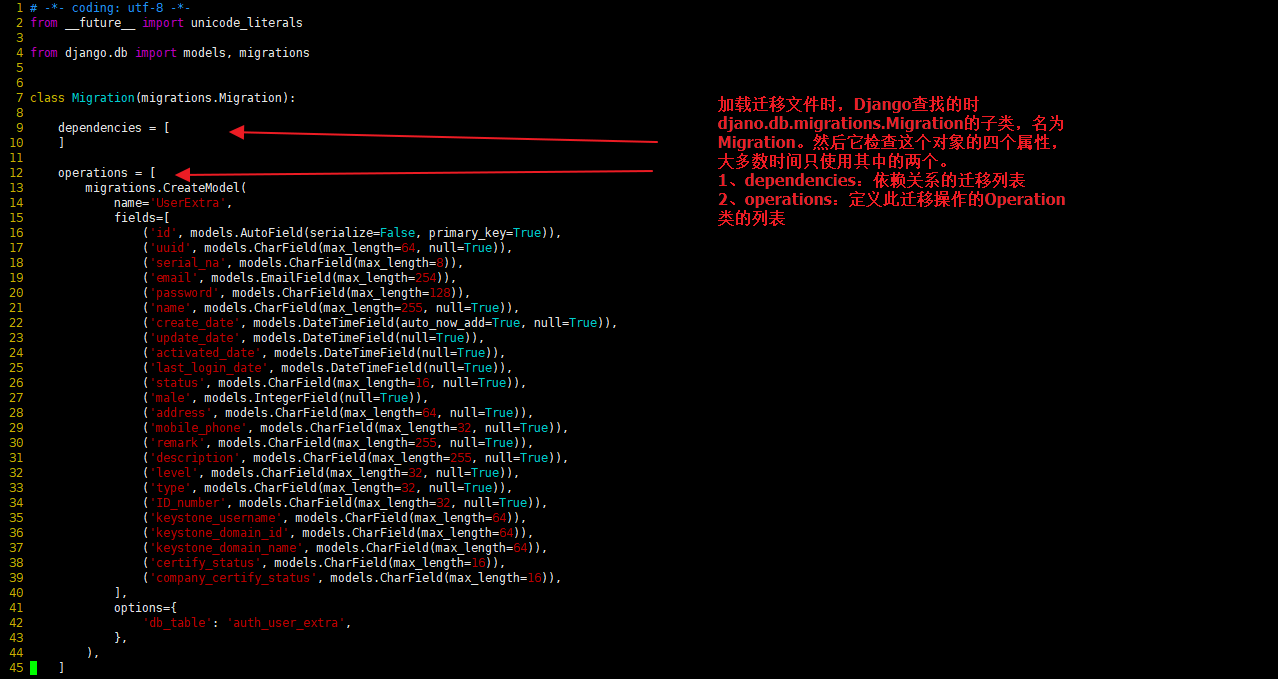
$ python manage.py makemigrations users

# 查看更多相关操作

$ python manage.py --help

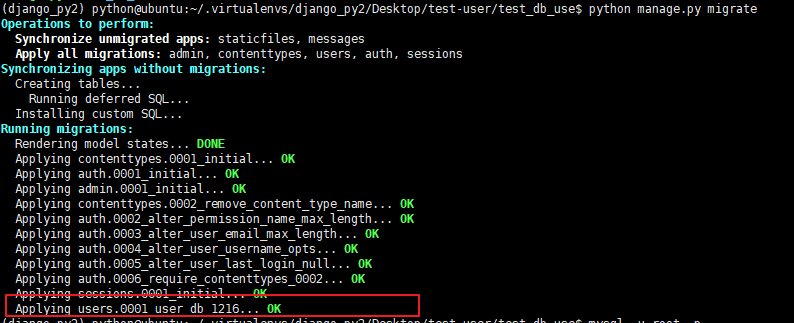
$ python manage.py makemigrations --help

# 查看生成的迁移文件

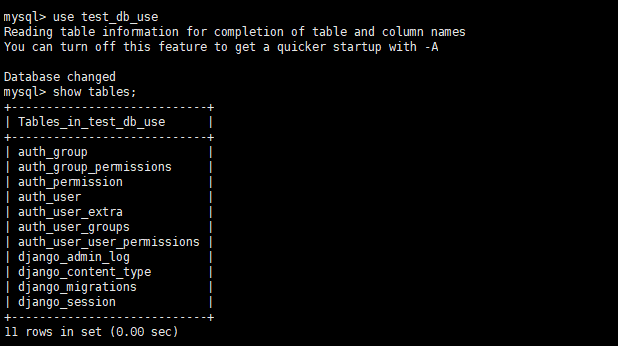


# 执行迁移操作

$ python manage.py



# 进入数据库查看执行结果



# 查看当前项目中，对应的已经生效的migration文件

$ python manage.py showmigrations

1. 验证操作：
2. 基于portal项目进行相关验证：

# 本地mysql数据库中，创建一个dashboard数据库

> create database dashboard;

> use dashboard;

> show tables;

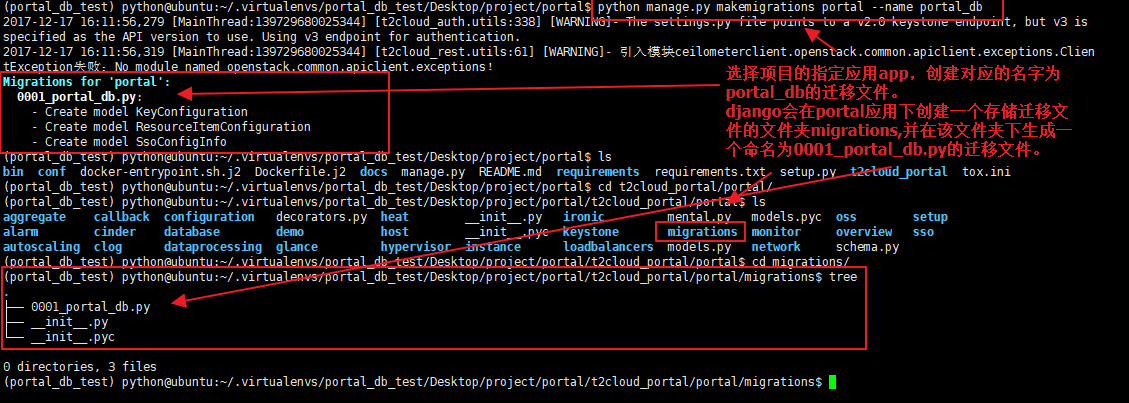
# 通过django项目文件的setting.py设置文件的INSTALLED\_APPS属性可以查询出当前项目的应用app。



# 以portal应用为例，实现数据库的迁移操作：

# 创建迁移文件

$ python manage.py makemigrations portal --name portal\_db



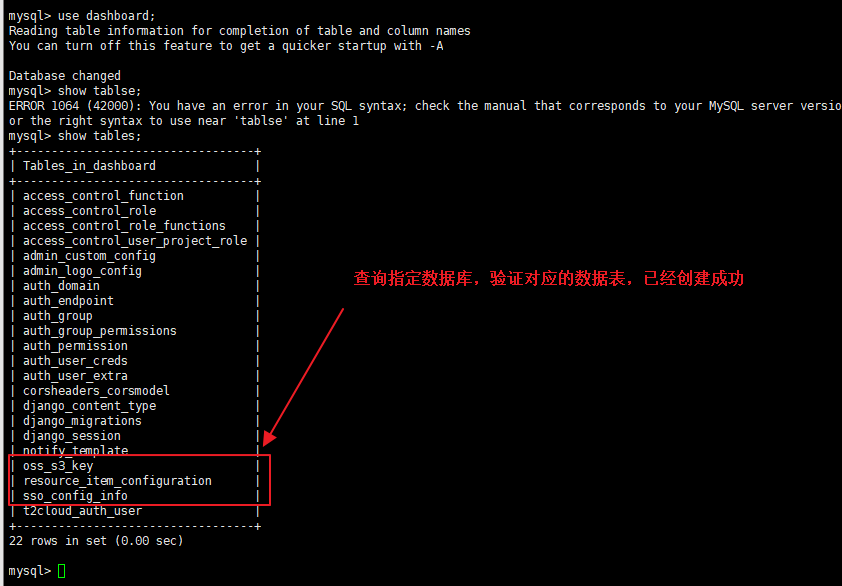
# 执行文件的迁移操作

$ python manage.py migrate

$ python manage.py migrate –list

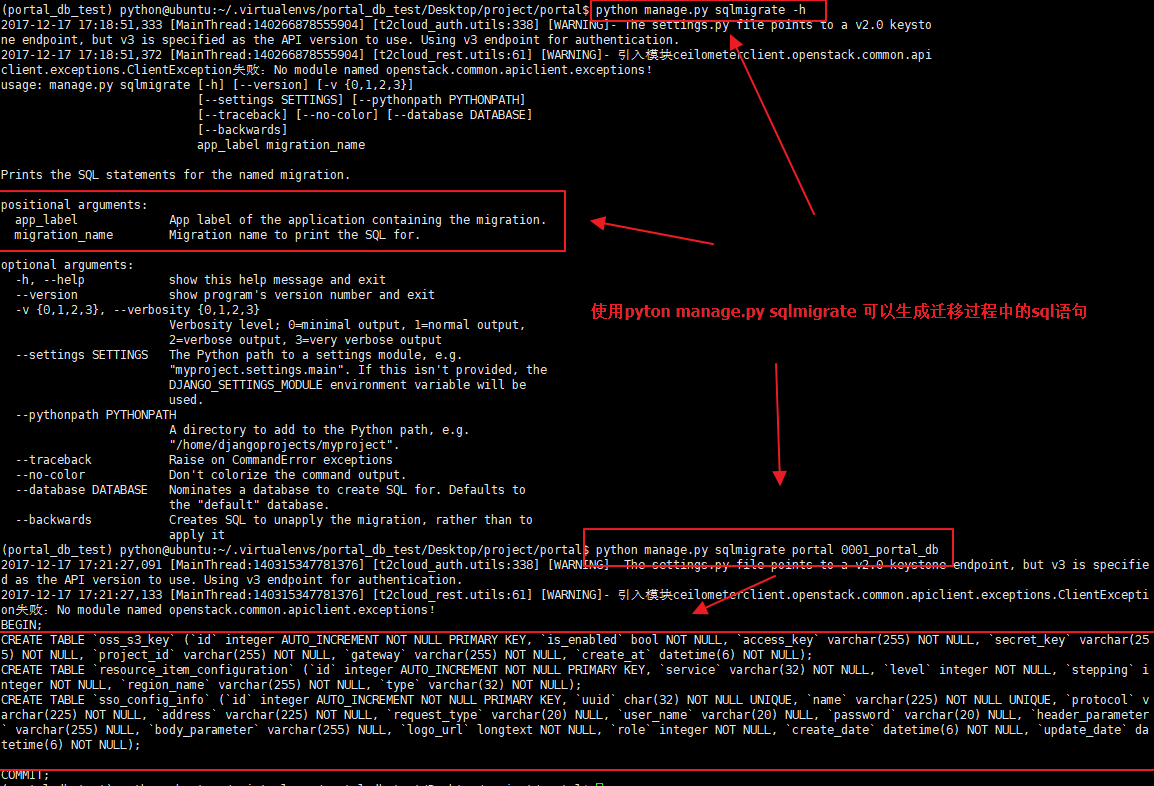


# 查看数据库，验证对应的结果。

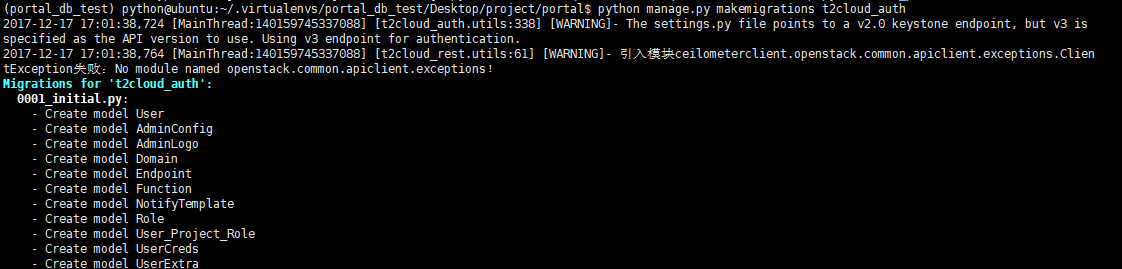


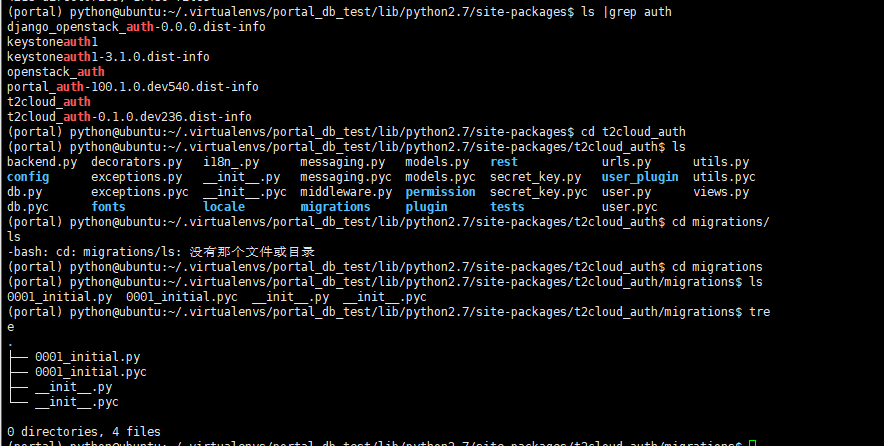
# 查看执行过程的sql语句

$ python manage.py sqlmigrate portal 0001\_portal\_db



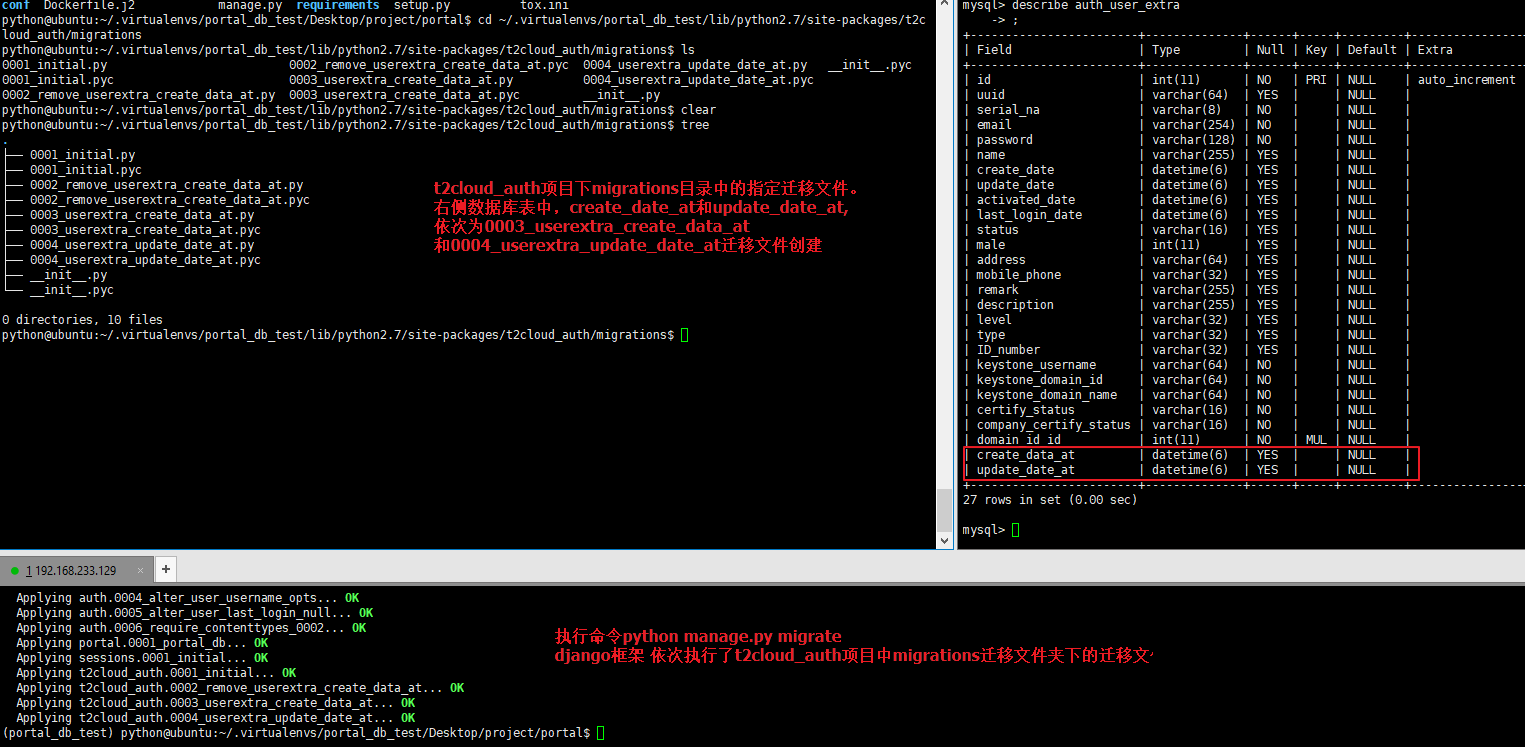
# 指定生成指定项目(t2cloud\_auth)的迁移文件





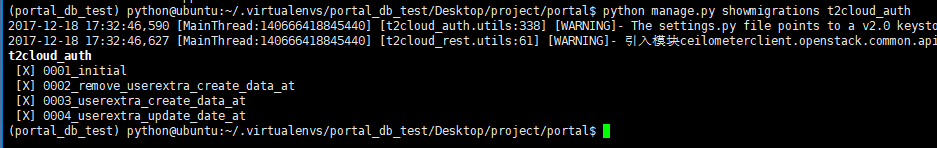
1. 以t2cloud\_auth项目为例，进行迁移的版本控制：

# dashboard库的auth\_user\_extra表为例

# python manage.py migrate 

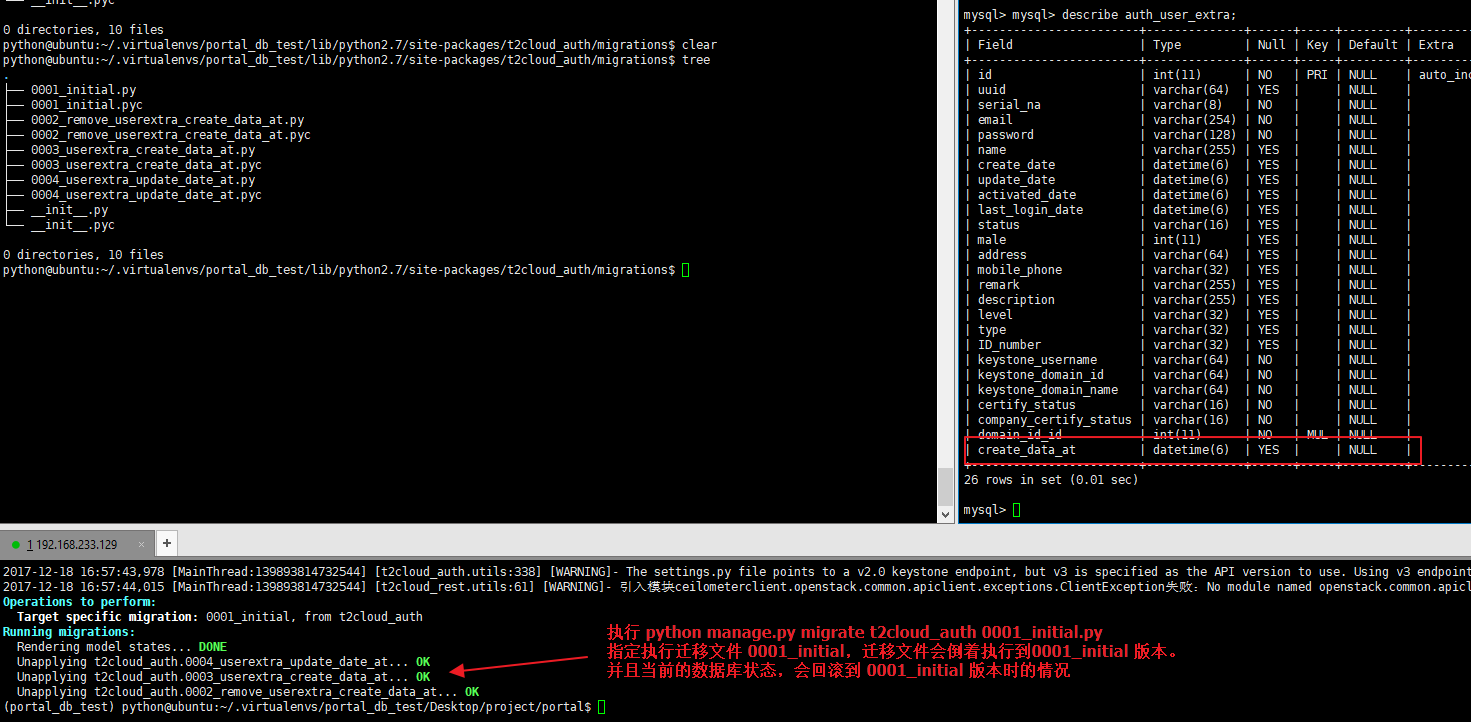
# 查看指定t2cloud\_auth项目中，对应的生效的migrations文件

$ python manage.py showmigrations t2cloud\_auth



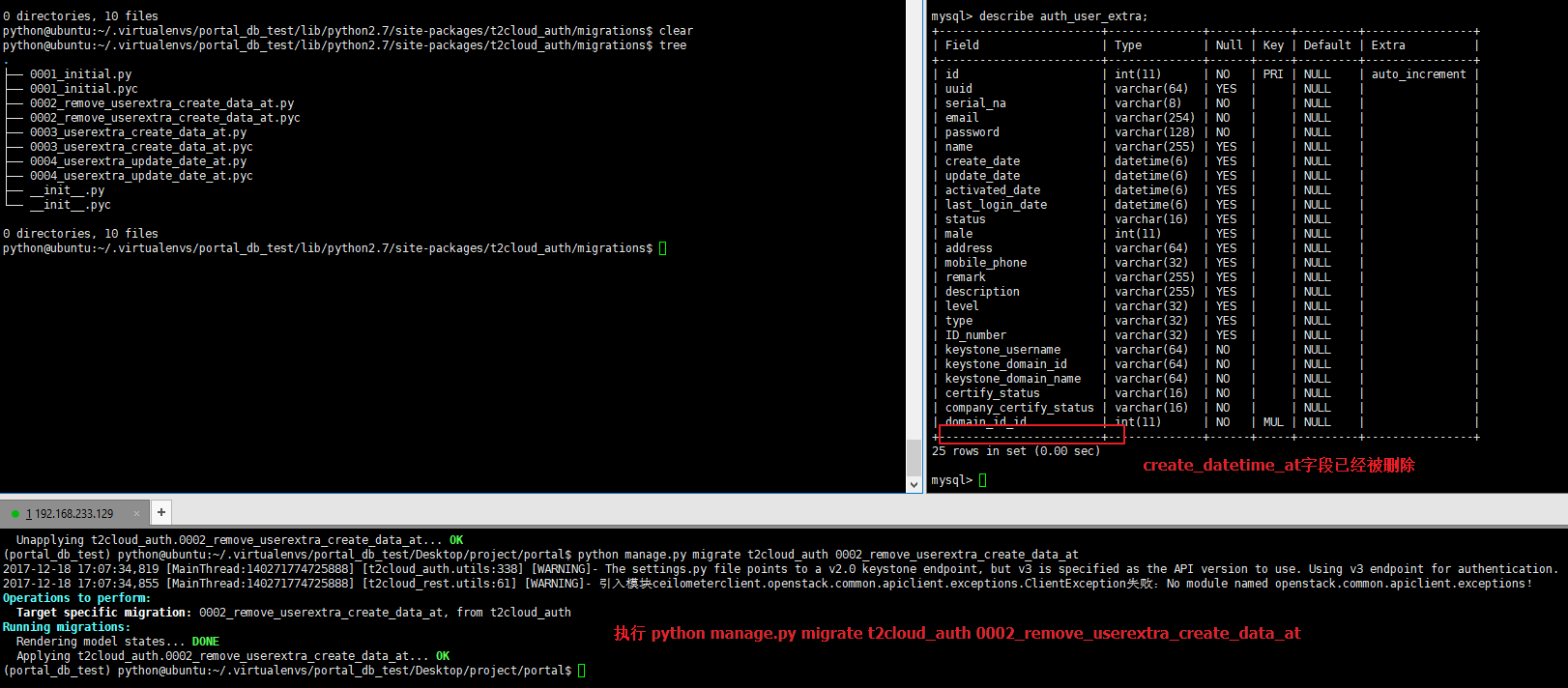
# 进行迁移数据的回归，例如要回滚操作到

# python manage.py migrate t2cloud\_auth 0001\_initial

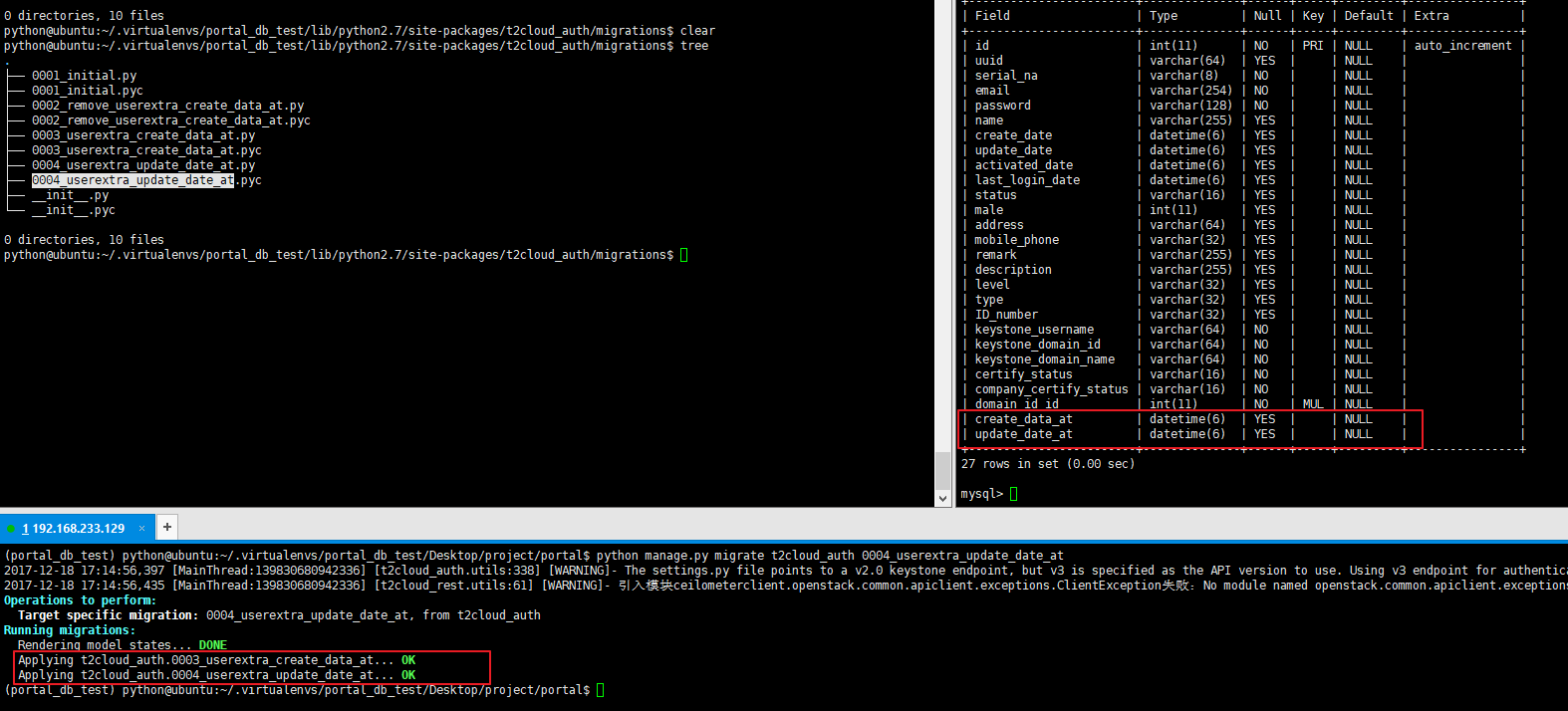


# 之后依次执行下面的操作，来控制数据库中数据的迁移。

# python manage.py migrate t2cloud\_auth 0001\_initial



# python manage.py migrate 0004\_userextra\_update\_date\_at

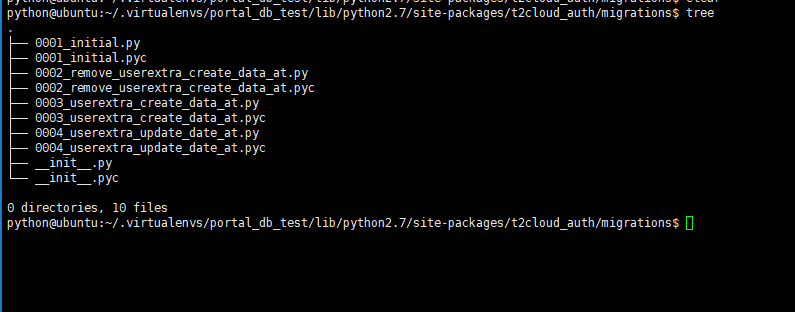


# 注1：数据库版本的回滚，对models.py 模型没有任何印象。也就是models.py文件中的模型不会根据数据的迁移回滚被修改。

# 注2：手动修改了数据库指定的字段（尤其时删除操作），比如执行create\_data\_at字段的删除，在执行版本迁移的时候，一旦有迁移文件中存在对create\_data\_at字段操作时就会出现报错，因为版本文件的迁移，是按照一个文件一个文件来执行操作的。

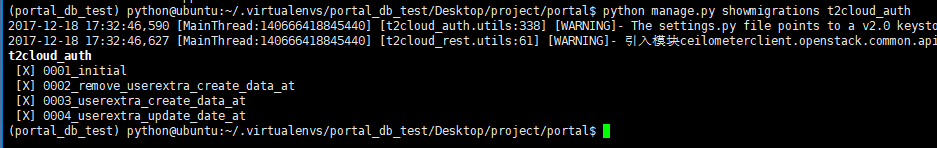
1. 清除指定app的迁移文件和迁移的历史，重新创建一个新的迁移文件

# 如下图显示当前app应用，t2cloud\_auth 的指定迁移文件中，有如下迁移版本。



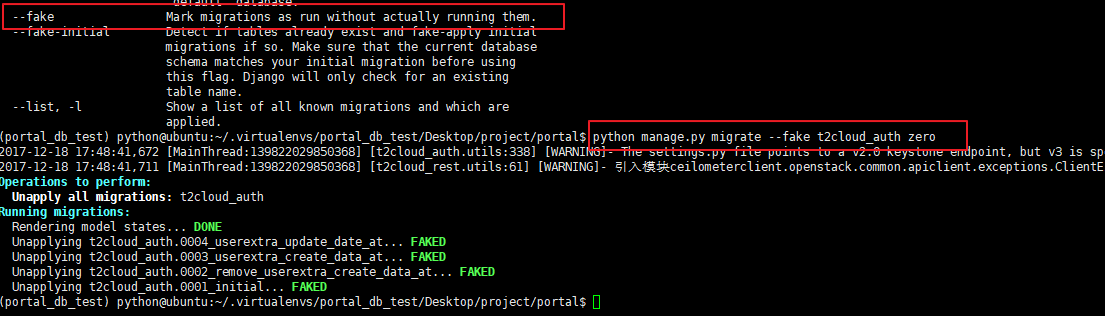
# 查看指定t2cloud\_auth项目中，对应的生效的migrations文件

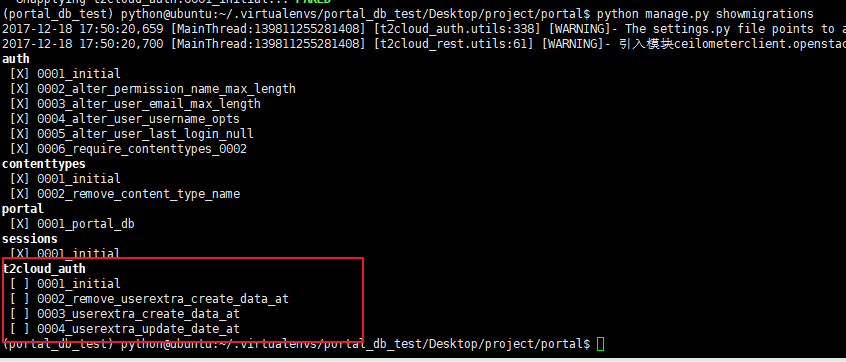
$ python manage.py showmigrations t2cloud\_auth



# 清楚迁移的历史

$ python manage.py migrate --fake t2cloud\_auth zero



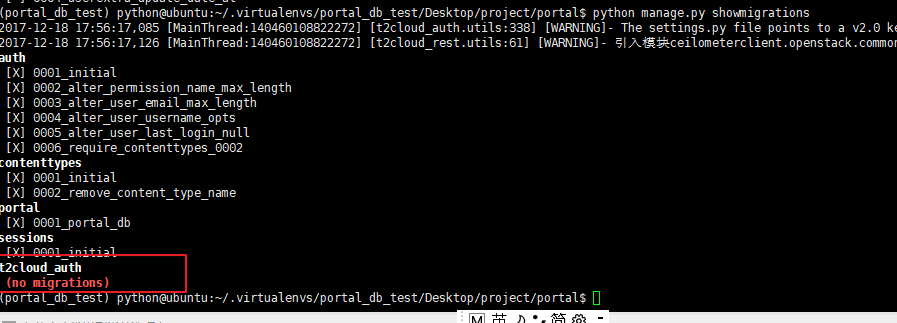


# 删除实际的迁移文件

$ find ./ -path "\*/migrations/\*.py" -not -name "\_\_init\_\_.py" -delete

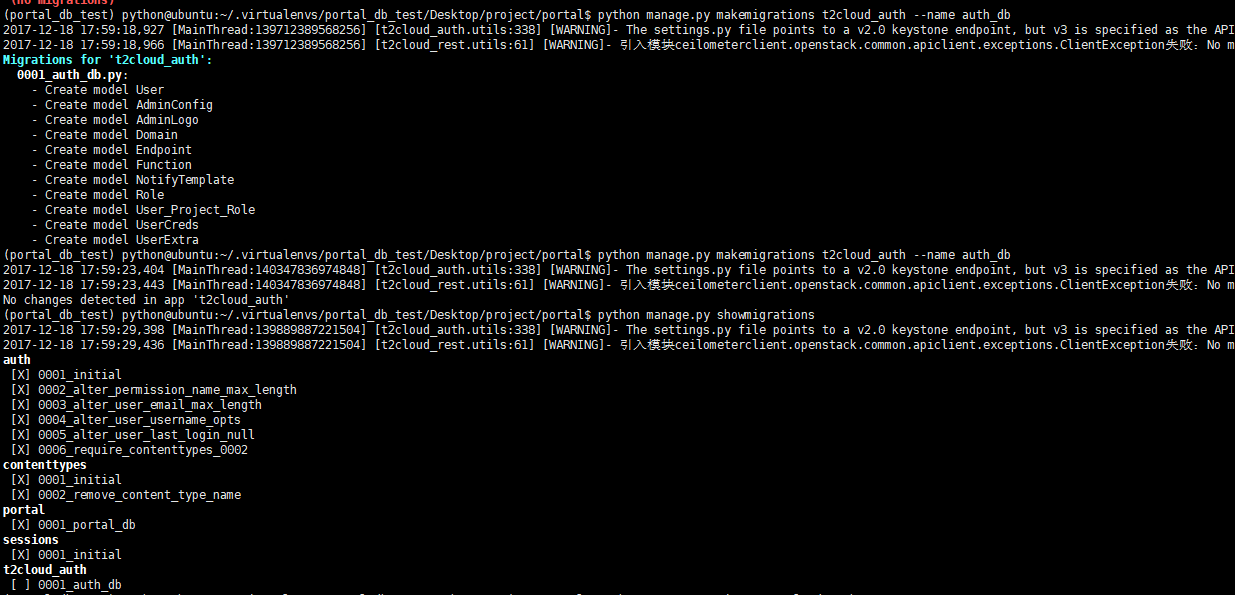
$ find ./ -path "\*/migrations/\*.pyc" -delete

$ python manage.py showmigrations



# 生成新的迁移文件

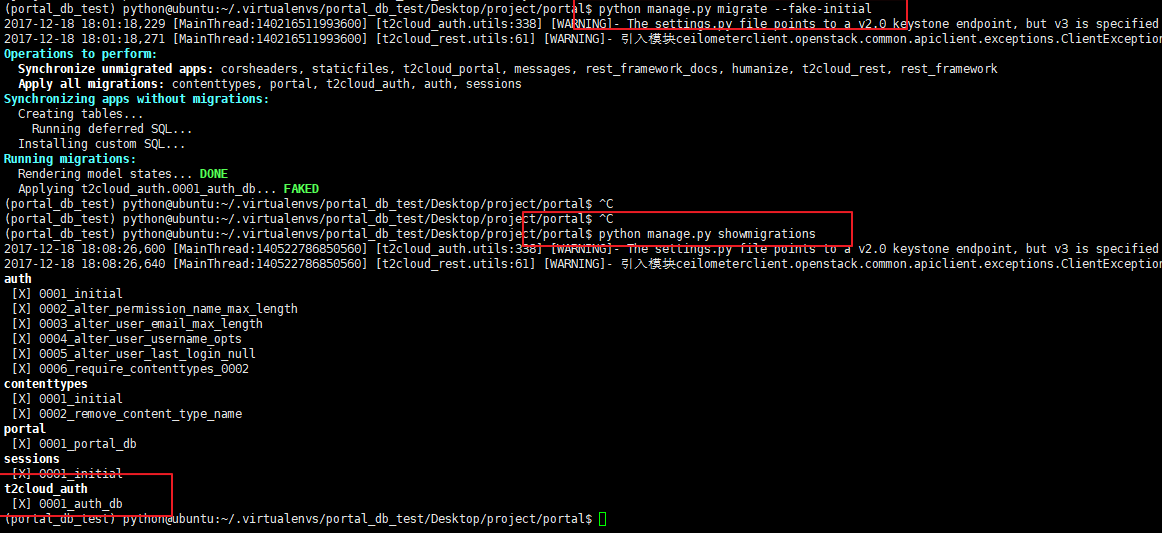
$ python manage.py makemigrations t2cloud\_auth --name auth\_db



# 如果这个时候使用python manage.py migrate 的化，会提示部分表已经存在，无法应用初始迁移，要执行一下命令，伪造初始迁移。

$ python manage.py migrate --fake-initial

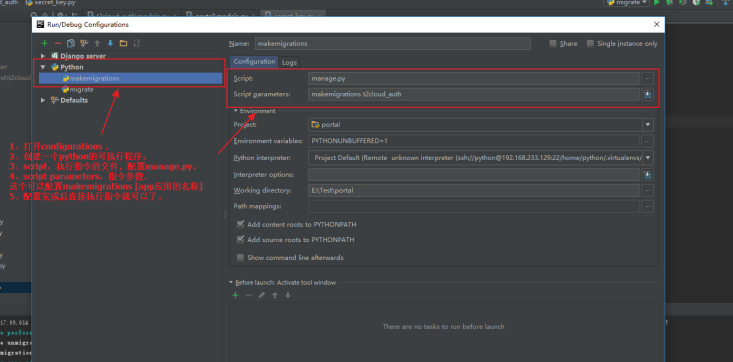
$ python manage.py showmigrations



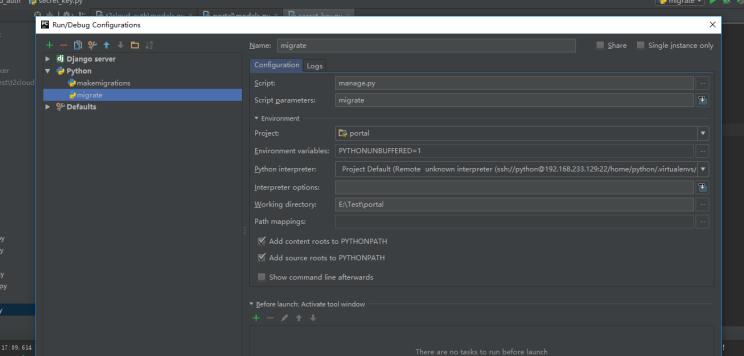
1. 应用软件辅助操作：

1、pycharm IDE下如何进行对应的迁移的操作和调试

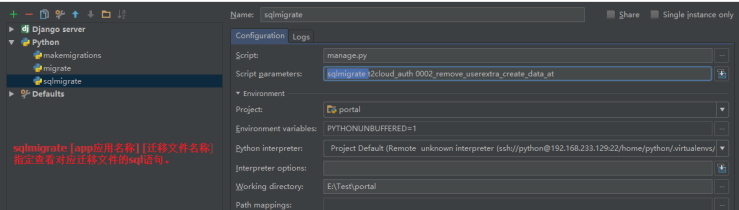
# IDE配置python manage.py makemigrations



# IDE 配置python manage.py migrate，与上面一样的道理

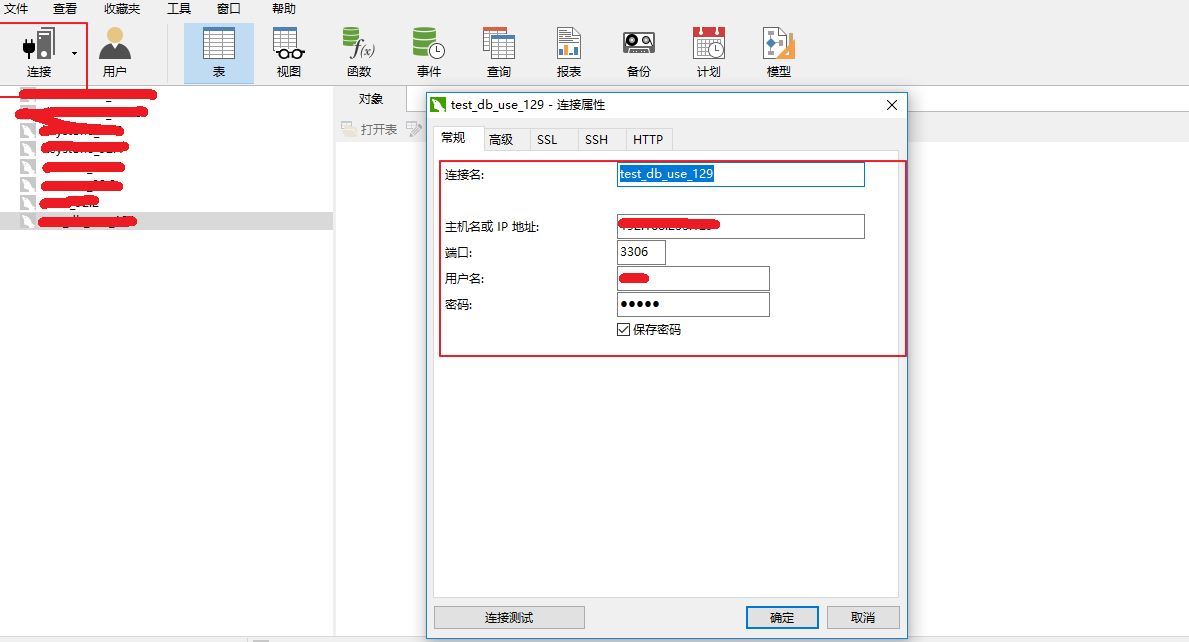


# IDE配置python manage.py sqlmigrate



2、navicat 进行数据库的对应操作

# 打开navicat 配置一个mysql数据库的链接



1. 扩展内容：
2. Setting 配置文件

# MIGRATION\_MODULES

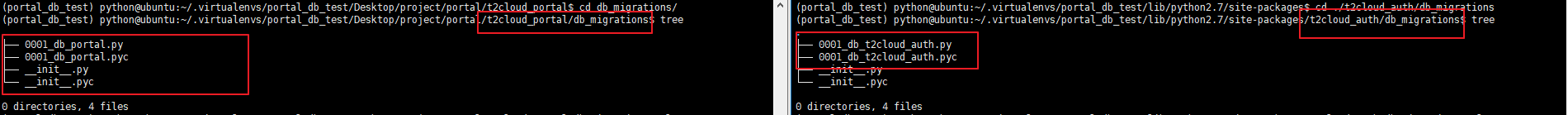
# 指定可在每个应用程序的基础上找到迁移模块的软件包的字典，此设置的默认值为空字典，但是迁移模块的默认软件包名称为migrations。



# python manage.py migrate

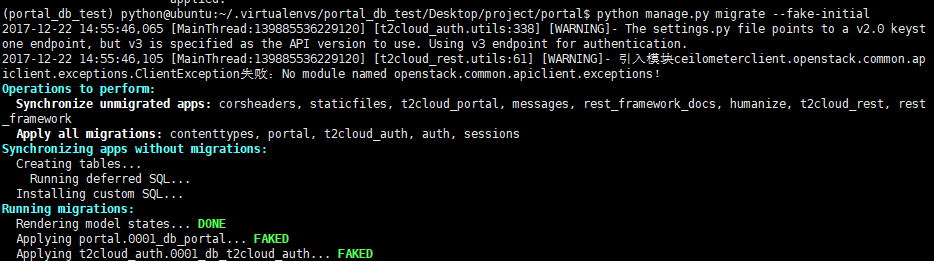
# python manage.py makemigrations t2cloud\_auth --name db\_auth

# python manage.py makemigrations portal --name db\_portal



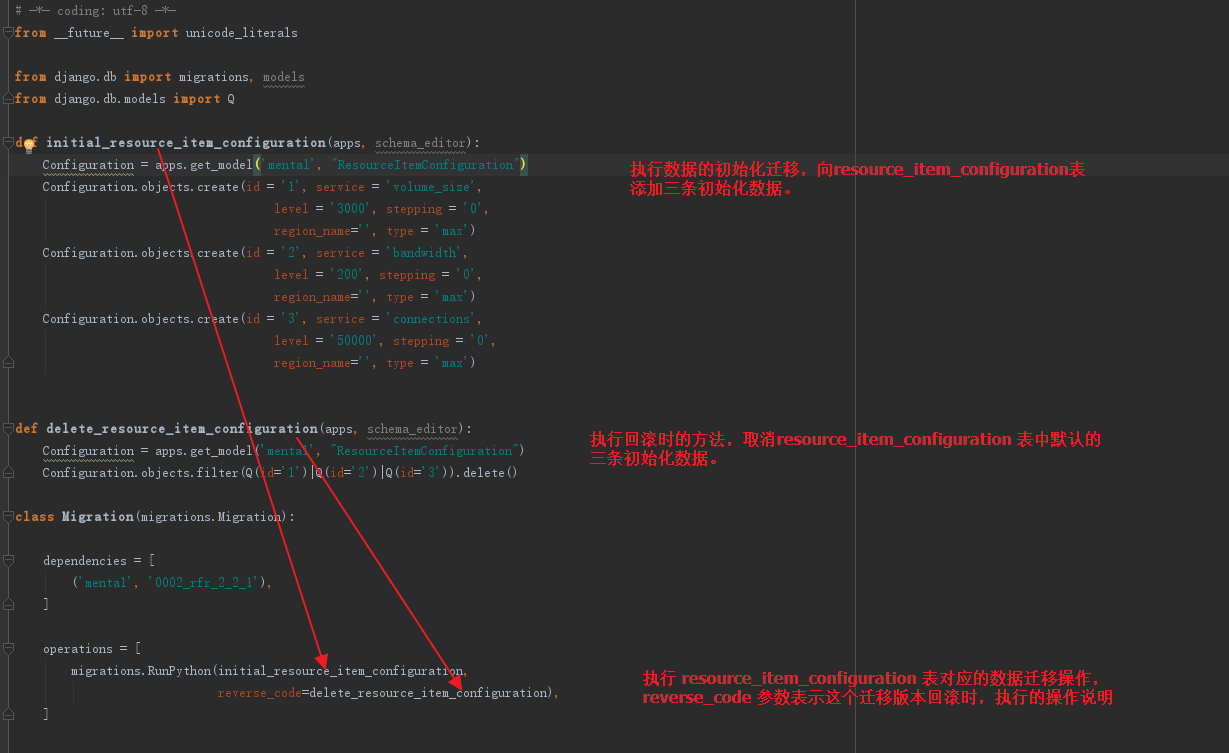
# 再次执行 python manage.py migrate 时，会提示指定的表已经存在。可使用--fake-initial 参数

# python manage.py migrate --fack-initial



1. 迁移的过程中添加初始化的数据
2. 有些时候，只初始化数据的表结构是不能满足需求的，还需要为指定的表添加指定的初始话数据信息：

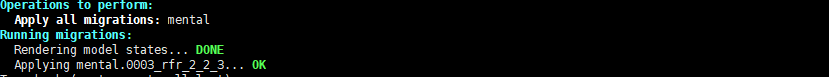
# 以 resource\_item\_configuration 表为例，使用迁移文件进行数据表中数据的初始化操作。

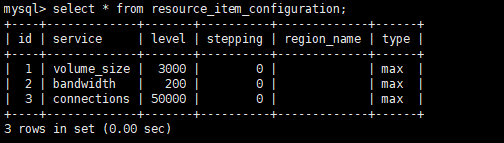


# python manage.py showmigrations mental

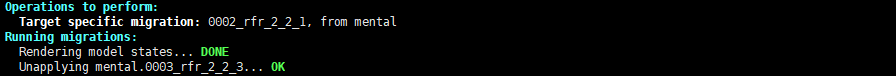


# python manage.py migrate mental





# python manage.py migrate mental 0002\_rfr\_2\_2\_1 (回滚，清楚初始化数据)

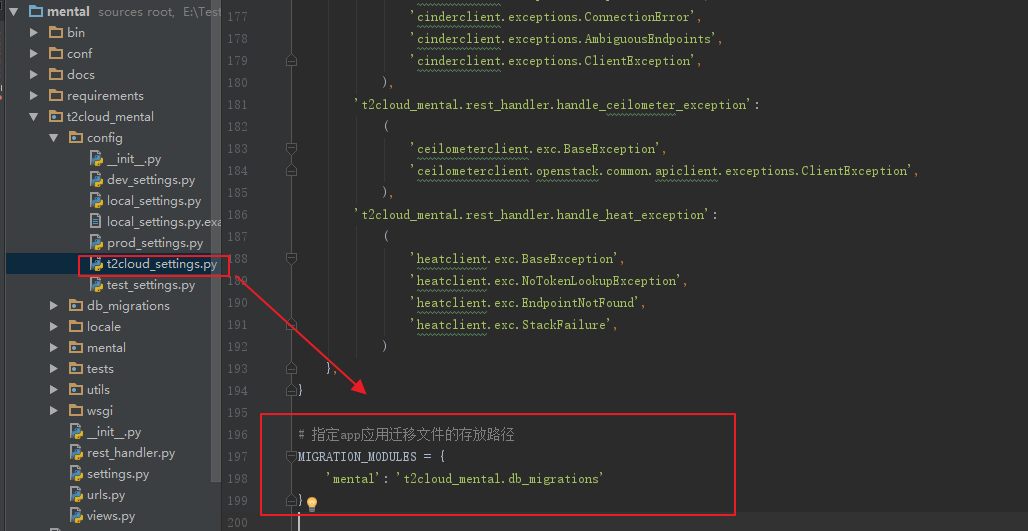




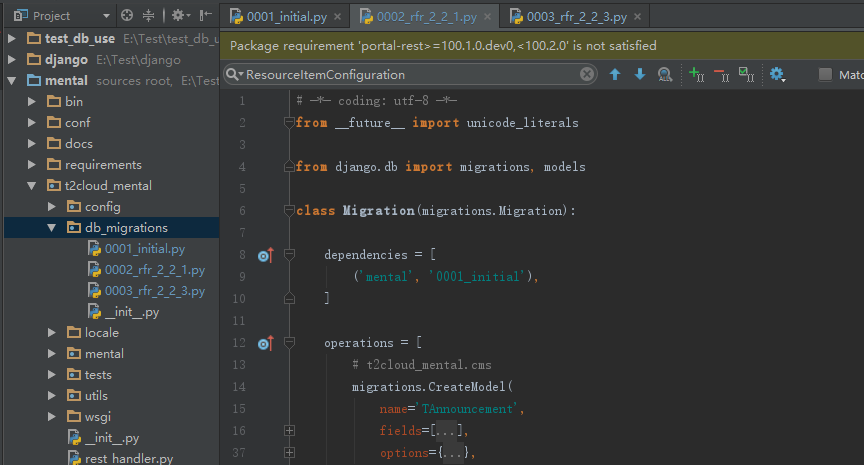
1. 进行数据库的实际迁移工作
2. 目前项目的dashboard 数据库的初始化脚本文件存放在mental项目中。为了能更好的对整个dashboard进行迁移版本的控制，初步决定在mental项目中创建一个db\_migrations文件夹进行dashboard整个项目的迁移文件统一存放，实现数据迁移的版本控制。

|  |  |  |  |
| --- | --- | --- | --- |
| 项目版本 | 迁移文件版本控制 | 当前使用版本 | 说明 |
|  | 0001\_initial.py |  | 初始化的版本 |
| rfr/2.2.1 | 0002\_rfr\_2.2.1.py |  | 2.2.1版本 |
| rfr/2.2.2 | 0003\_rfr\_2.2.2.py |  | 2.2.2版本 |
| rfr/2.2.3 | 0004\_rfr\_2.2.3.py |  | 2.2.3版本 |
| rfr/2.2.4 | 0005\_rfr\_2.2.4.py |  | 2.2.4版本 |

1. 初始化mental下面的脚本存放路径



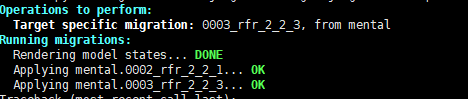
1. 手动编写对应的迁移文件脚本，并配置迁移的顺序。将脚本文件统一存放起来方便之后的管理操作。由于不太清楚rfr/2.2.3 和 rfr/2.2.4 两个版本的区别，所以目前先以 mental、portal、mental\_cms、auth这个四个应用进行控制操作。



# 如上图所示 创建了三个初始化的迁移文件。通过设置dependencies参数，控制迁移版本的顺序。Intital.py存放着 t2cloud\_auth和mental 的表结构迁移内容；0002\_rfr\_2\_2\_1.py 中存放着 mental.cms和portal的表结构迁移内容；0003\_rfr\_2\_2\_3.py中存放着resource\_item\_configuration表的初始化数据迁移内容。

# 进行mental项目的manage.py 文件目录下，执行迁移操作

# python manage.py migrate mental

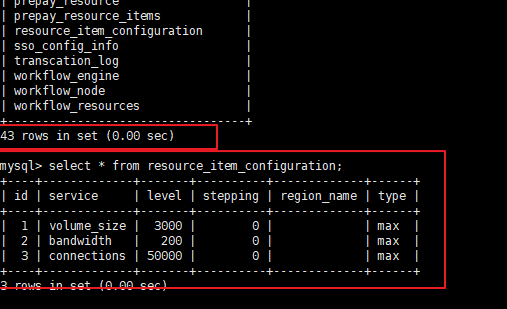


验证执行结果

>use dashboard;

>show tables;

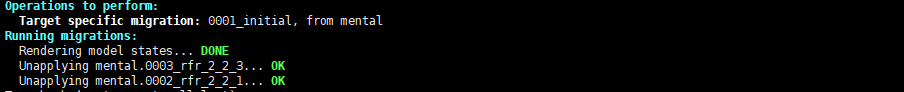
> select \* from resource\_item\_configuration;



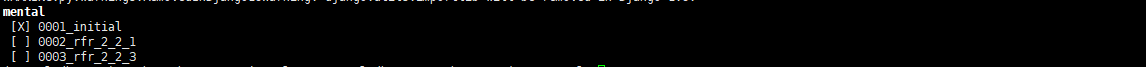
43个数据表和3条初始化数据，迁移成功。

# 验证回滚

# python manage.py migrate mental 0001\_initial



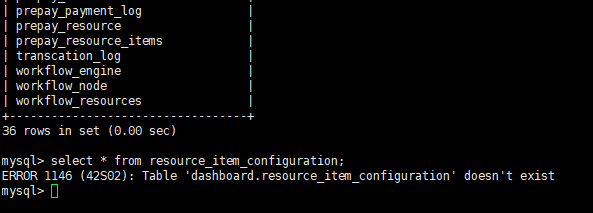
# python manage.py showmigrations mental



>use dashboard;

>show tables;

>select \* from resource\_item\_configuration;



数据表确认减少到36张，并且指定的表resource\_item\_configuration 被删除，并且数据也被删除。

1. 引用文献：

<http://python.usyiyi.cn/documents/django_182/topics/migrations.html>

<https://simpleisbetterthancomplex.com/tutorial/2016/07/26/how-to-reset-migrations.html>