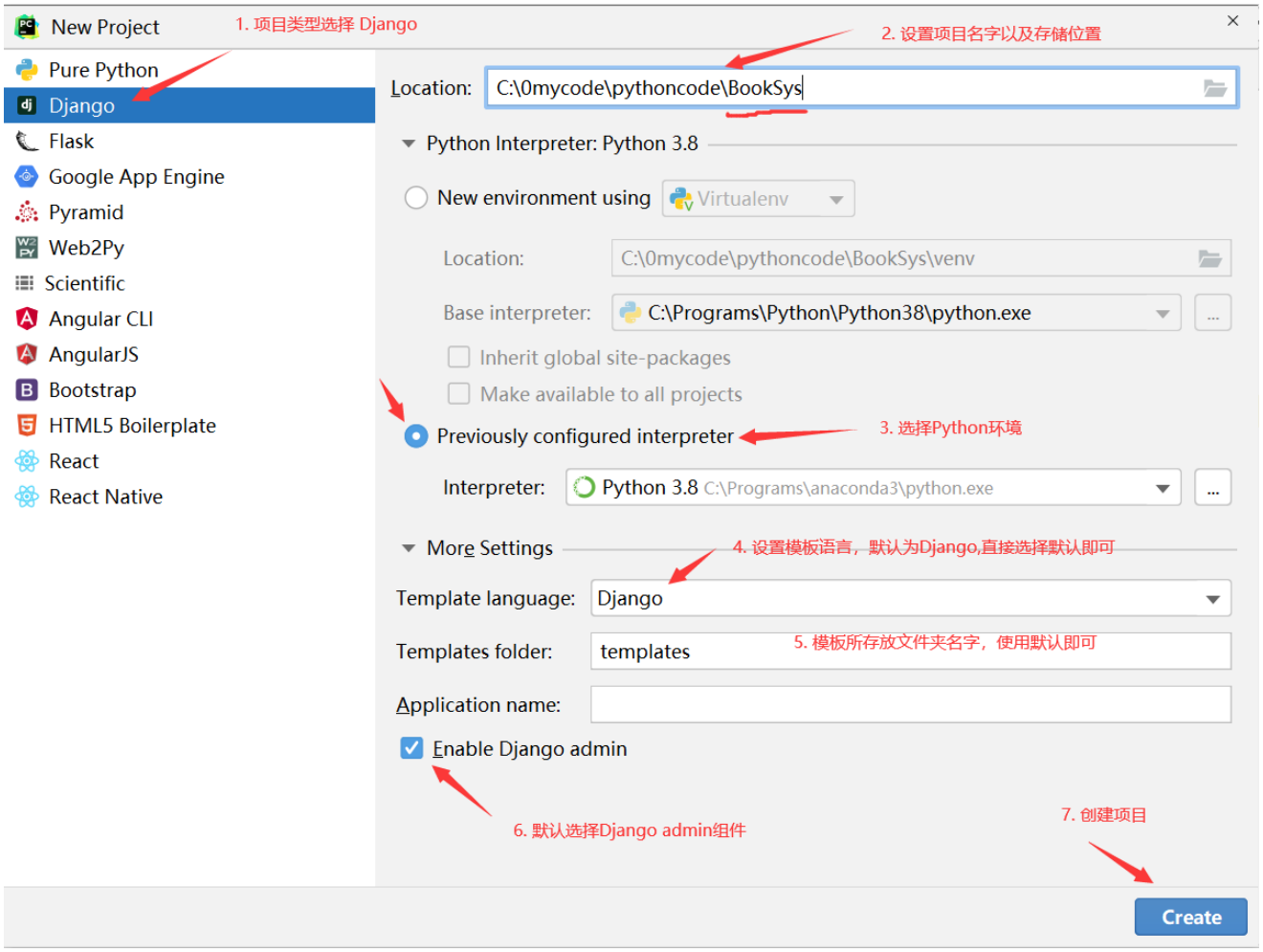
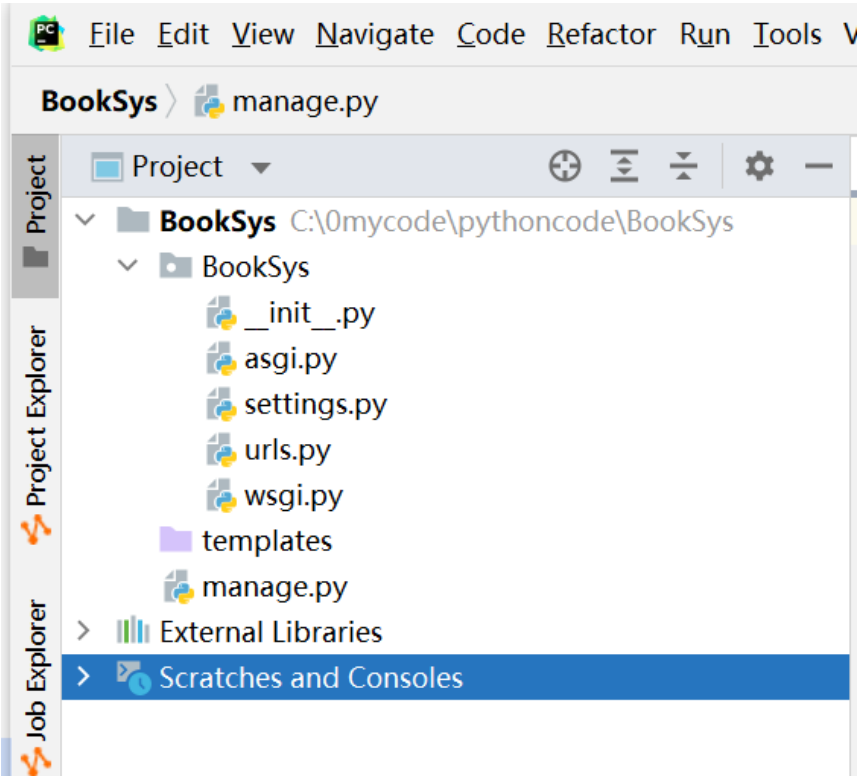
一、 使用Pycharm创建Django项目

1. 创建项目

打开Pycharm，File-->New Project



2. 项目结构以及配置文件介绍



BookSys 项目下二级子目录 BookSys 下的文件称为 Django 项目的配置文件， 它们在创建项目的时候自动生成。详细介绍如下：

1) manage.py文件

一级子目录中的 manage.py 文件是管理 Django 项目的重要命令行工具，它主要用于启动项目、创建应用和完成数据库的迁移等。

2) __init__.py文件

二级子目录中的 __init__.py 文件用于标识当前所在的目录是一个 Python 包，如果在此文件中，通过 import 导入其他方法或者包会被 Django 自动识别。

3) settings.py文件

settings.py 文件是 Django 项目的重要配置文件。项目启动时，settings.py 配置文件会被自动调用，而它定义的一些全局为 Django 运行提供参数，在此配置文件中也可以自定义一些变量，用于全局作用域的数据传递。

4) urls.py文件

url.py 文件用于记录 Django 项目的 URL 映射关系，它属于项目的基础路由配置文件，路由系统就是在这个文件中完成相应配置的，项目中的动态路径必须先经过该文件匹配，才能实现 Web 站点上资源的访问功能。

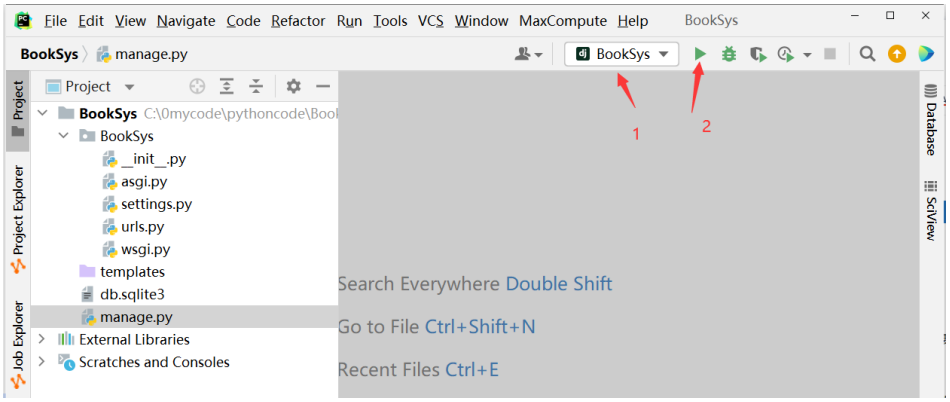
5) wsgi.py文件

wsgi.py 是 WSGI（Web Server Gateway Interface）服务器程序的入口文件，主要用于启动应用程序。它遵守 WSGI 协议并负责网络通讯部分的实现，只有在项目部署的时候才会用到它。

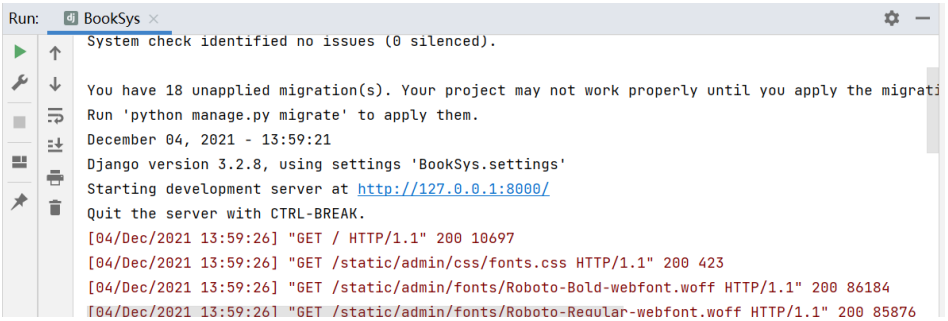
二、 启动项目

1. 启动方式一：通过Pycharm来启动

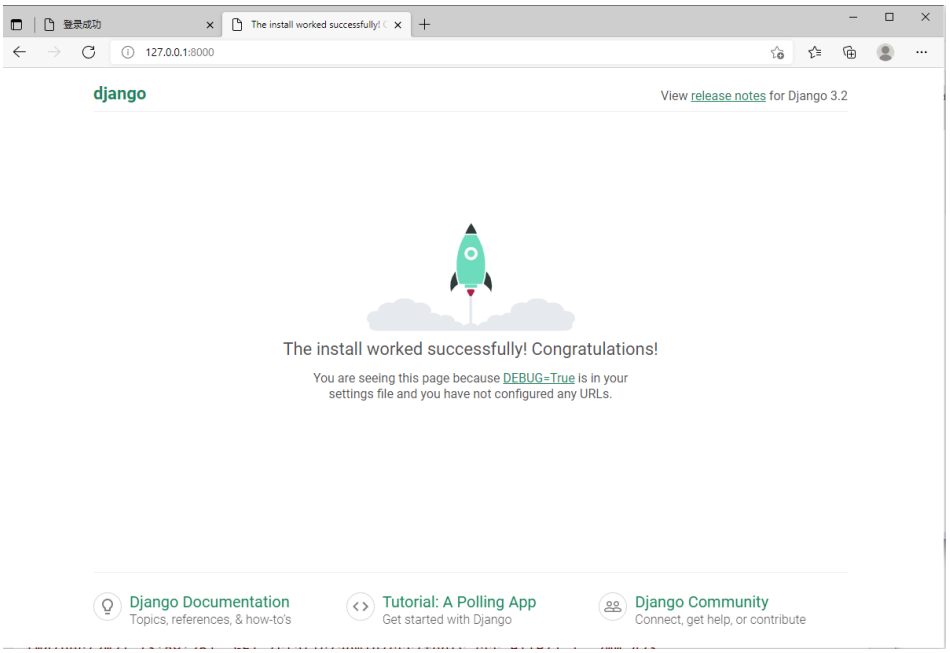
- 运行BookSys项目



- 控制台中显示如下信息，



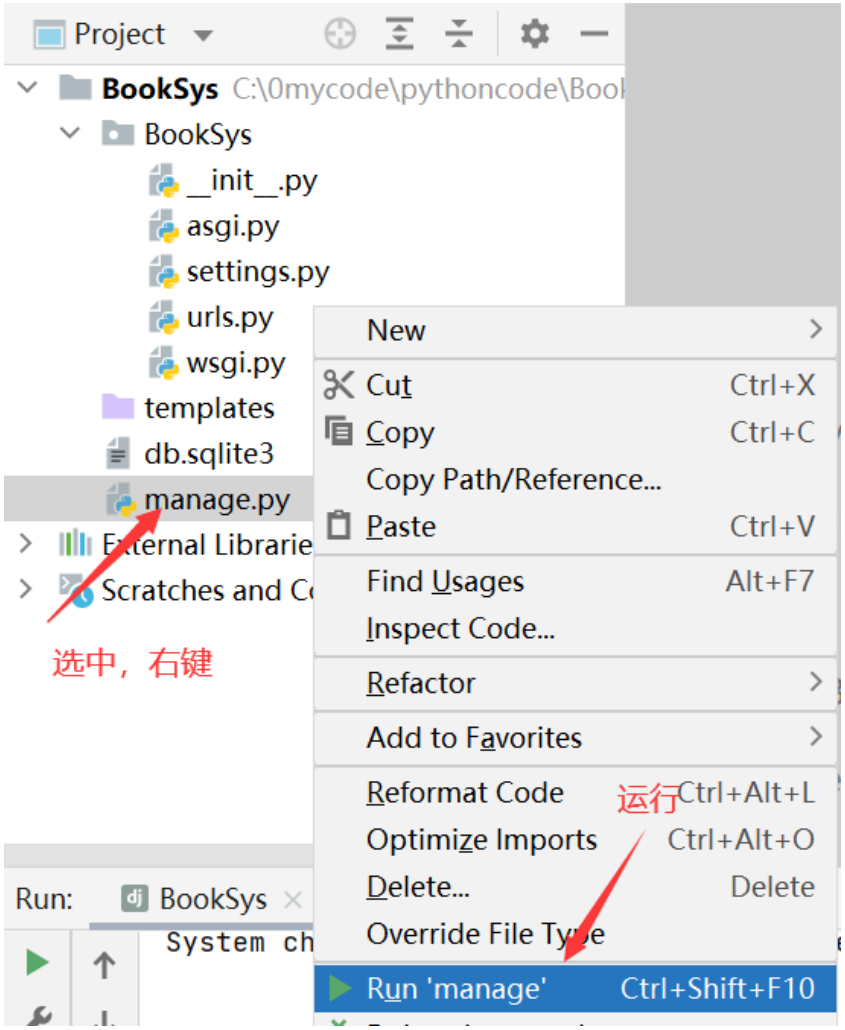
- 在浏览器中访问 **http://127.0.0.1:8000**

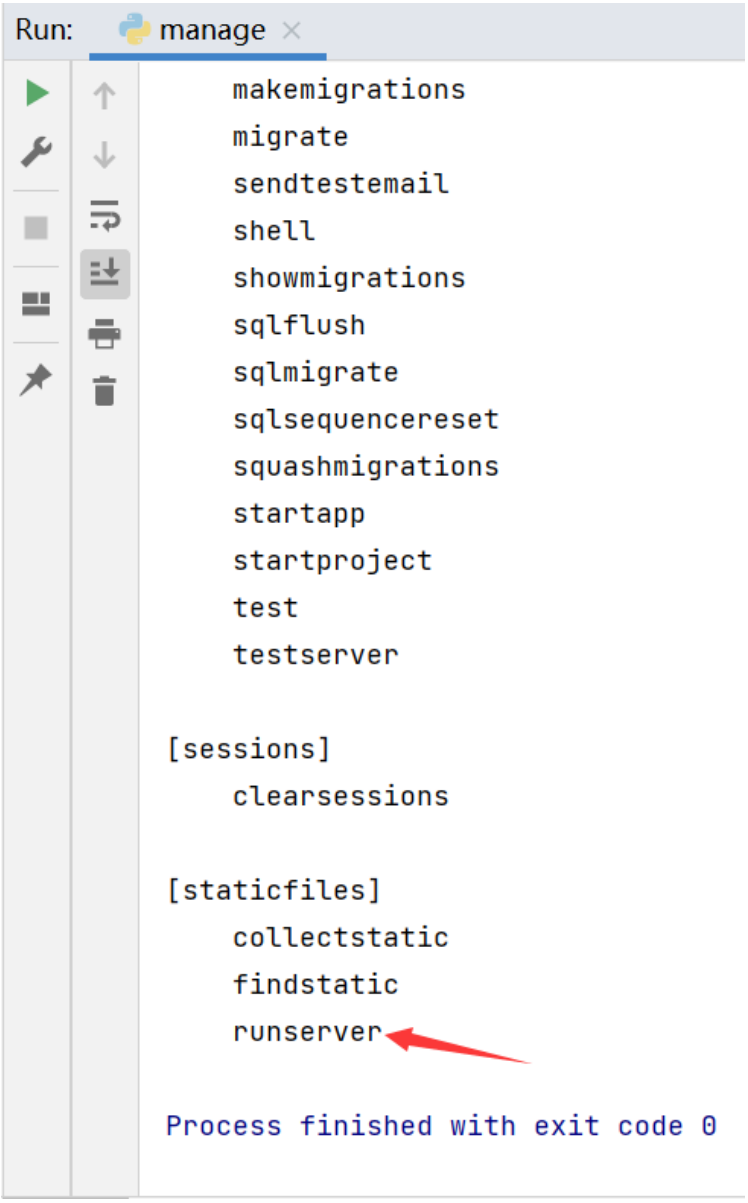


说明：如能出现以上页面，则Django启动成功

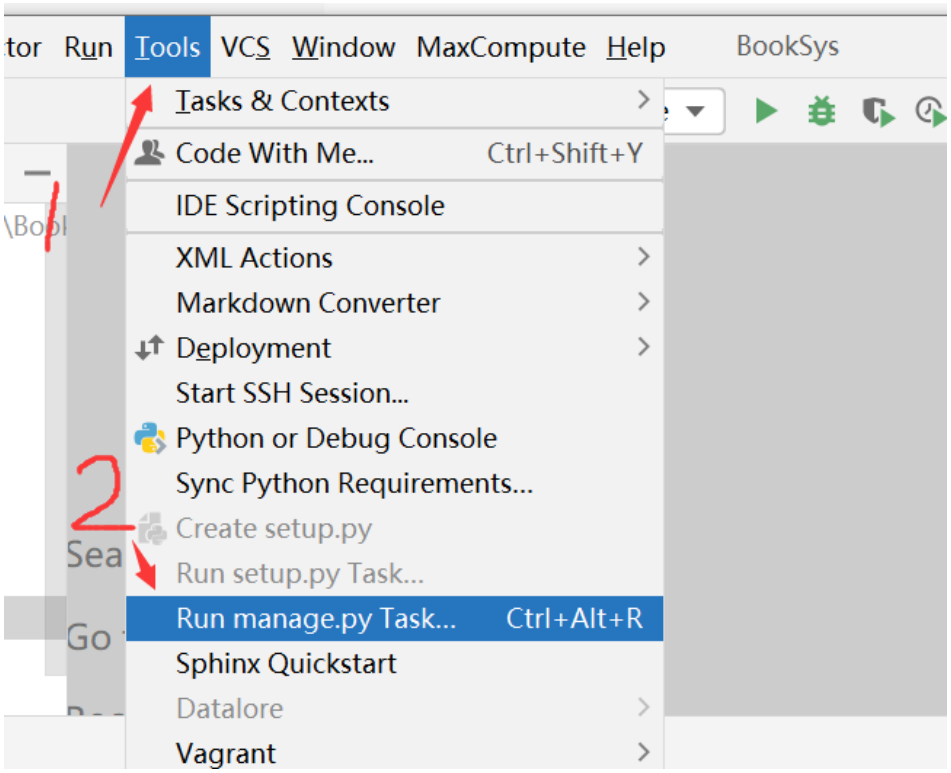
2. 启动方式二：通过启动manager命令行来启动

- 通过运行manager命令获取django命令列表

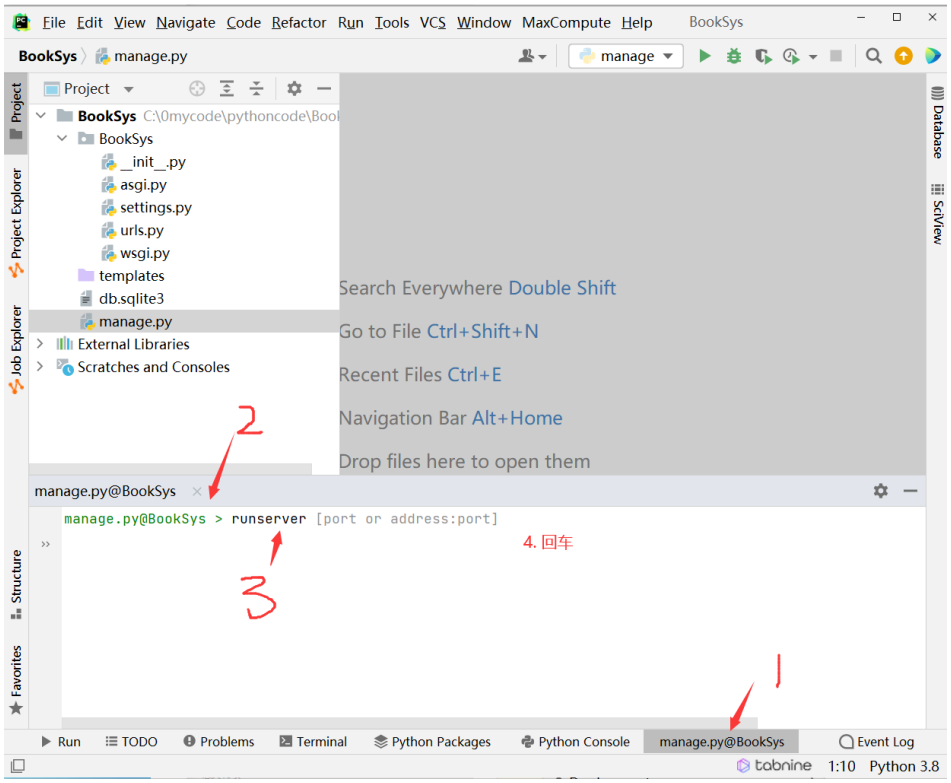

Project
BookSys C:\0mycode\pythoncode\Book
BookSys
__init__.py
asgi.py
settings.py
urls.py
wsgi.py
templates
db.sqlite3
manage.py
External Librarie
Scratches and C
选中，右键
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path/Reference...
Paste Ctrl+V
Find Usages Alt+F7
Inspect Code...
Refactor
Add to Favorites
Reformat Code Ctrl+Alt+L
运行
Optimize Imports Ctrl+Alt+O
Delete... Delete
Override File Type
Run 'manage' Ctrl+Shift+F10
Run:
BookSys
System ch



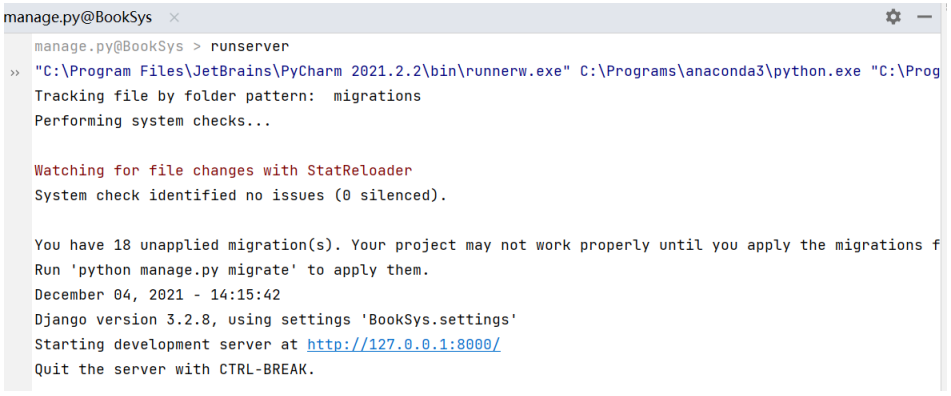
runserver是启动Django服务的命令



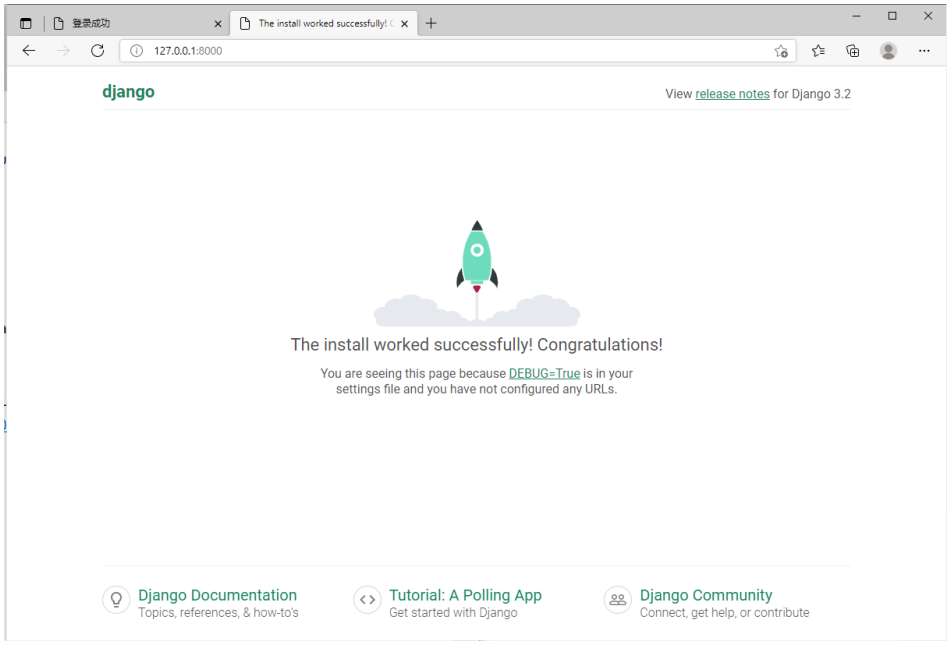
在弹出的manage.py控制台运行窗口中输入命令runserver



运行结果

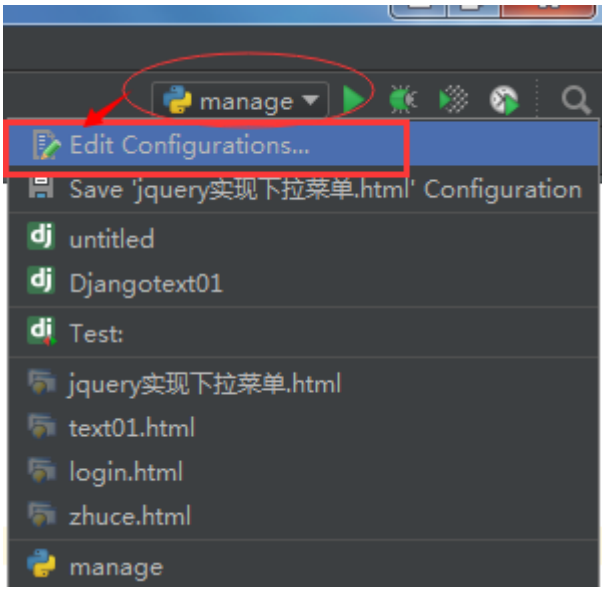


打开浏览器，访问http://127.0.0.1:8000,如出现如下界面，则启动成功

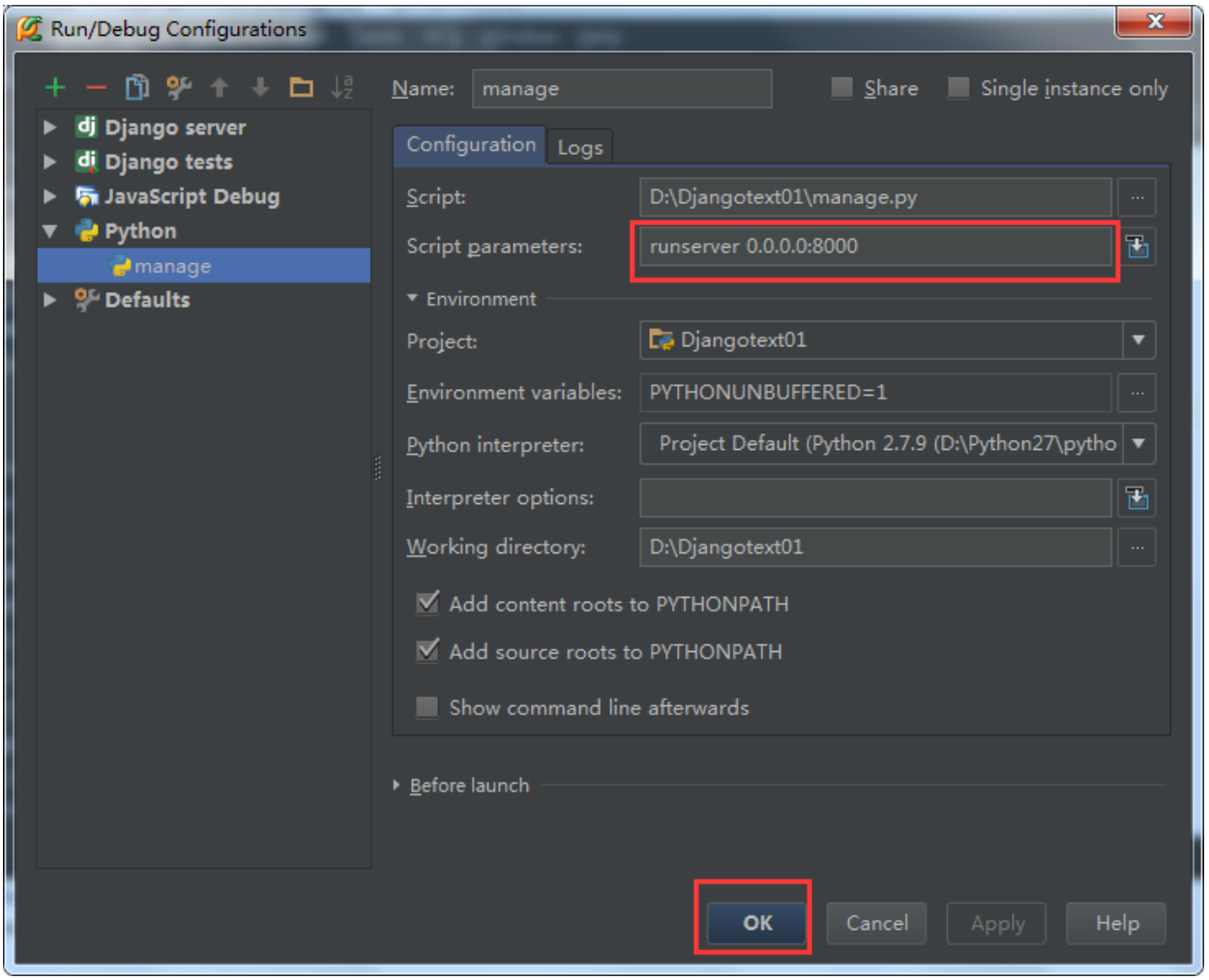


在pycharm中启动Django服务器

1.要是直接运行manage.py程序的话 会提示一大堆东西，那无非是提示没有传入参数。先打开mange.py，然后再运行，会提示一堆东西，表示没有配置参数。在pycharm**右上角**点击**edit configurations** 编辑配置参数。

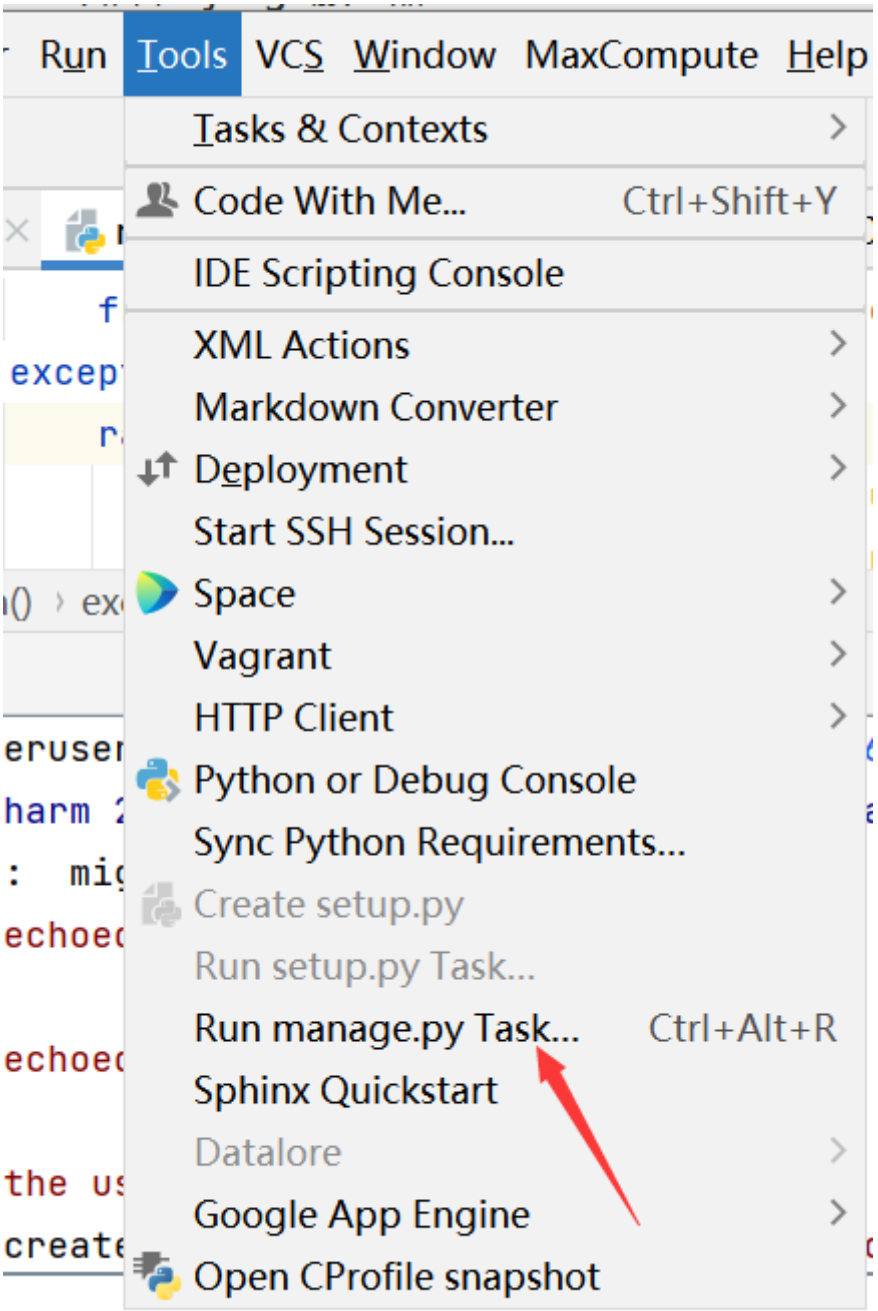


2.点开之后弹出如下对话框，在scrip parameters 对应的对话框中输入配置参数 **runserver 0.0.0.0:8000**.配置完成之后点击ok就完成了。



3.配置完以上信息之后，直接按Ctrl+shift+F10 运行一下manage.py文件：出现如下结果，表示配置成功。

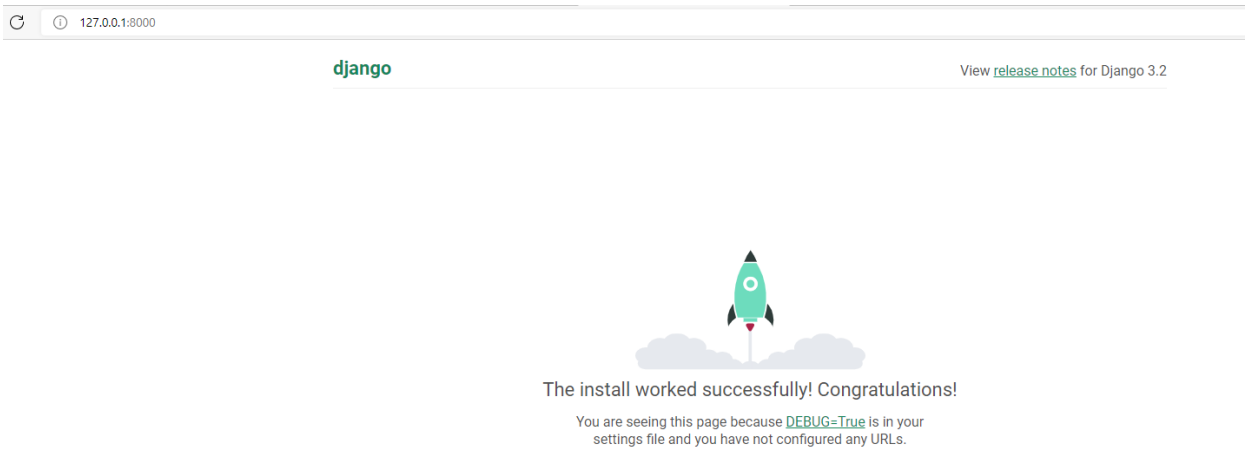
D:\workpace\Python\PythonIDE\python.exe D:/Code/Python/Django/manage.py runserver 0.0.0.0:8000
Performing system checks...
System check identified no issues (0 silenced).
You have 13 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
November 26, 2017 - 11:40:48
Django version 1.11.7, using settings 'Django.settings'
Starting development server at http://0.0.0.0:8000/
Quit the server with CTRL-BREAK.


Run
Tools
VCS
Window
MaxCompute
Help
Tasks & Contexts
Code With Me... Ctrl+Shift+Y
IDE Scripting Console
XML Actions
Markdown Converter
Deployment
Start SSH Session...
Space
Vagrant
HTTP Client
Python or Debug Console
Sync Python Requirements...
Create setup.py
Run setup.py Task...
Run manage.py Task... Ctrl+Alt+R
Sphinx Quickstart
Datalore
Google App Engine
Open CProfile snapshot

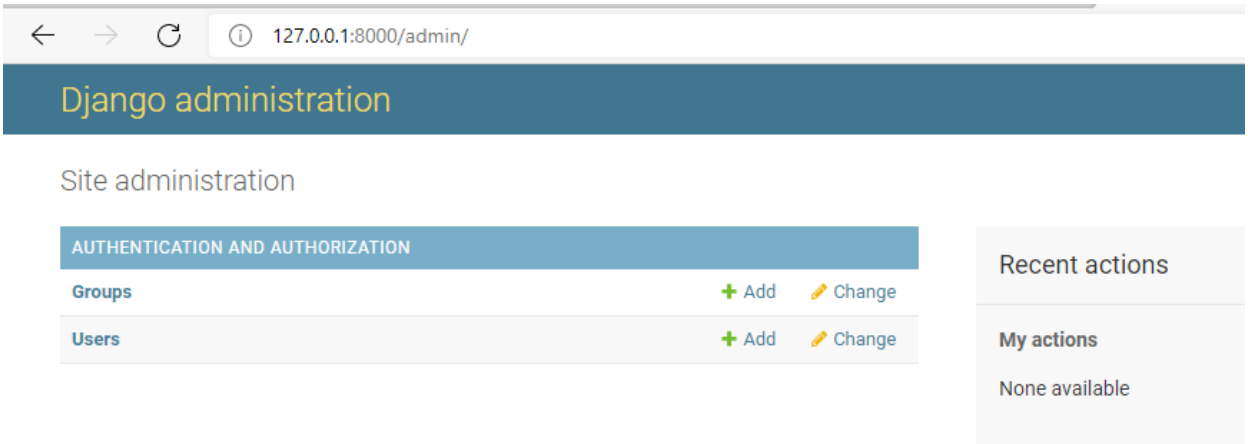
```
Process finished with exit code 0
manage.py@django_01 > runserver
"C:\Program Files\JetBrains\PyCharm 2021.2.2\bin\runnerw.exe" C:\P
Tracking file by folder pattern:  migrations
Performing system checks...

Watching for file changes with StatReloader
System check identified no issues (0 silenced).
November 29, 2021 - 14:56:19
Django version 3.2.8, using settings 'django_01.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

运行命令

在浏览器中访问这个地址

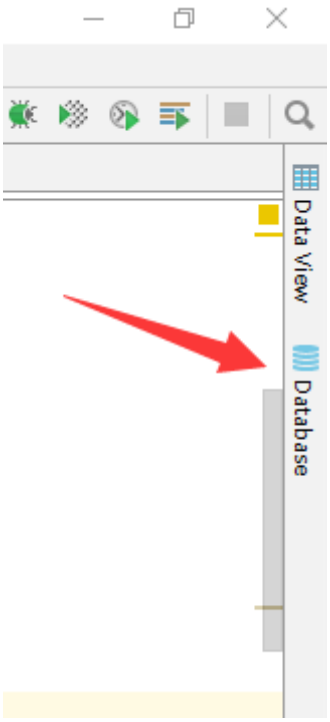


后台地址

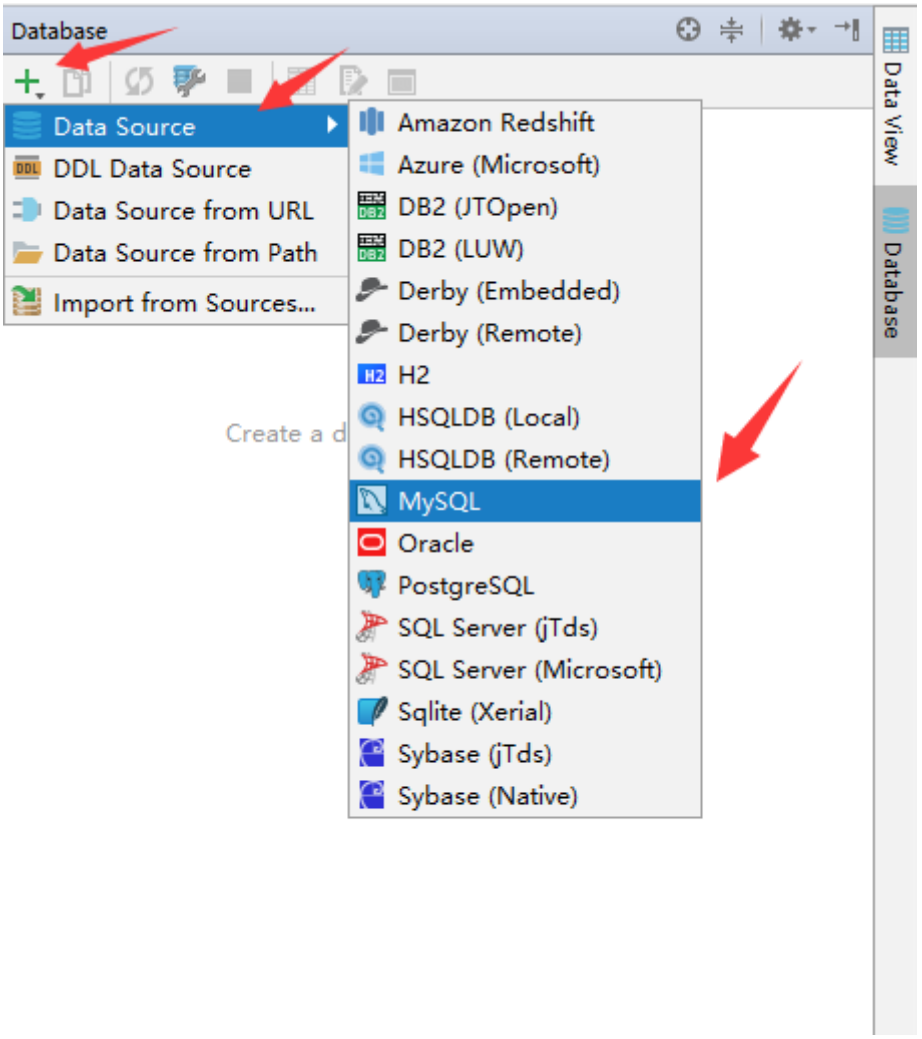


图文详解Django使用Pycharm连接MySQL数据库_python_脚本之家 (jb51.net)

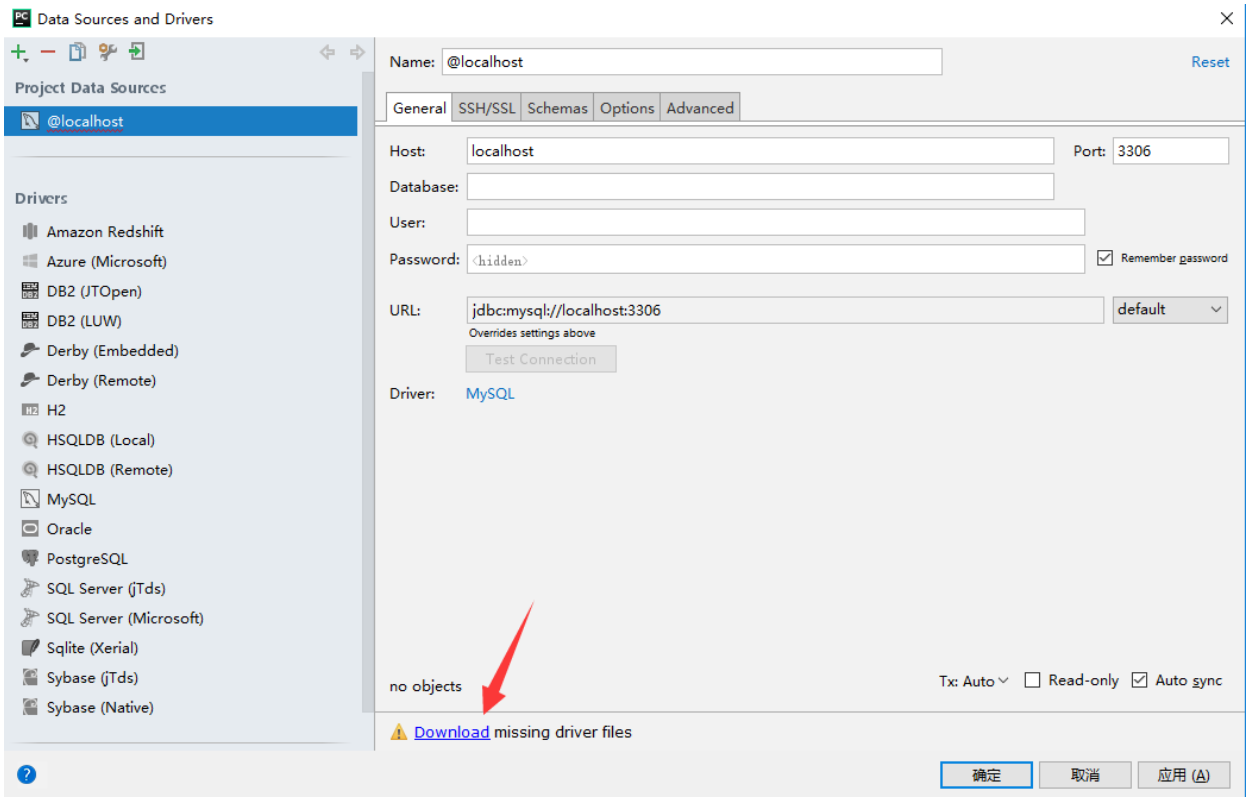
在 Pycharm 的右上方找到 Database



点击

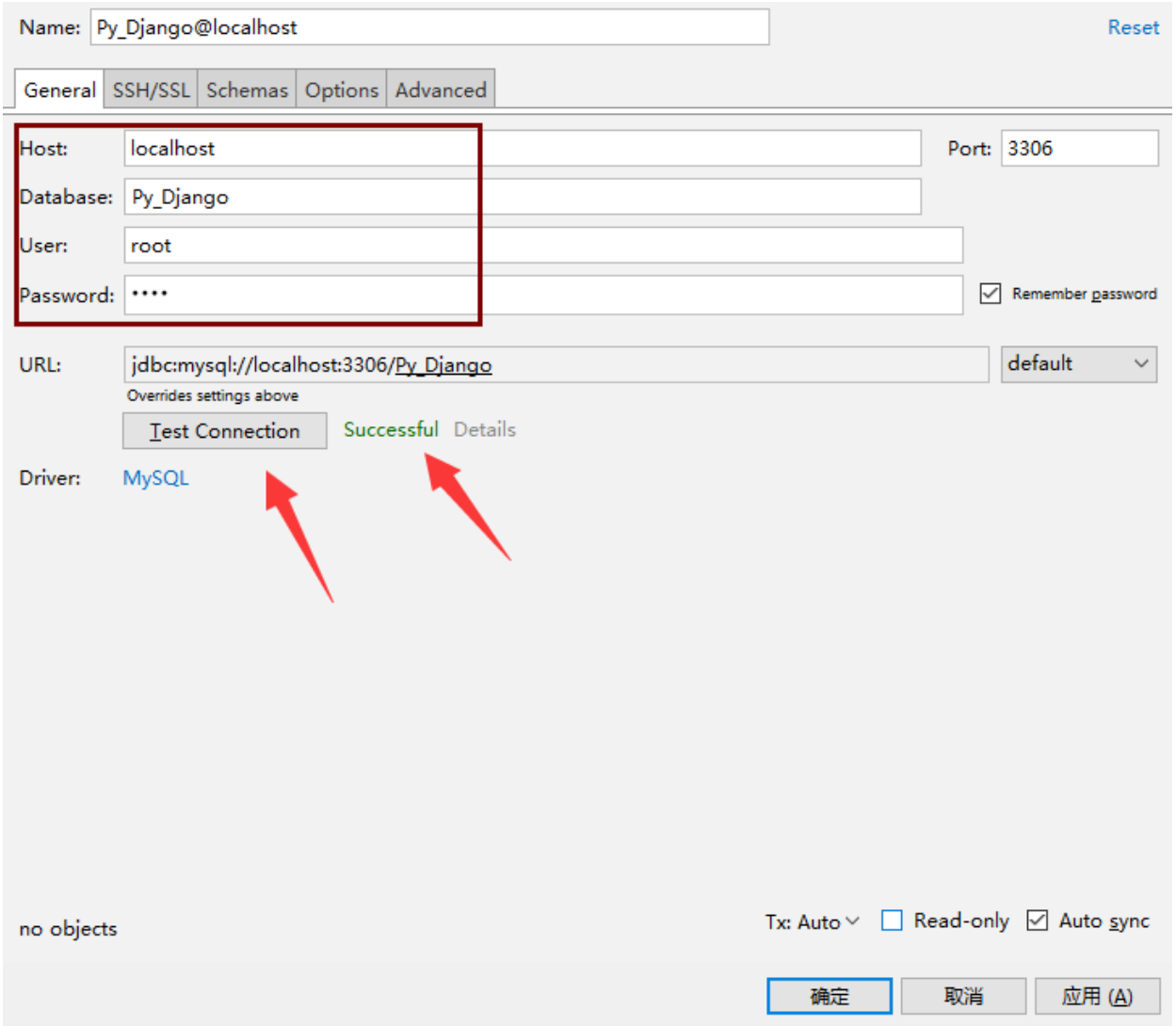


依次点击，选择 MySQL 数据库



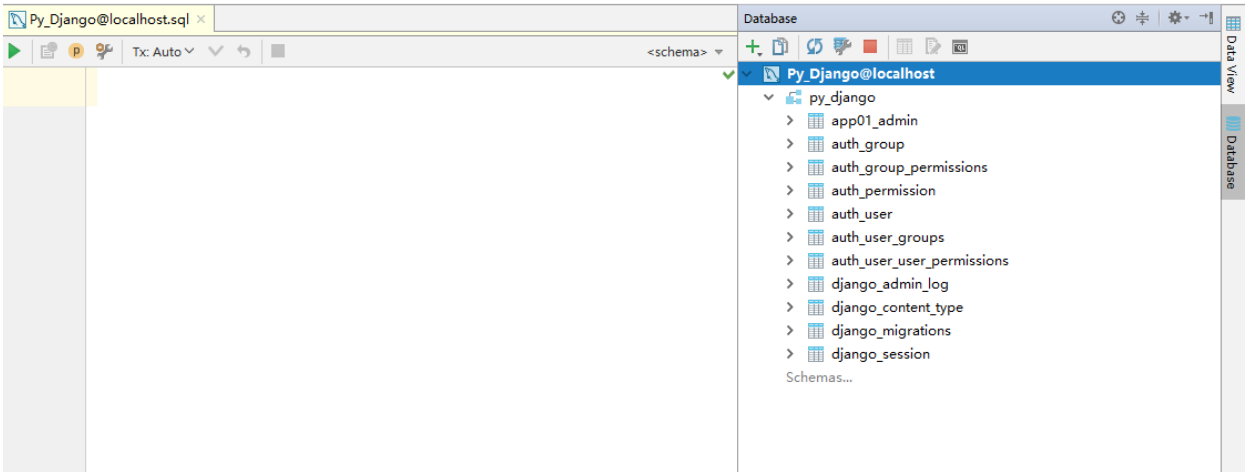
点击 Download 下载驱动文件

下载完成后对数据库的相关信息进行填写



填写完成后点击“Test Connection”，如果出现 Successful 就说明连接成功

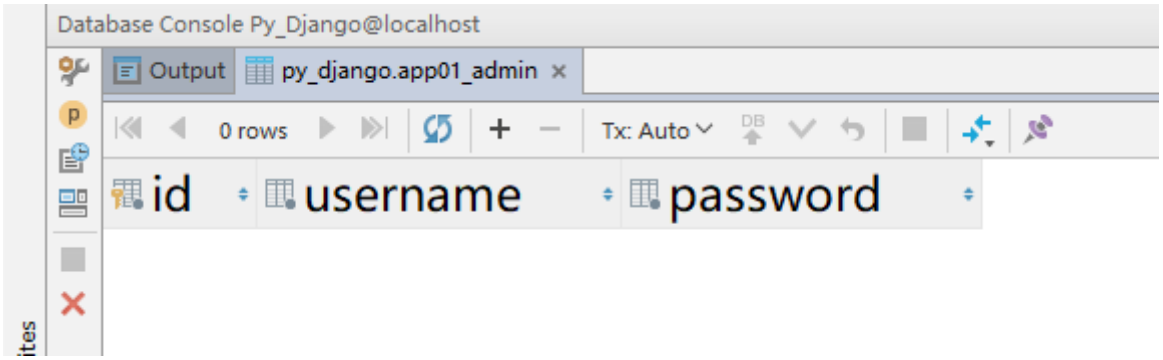
然后点击“应用”，再点击“确定”



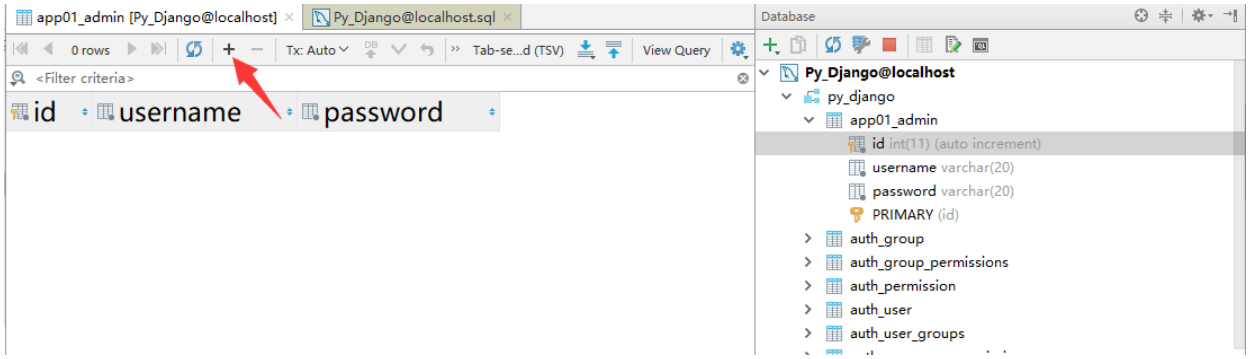
左边这个窗口是写 SQL 语句的地方

例如查询 app01_admin 表

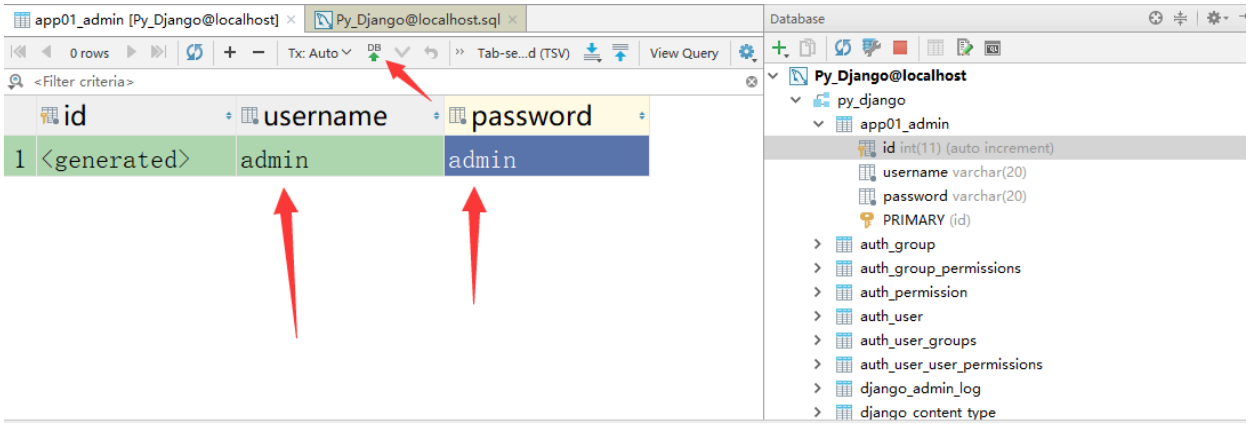
点击这个绿色三角形执行



如果要添加数据的话

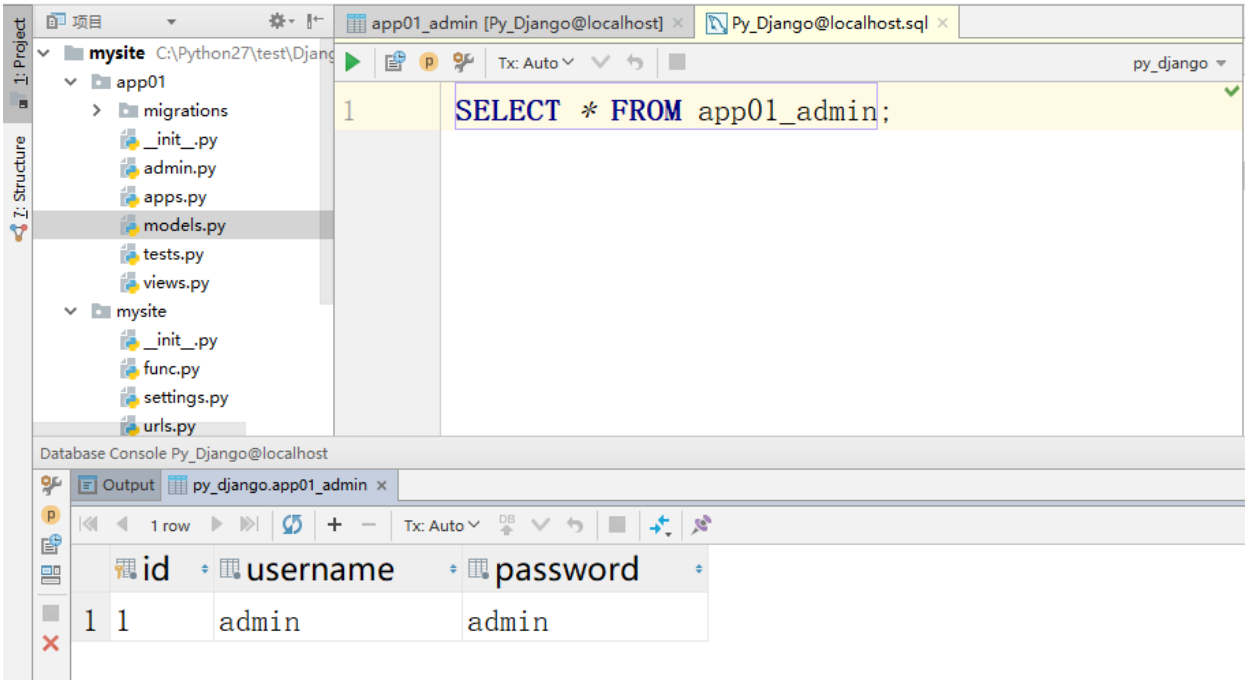


点这个加号

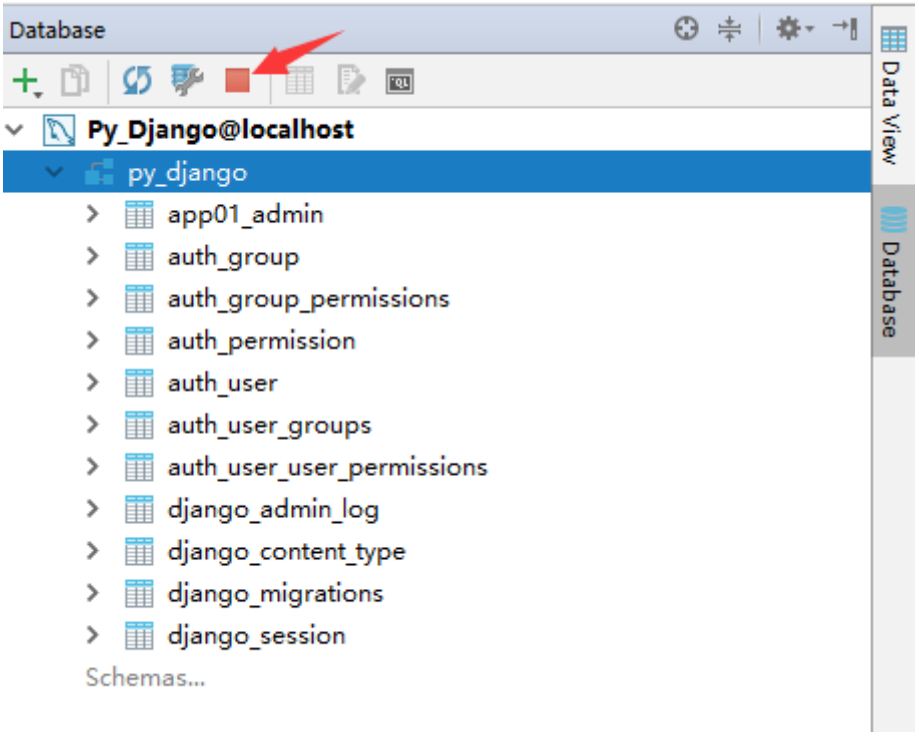


填写内容，填写完之后，点击上面那个绿色的箭头，更新到数据库中

再次查询



如果要关闭连接的话，点击红色正方形的按钮即可



PyMySQL

Table of Contents

This package contains a pure-Python MySQL client library, based on PEP 249.

Most public APIs are compatible with mysqlclient and MySQLdb.

NOTE: PyMySQL doesn't support low level APIs *_mysql* provides like *data_seek*, *store_result*, and *use_result*. You should use high level APIs defined in PEP 249. But some APIs like *autocommit* and *ping* are supported because PEP 249 doesn't cover their usecase.

Requirements

- Python -- one of the following:
  - CPython : 3.6 and newer
  - PyPy : Latest 3.x version
- MySQL Server -- one of the following:
  - MySQL >= 5.6
  - MariaDB >= 10.0

Installation

Package is uploaded on PyPI.

You can install it with pip:

```
$ python3 -m pip install PyMySQL
```

To use "sha256_password" or "caching_sha2_password" for authenticate, you need to install additional dependency:

```
$ python3 -m pip install PyMySQL[rsa]
```

To use MariaDB's "ed25519" authentication method, you need to install additional dependency:

```
$ python3 -m pip install PyMySQL[ed25519]
```

Documentation

Documentation is available online: https://pymysql.readthedocs.io/

For support, please refer to the StackOverflow.

Example

The following examples make use of a simple table

```
CREATE TABLE `users` (
    `id` int(11) NOT NULL AUTO_INCREMENT,
    `email` varchar(255) COLLATE utf8_bin NOT NULL,
    `password` varchar(255) COLLATE utf8_bin NOT NULL,
    PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
AUTO_INCREMENT=1 ;
```

```
import pymysql.cursors

# Connect to the database
connection = pymysql.connect(host='localhost',
                             user='user',
                             password='passwd',
                             database='db',
                             cursorclass=pymysql.cursors.DictCursor)

with connection:
    with connection.cursor() as cursor:
        # Create a new record
        sql = "INSERT INTO `users` (`email`, `password`) VALUES (%s, %s)"
        cursor.execute(sql, ('webmaster@python.org', 'very-secret'))

    # connection is not autocommit by default. So you must commit to save
    # your changes.
    connection.commit()

    with connection.cursor() as cursor:
        # Read a single record
        sql = "SELECT `id`, `password` FROM `users` WHERE `email`=%s"
        cursor.execute(sql, ('webmaster@python.org',))
        result = cursor.fetchone()
        print(result)
```

This example will print:

```
{'password': 'very-secret', 'id': 1}
```

Resources

- DB-API 2.0: https://www.python.org/dev/peps/pep-0249/
- MySQL Reference Manuals: https://dev.mysql.com/doc/
- MySQL client/server

protocol: https://dev.mysql.com/doc/internals/en/client-server-protocol.html

- "Connector" channel in MySQL Community

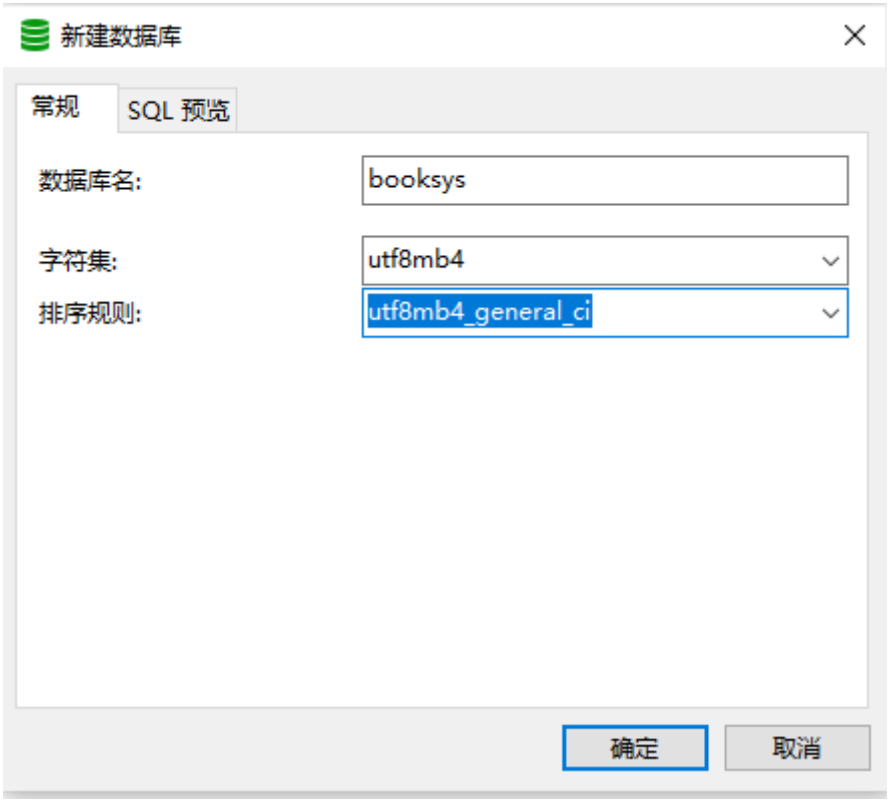
Slack: https://lefred.be/mysql-community-on-slack/

- PyMySQL mailing

list: https://groups.google.com/forum/#!forum/pymysql-users

License

一、 创建数据库



二、环境搭建

1. Django默认数据库

```
76
77  DATABASES = {
78      'default': {
79          'ENGINE': 'django.db.backends.sqlite3',
80          'NAME': BASE_DIR / 'db.sqlite3',
81      }
82  }
```

2. 将Django的数据库更改为mysql

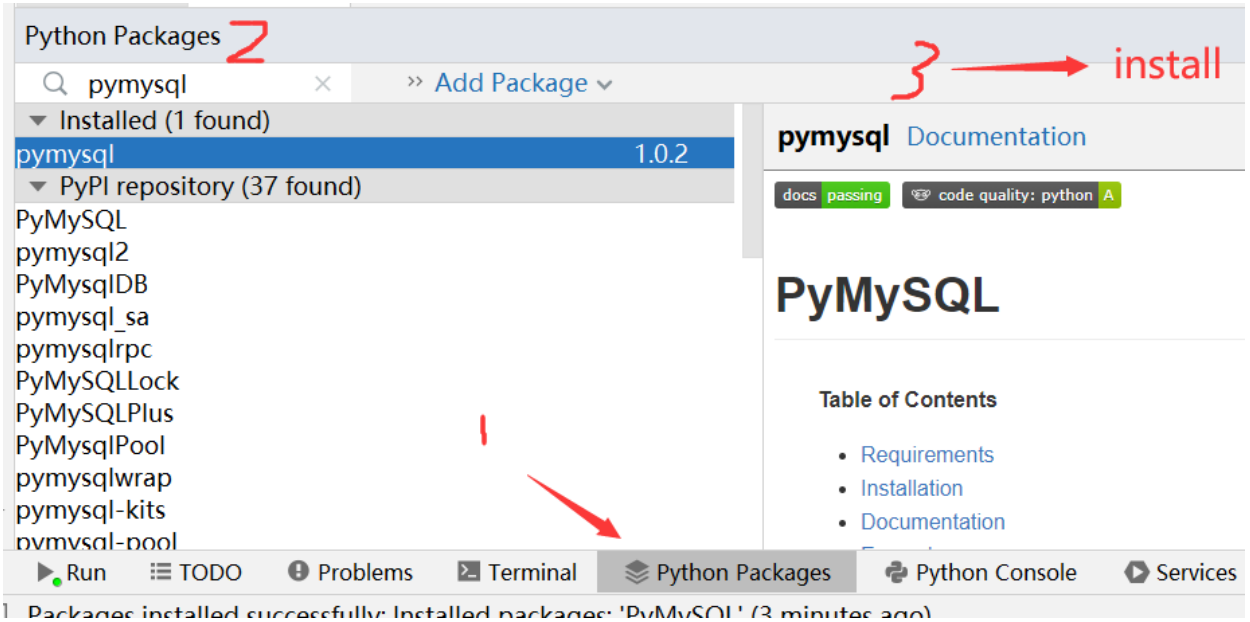
2.1 第一步:更改setting.py中的DATABASES

```
1  # 配置数据库
2  DATABASES = {
3   'default': {
4   # python自带的一个数据库，基本不会被使用
5   # 'ENGINE': 'django.db.backends.sqlite3',
```

```
    # 'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    # 注册我们自己使用的数据库连接
    'ENGINE': 'django.db.backends.mysql', # 数据库引擎
    'NAME': 'booksys', #数据库名称
     'USER':'root', # 连接数据库的用户名称
     'PASSWORD':'admin', # 用户密码
     'HOST':'127.0.0.1', # 访问的数据库的主机的ip地址
     'PORT':'3306', # 默认mysql访问端口
     }
}
```

第二步在python控制台输入:pip install pymysql安装pymysql

```
projectOne>pip install pymysql
Collecting pymysql
  Downloading https://files.pythonhosted
     |████████████████████████████████|
Installing collected packages: pymysql
Successfully installed pymysql-0.10.1
```



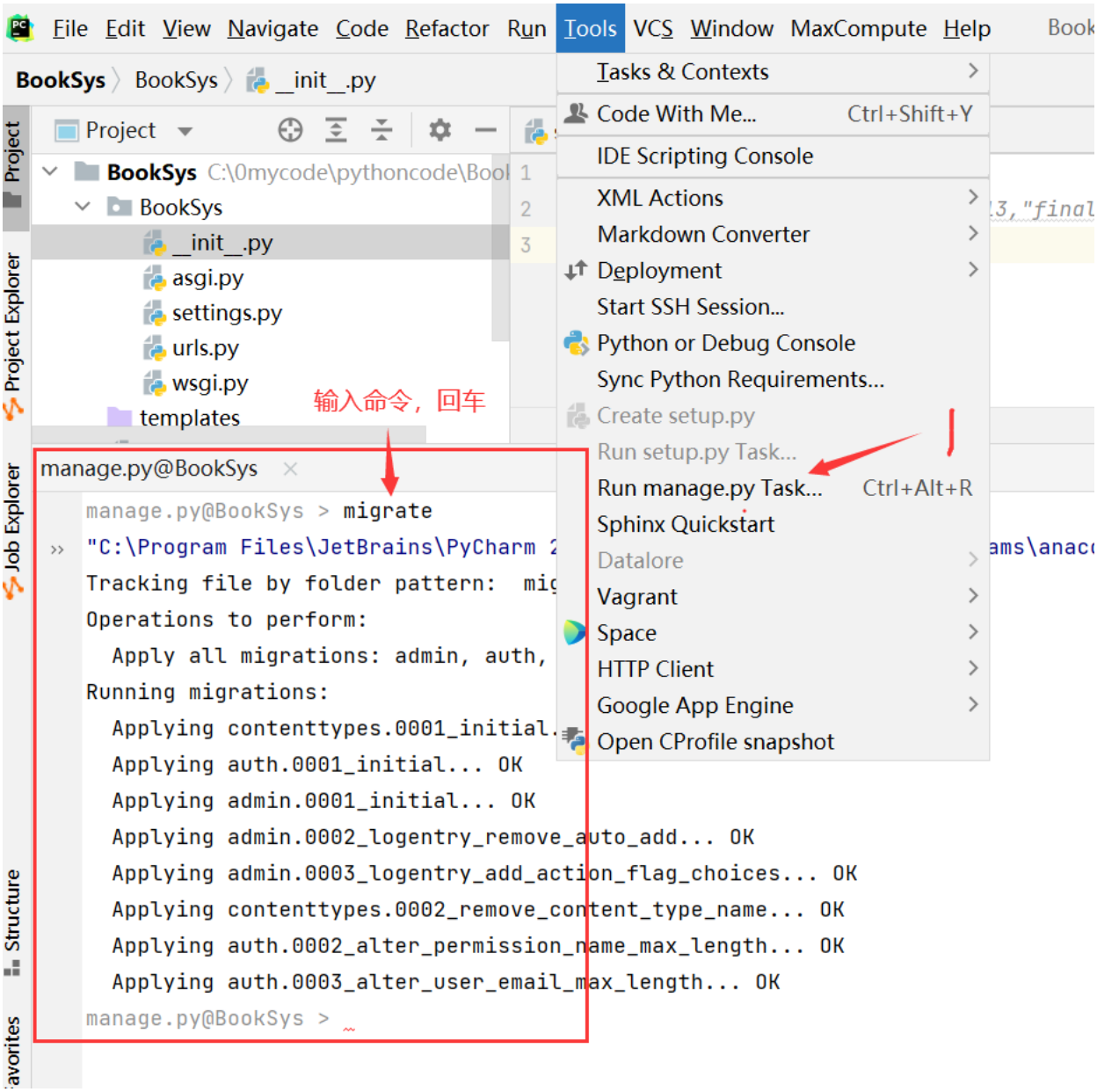
第三步:在项目中的__init__ .py添加下面代码

因为我出现了下面问题↓所以添加了pymysql.version_info = (1,4,13,“final”,0)

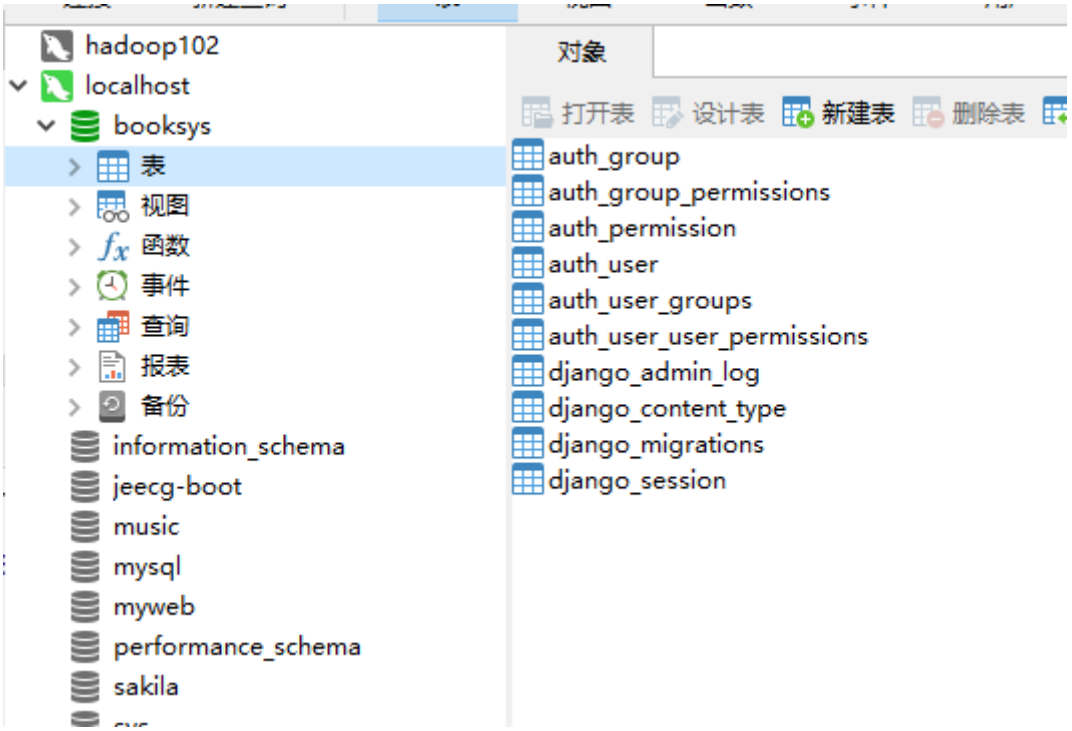
```
    raise ImproperlyConfigured('mysqlclient 1.4.0 or newer is required; you have %s.' % Database.__version__)
django.core.exceptions.ImproperlyConfigured: mysqlclient 1.4.0 or newer is required; you have 0.10.1.
```

```python
import pymysql
pymysql.version_info = (1,4,13,"final",0)
pymysql.install_as_MySQLdb()
```

最后就可以执行程序 python manage.py migrate，将会同步Django相关数据到mysql



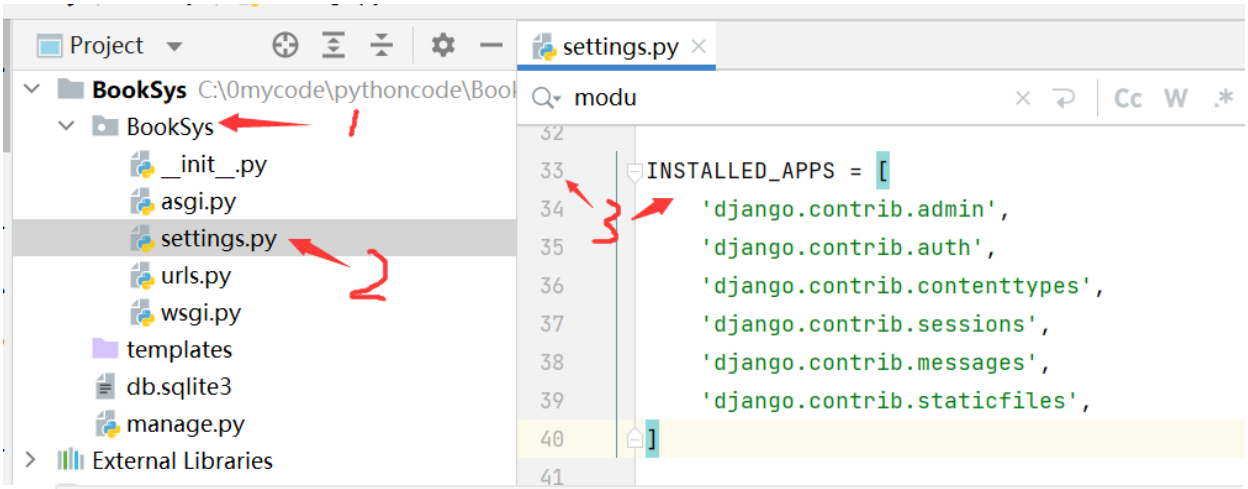
执行完毕之后，打开navicat,刷新booksys数据库，发现已经有如下数据表了


hadoop102
localhost
booksys
表
视图
函数
事件
查询
报表
备份
information_schema
jeecg-boot
music
mysql
myweb
performance_schema
sakila
对象
打开表
设计表
新建表
删除表
auth_group
auth_group_permissions
auth_permission
auth_user
auth_user_groups
auth_user_user_permissions
django_admin_log
django_content_type
django_migrations
django_session

一、 Django项目默认应用列表

在BookSys项目中的二级目录BookSys文件夹下的settings.py中的文件中的

INSTALLED_APPS 内容



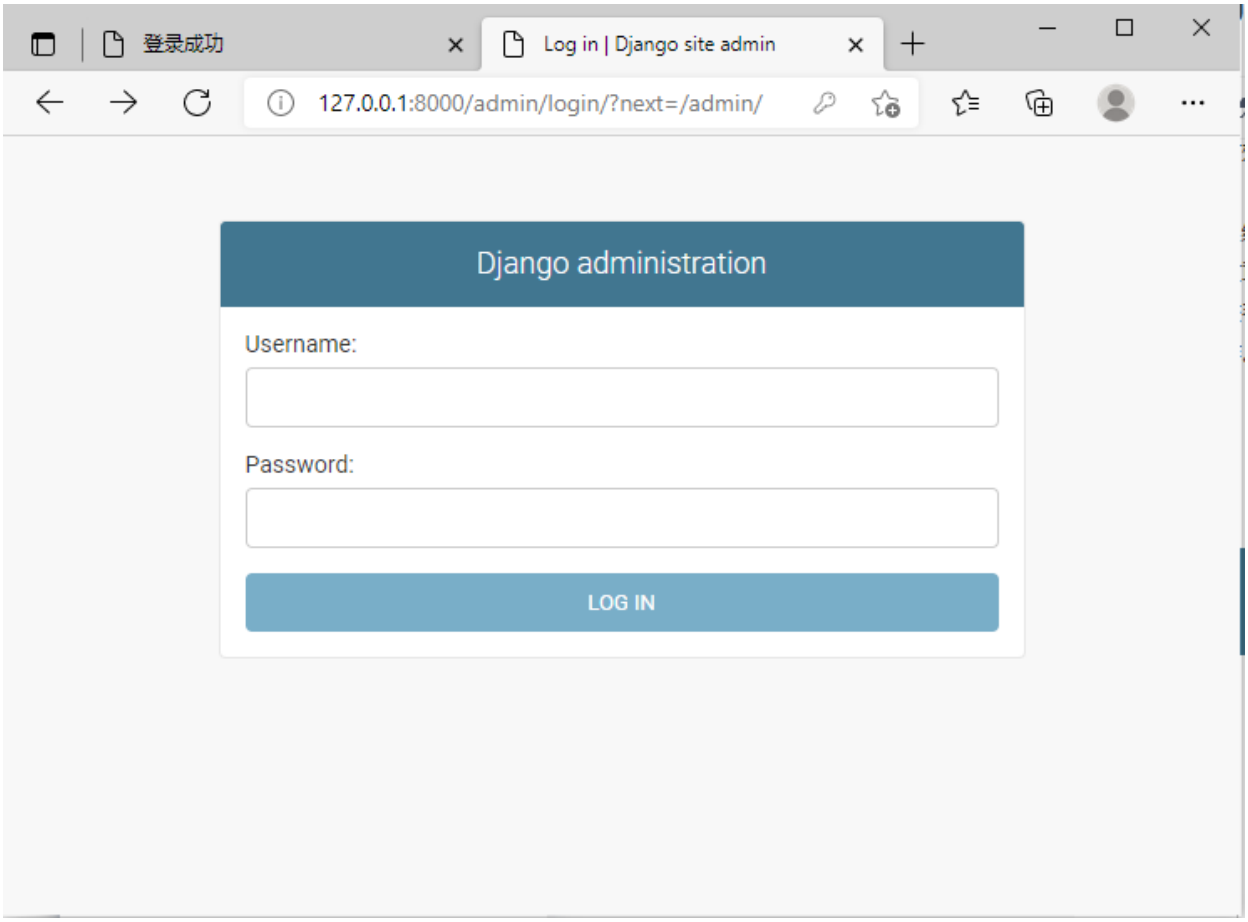

```
INSTALLED_APPS = [
 'django.contrib.admin',
 'django.contrib.auth',
 'django.contrib.contenttypes',
 'django.contrib.sessions',
 'django.contrib.messages',
 'django.contrib.staticfiles',
]
```

二、 admin应用

1. admin应用简介：

后台管理系统主要是对数据表的存储做专门的管理，例如针对微博或者论坛类的站点，管理员需要删除不合规的文章，或者公司内部需要发布新的话题等，这些都是通过数据表的管理实现的。单一功能的后台系统比较容易构建，但是如果功能增多情况下，就需要对多个数据表做管理，这就增加了开发人员的重复性工作。Django 提供的后台管理系统很好的解决了这个问题

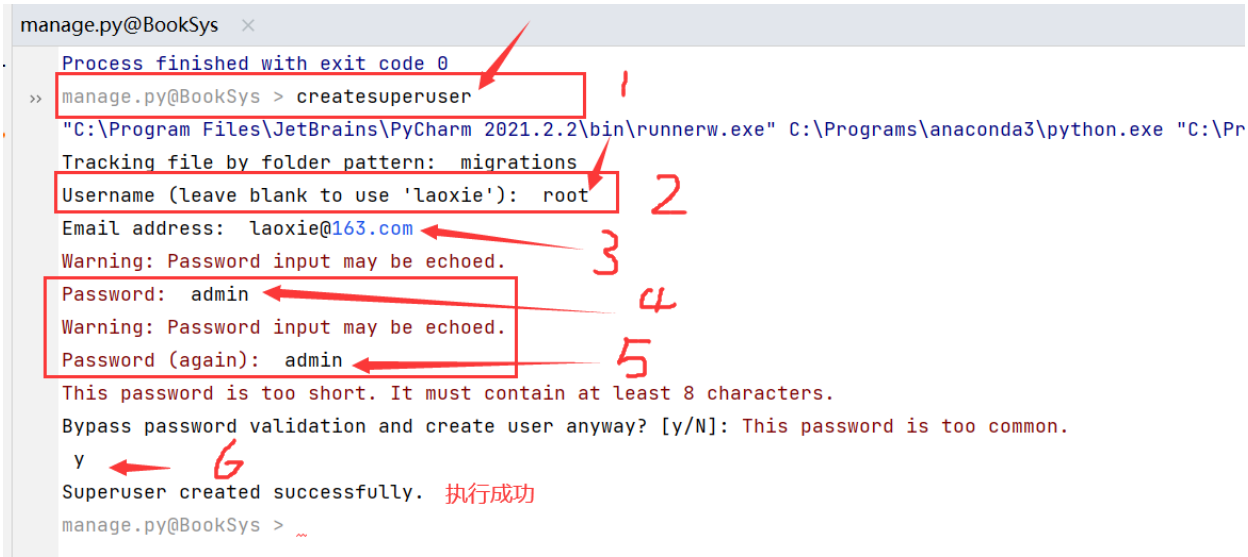
Django默认的后台地址：http://127.0.0.1:8080/admin,效果如下：



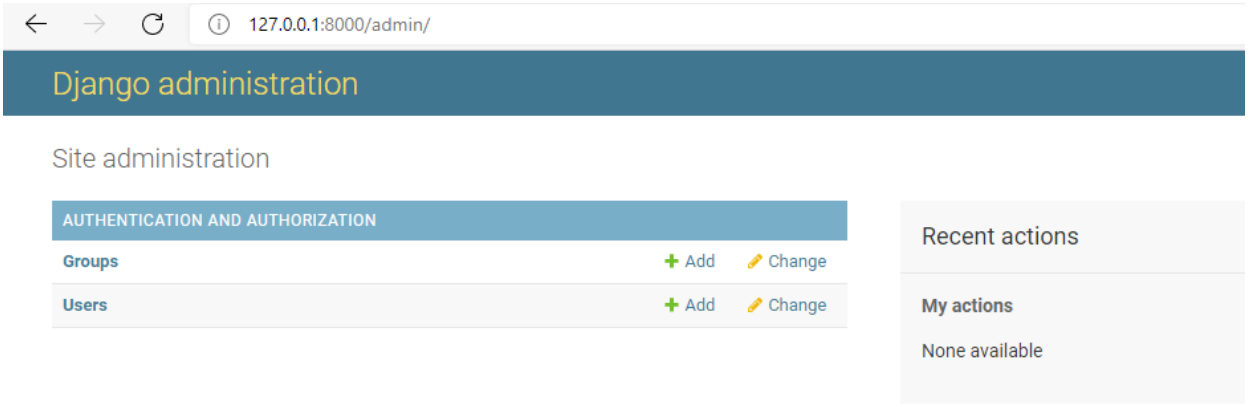
2. 创建用户

创建用户

```
createsuperuser
```



3. 在浏览器中访问http://127.0.0.1:8000/admin/



4. 后台功能简介

Admin 后台管理系统提供了用户类别、用户权限以及用户组权限的划分功能，如图3所示，Active (有效）、Staff status（人员状态）、Superuser status（超级用户状态）是用户的三种类别。

图3：Admin后台用户类别设置

如图4所示，Django 后台管理系统提供了用户权限划分功能。

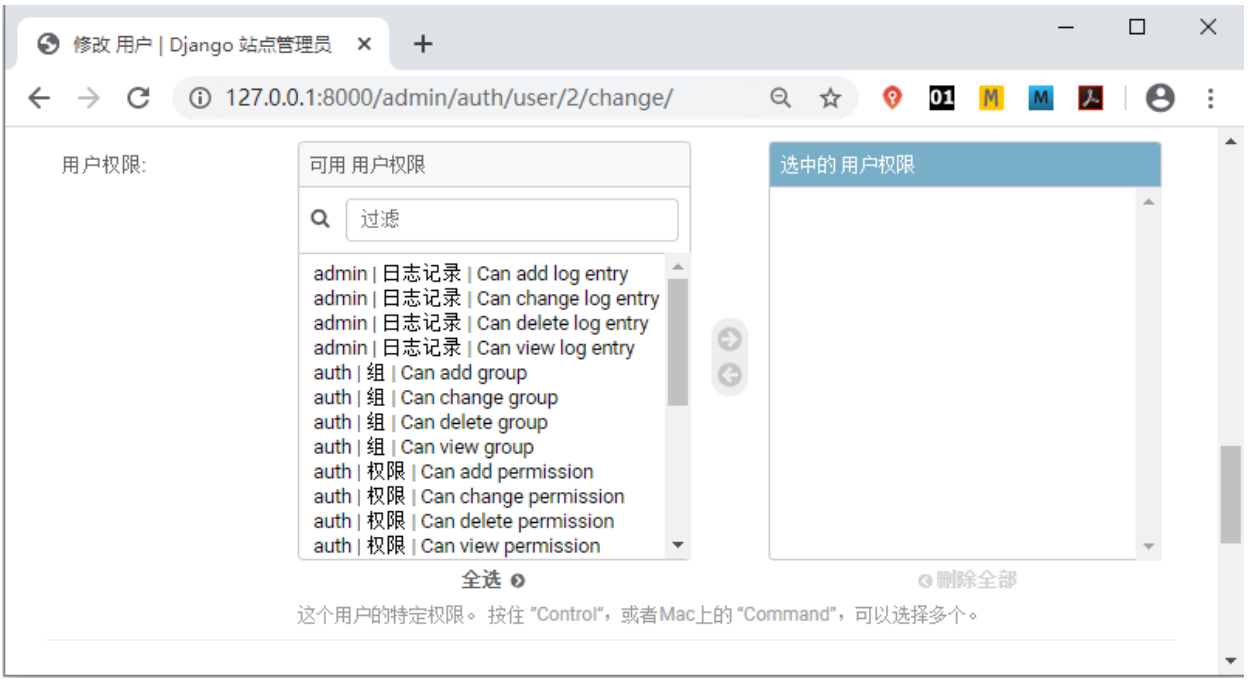


图4：用户权限划分

当然也提供了用户组权限的分配功能，图5、6所示，分别是添加组和分配组权限。

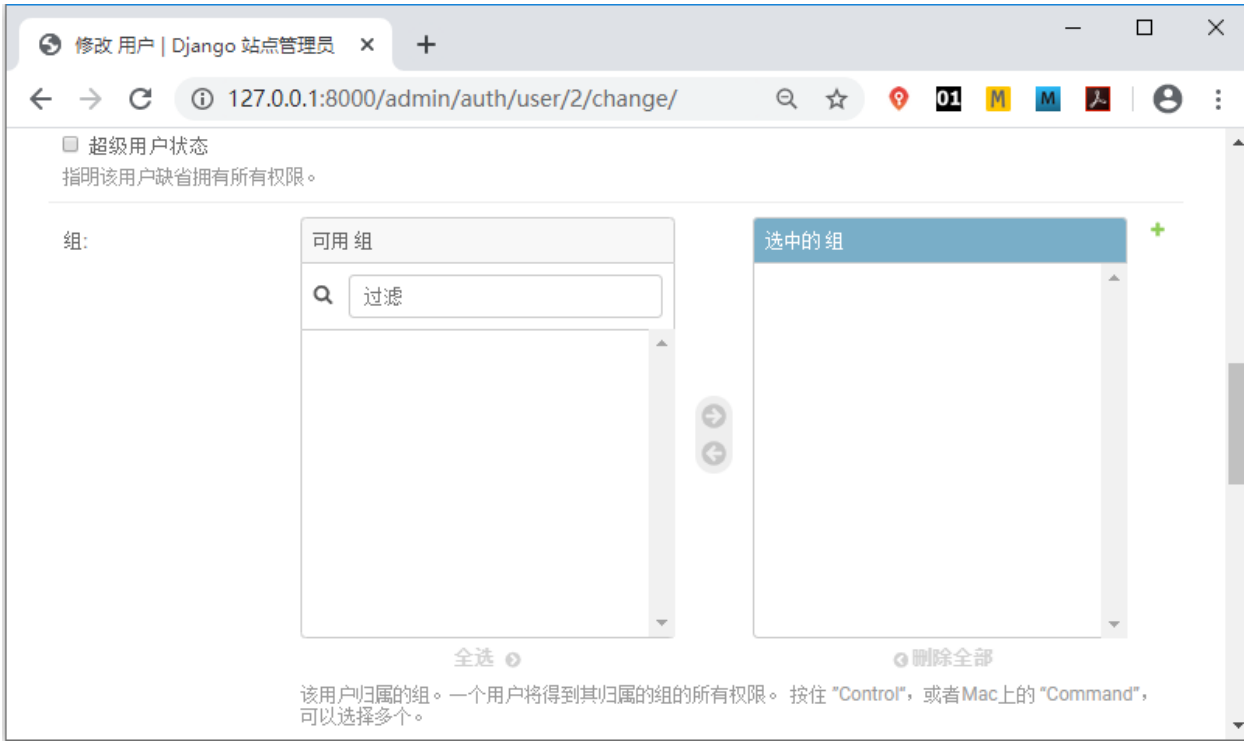


图5：用户组权限划分

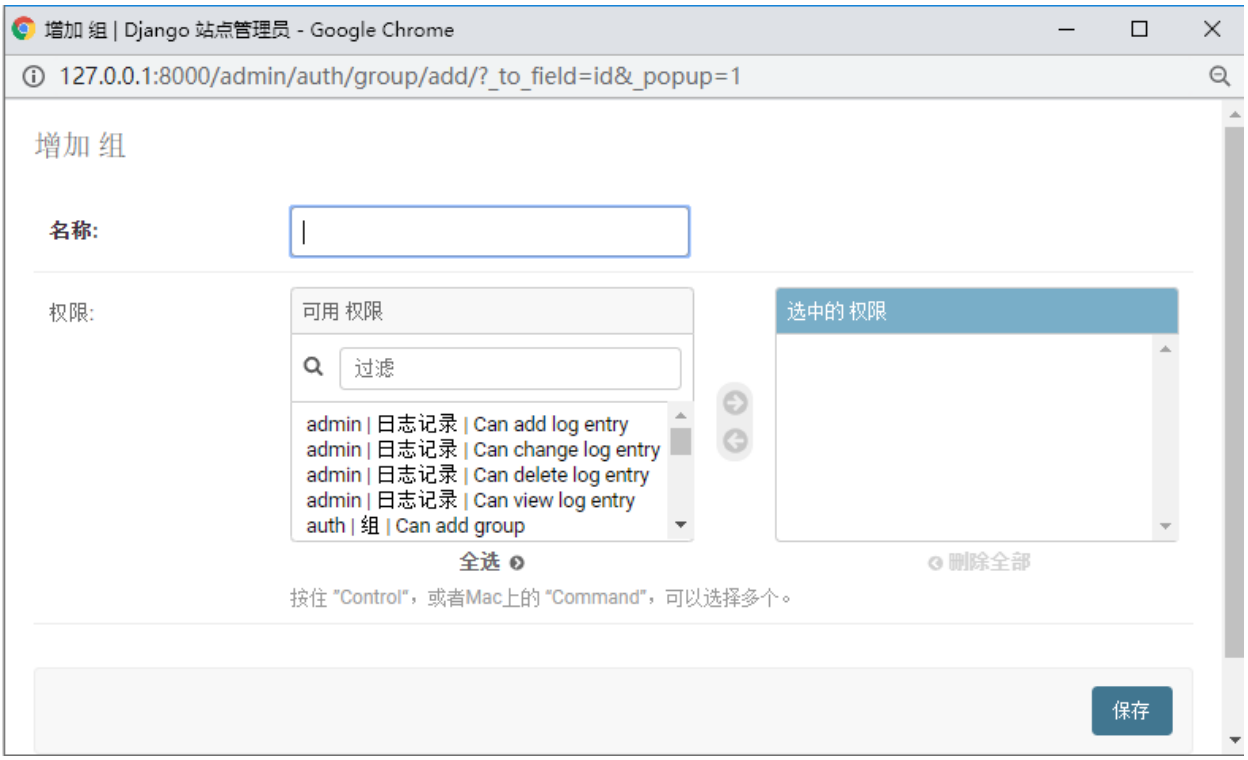


图6：新添加用户组权限

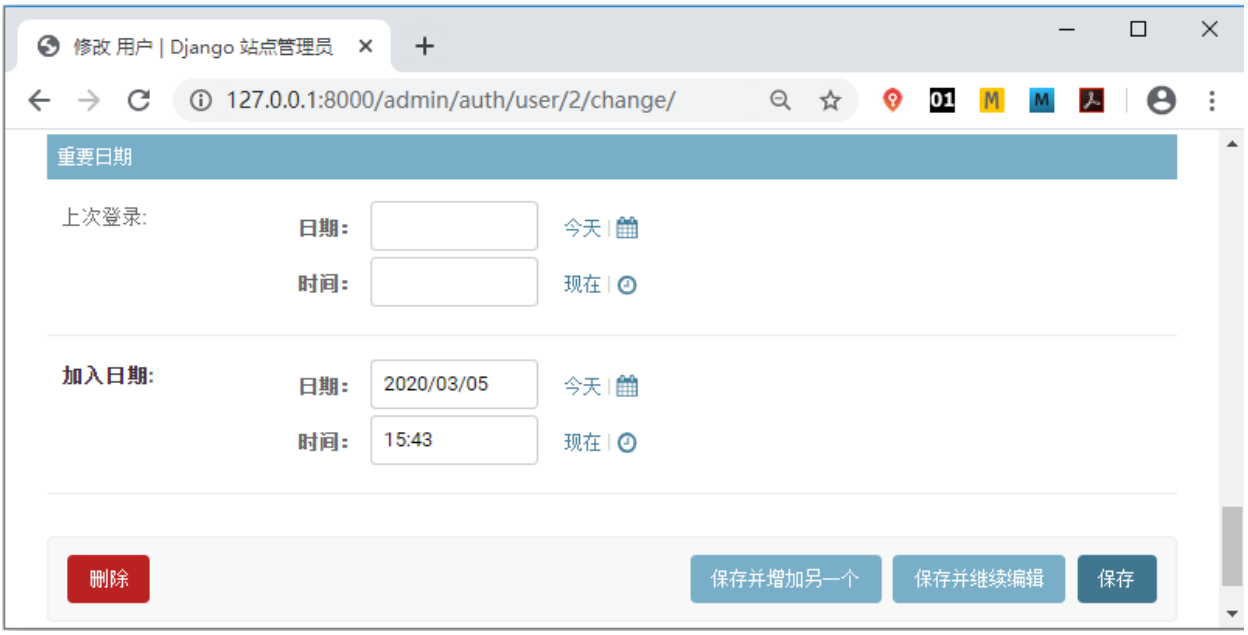


图7：时间记录

Django 原生的 Admin 管理界面，虽然没有做太多的修饰，但是开发者可以利用 Model Admin 实现个性化定制，比如字段值的过滤功能、表字段展示的排序、搜索功能等。上面展示的是中文的后台管理界面，原生的 Django 后台管理系统是英文的。所以需要简单设置才可以实现中文模式。

三、 auth应用

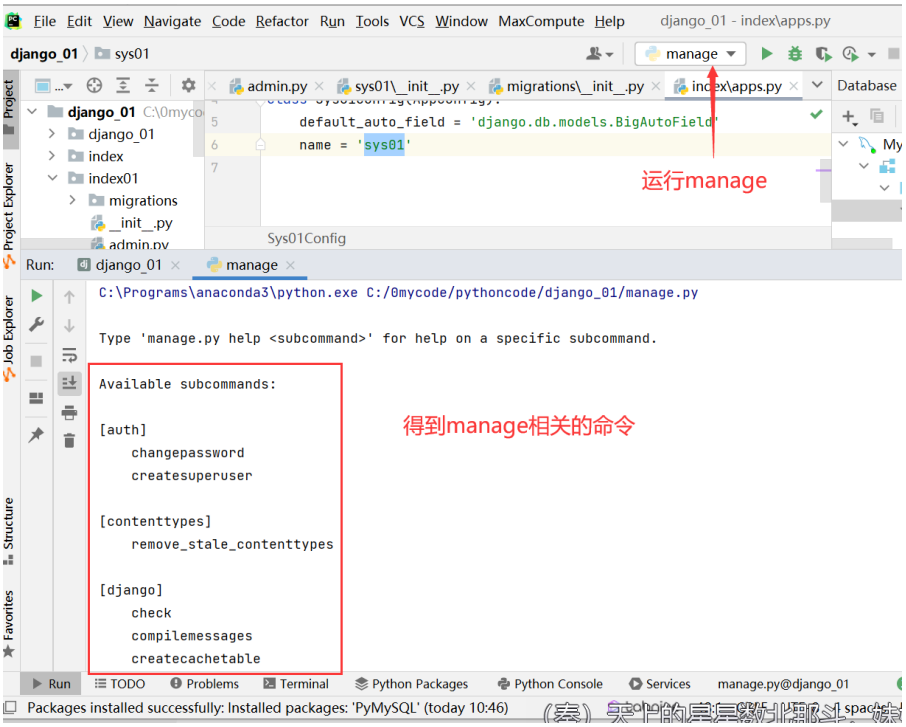
四、contenttypes

五、 sessions

六、messages

七、staticfiles

一、manage命令列表



Type 'manage.py help <subcommand>' for help on a specific subcommand.

Available subcommands:

[auth]

changepassword

createsuperuser

[contenttypes]

remove_stale_contenttypes

[django]

check

compilemessages

createcachetable

dbshell

diffsettings

dumpdata

flush

inspectdb

loaddata

makemessages
makemigrations
migrate
sendtestemail
shell
showmigrations
sqlflush
sqlmigrate
sqlsequencereset
squashmigrations
startapp
startproject
test
testserver

[sessions]
clearsessions

[staticfiles]
collectstatic
findstatic
runserver

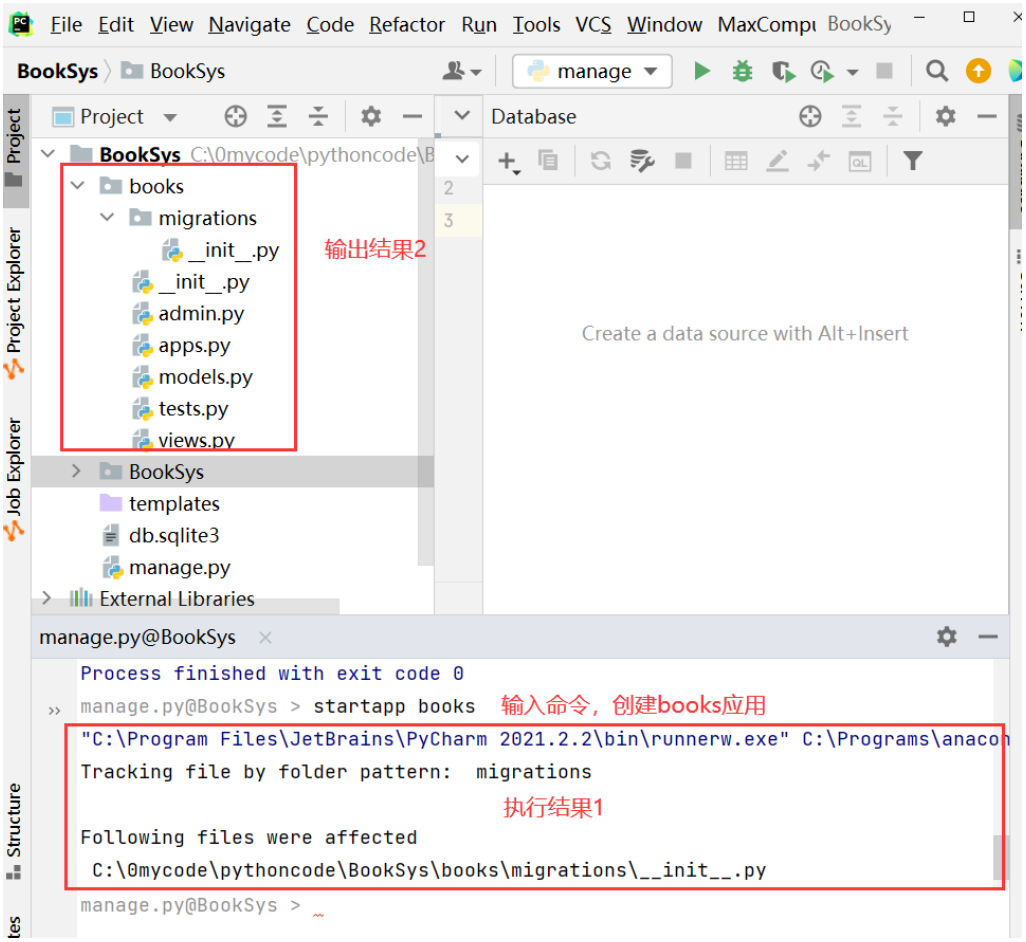
二、使用shell命令创建应用

1. 使用命令创建应用

PyCharm IDE 则选择Tools->Run manage.py Task..

底部弹出shell框 输入startapp 应用名　创建应用

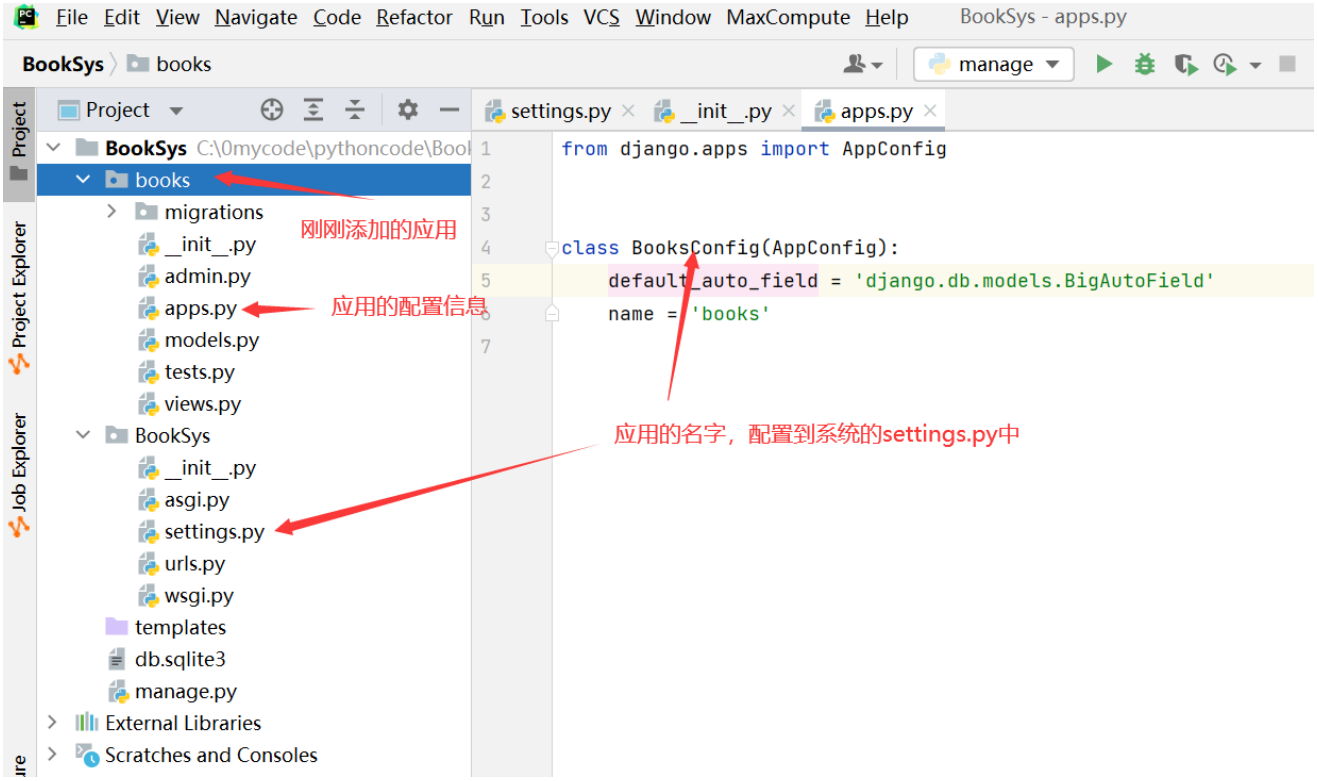
比如创建books应用　startapp books 回车，当命令执行完毕之后，会自动在项目代码中创建books相关的代码

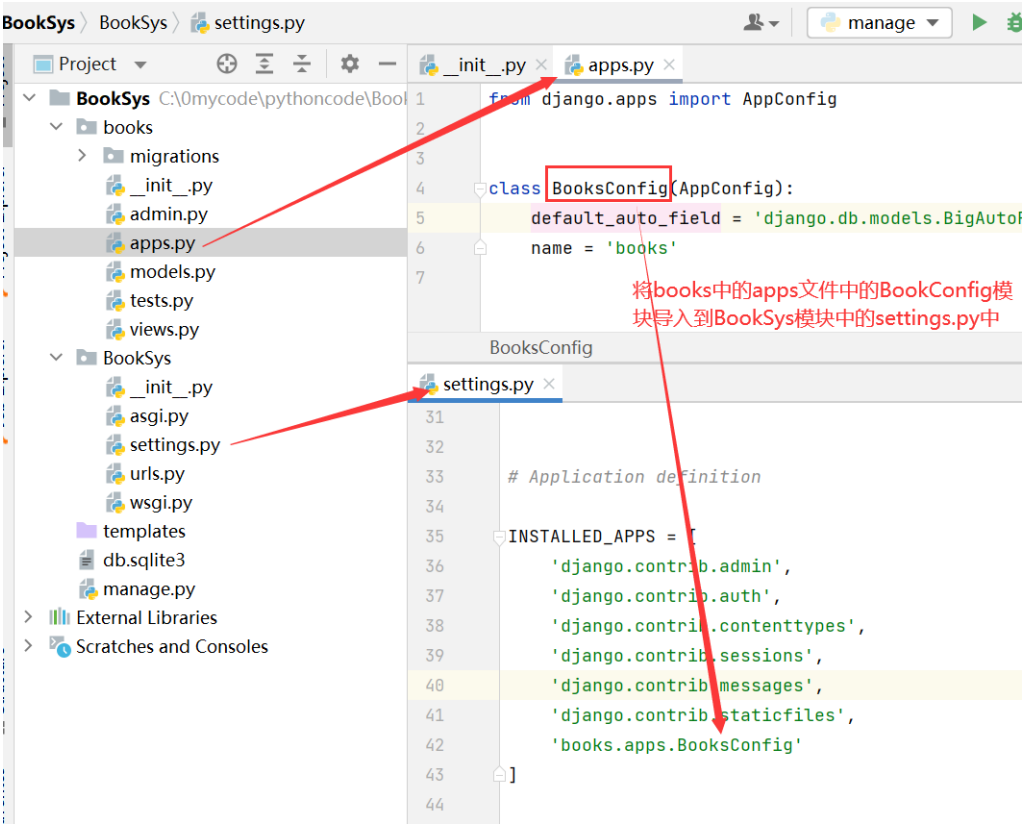


2. 应用相关配置文件说明：

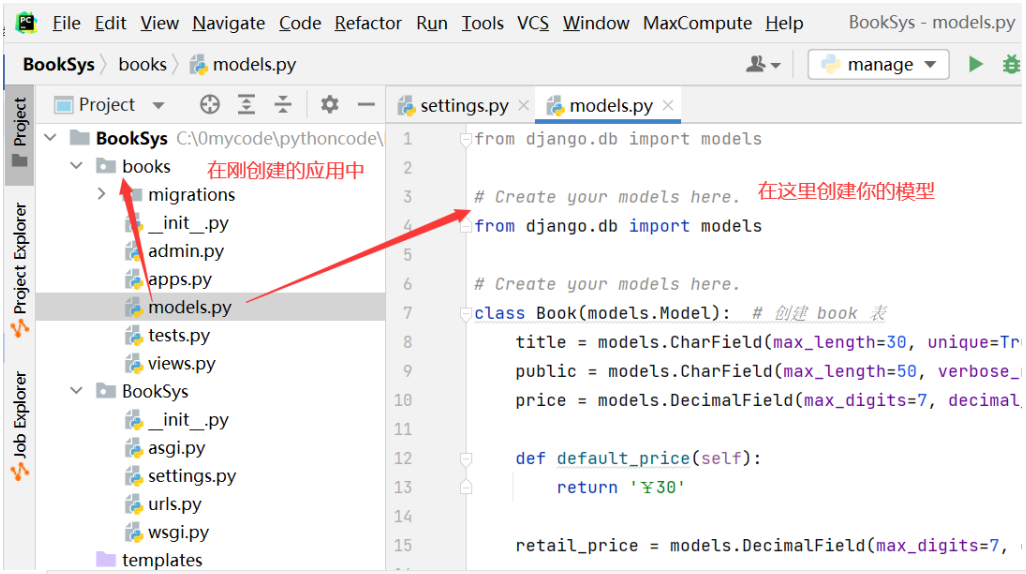
3. 在配置文件中配置新添加的模块

将应用的名字复制到





四、 在models.py中添加模型类




```python
from django.db import models

# Create your models here.
class Book(models.Model): #创建 book 表
 title=models.CharField( max_length=30,unique=True, verbose_name='书名')
 public=models.CharField(max_length=50,verbose_name='出版社')
 price=models.DecimalField(max_digits=7,decimal_places=2,verbose_name='定价')
 def default_price(self):
 return '¥30'
 retail_price=models.DecimalField(max_digits=7,decimal_places=2,verbose_name='零售价',default=default_price)
 def __str__(self):
 return "title:%s pub:%s price:%s" % (self.title, self.public, self.price)

class Author(models.Model): #创建作者表
 name=models.CharField(max_length=30,verbose_name='姓名')
 email=models.EmailField(verbose_name='邮箱')
 def __str__(self):
 return '作者: %s'%(self.name)

class UserInfo(models.Model): #创建用户信息表
 username=models.CharField(max_length=24,verbose_name='用户注册')
 password =models.CharField(max_length=24,verbose_name='密码')
```

Django 把表模型定义为 Model，他需要继承自django.db.models中的 Model 类，只要是与数据表相关的操作，都需要继承这个类。同时ORM 对于数据库的的增删改查，也提供了一些简单的 API，例如 F 查询、Q 查询。

针对数据库中的字段类型，Django ORM 都有对应的 “xxxField” 来表述，见如下表格。

字段	说明	字段属性
AutoFiled	默然自增主键（Primary_key=Ture），Django 默认建立id字段为主键。	
CharFiled	字符类型	Max_length=32，字符长度需要明确

IntgerFiled	整型 int	
DateFiled	年月日时间类型	auto_now=True，数据被更新就会更新时间； auto_now_add=True，数据第一次参数时产生。
DateTimeFiled	年月日小时分钟秒时间类型	auto_now=True，数据被更新就会更新时间; auto_now_add=True，数据第一次参数时产生。
DecimalFiled	混合精度的小数类型	max_digits=3，限定数字的最大位数(包含小数位)；decimal_places=2，限制小数的最大位数。
BooleanFiled	布尔字段，对应数据库tinyint 类型数据长度只有1位。	值为True或False
TextFiled	用于大文本	

4. 魔术方法__str__

__str__ 方法是 Python 中的 "魔术" 方法，它是为 print 这样的打印函数设计的，它属于 python 的 object 基类的一个方法，也就是说 python 所有的类都有该方法，当然 Django 的 modle 类也有。如果没有这个方法定义，打印对象会显示对象的内存地址，但是这样的显示方式不够友好，且不利于调试，而用 __str__ 方法后，可以在 print 时得到易于人阅读的信息，在如下所示：

```
# 直接print打印
class TestClass:
 def __init__(self):
 self.name = 'testcase'
t = TestClass() #实例化对象
print(t)              # 结果显示：<__main__.TestClass object at 0x8f5c27b42367>
# __str__方法
class TestClass:
 def __init__(self):
  self.name = '小明'
  def __str__(self):
  return self.name
t = TestClass() #实例化对象
print(t)               # 结果显示：小明
```

不仅如此 __str__ 方法在 admin 后台系统也发挥着巨大的作用，它会将函数的返回值作为对象来显示。

五、 将模型同步到数据库

1. 将models.py中模型定义的Class生成数据库中的表的执行命令，使用命令 makemigrations books

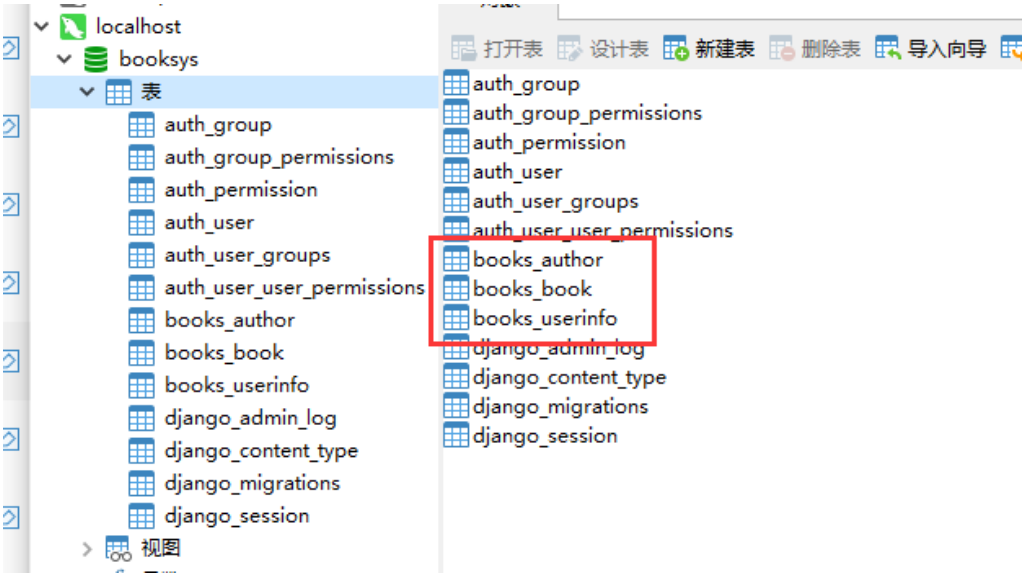
说明：books是应用名称

```
manage.py@BookSys
manage.py@BookSys > makemigrations books
"C:\Program Files\JetBrains\PyCharm 2021.2.2\bin\runnerw.ex
Tracking file by folder pattern:  migrations
Migrations for 'books':
  books\migrations\0001_initial.py
    - Create model Author
    - Create model Book
    - Create model UserInfo
```

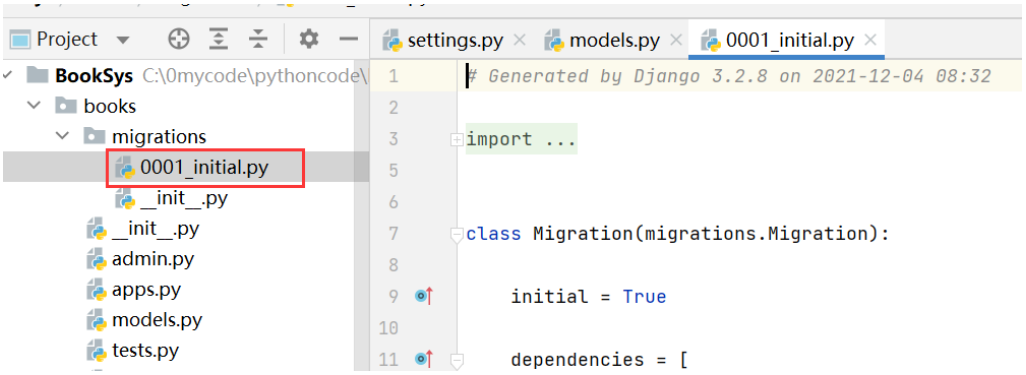
2. 使用migration books将上面生成的执行文件执行到数据库

```
manage.py@BookSys > migrate books
"C:\Program Files\JetBrains\PyCharm 2021.2.2\bin\runnerw.exe" C:\Programs\anaconda3\python.
Tracking file by folder pattern:  migrations
Operations to perform:
  Apply all migrations: books
Running migrations:
  Applying books.0001_initial... OK

Process finished with exit code 0
```

当执行完成后，打开navicat查看对应数据库中是否有生成表：



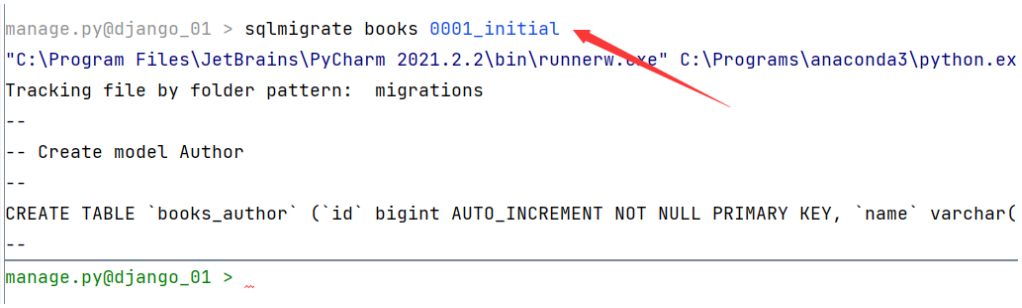
执行完这个操作后，books应用下的 migrations 目录下生还成了一个 0001_initial.py 的文件



这个迁移文件包含了创建数据表时用到的所有信息，这是一个临时的过度文件，在执行完 makemigrations 命令后生成

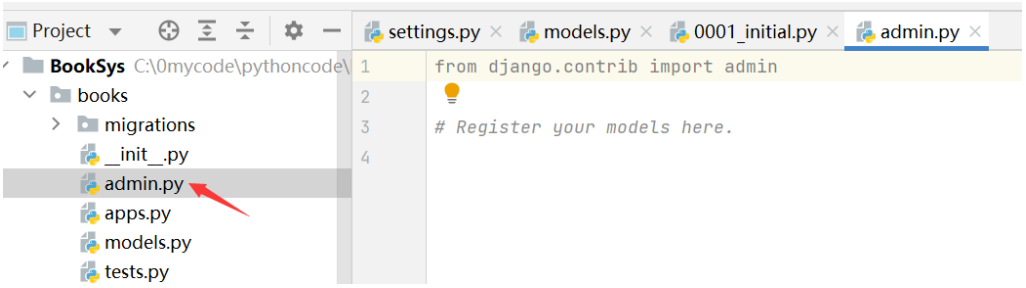
在执行完 makemigrations 命令后生成。我们可以使用如下命令打印迁移文件执行的 SQL 语句：

python manage.py sqlmigrate index 0001_initial

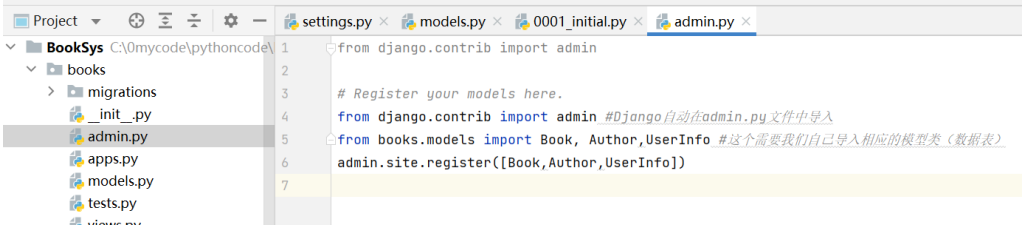


六、 将自定义模块注册到管理后台

当我们使用 startapp 命令创建 books应用的时候会自动创建 admin.py 文件，



想要把自定义的 Model 注册到管理后台，就需要在 admin.py 文件中进行声明，添加如下代码：



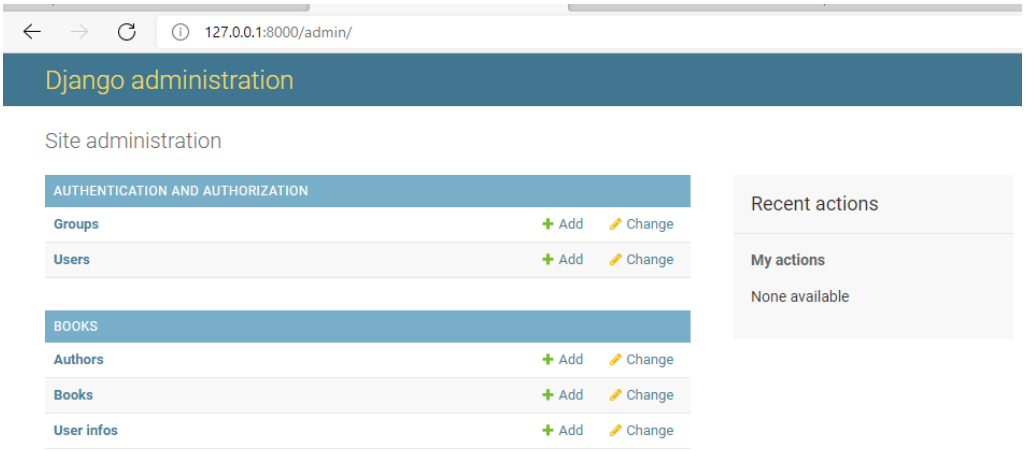

```
from django.contrib import admin #Django自动在admin.py文件中导入
from books.models import Book, Author,UserInfo #这个需要我们自己导入相应的模型类（数据表）
admin.site.register([Book,Author,UserInfo])
```

通过上述代码，我们就完成了将 Model 注册到后台管理系统的操作，其实实现的过程也非常的简单，首先通过 django.contrib 的标准库引入 admin 应用，然后把 index 应用下我们自定义的三张数据表引入，最后我们调

用 admin.site.register() 方法实现模型类的注册。多个模型类一起注册我们使用列表的形式来统一注册，如果是单一的模型类注册，我们可以使用以下方式即可：

```
admin.site.register(Book)
```

至此我们就完成了数据表在 Admin后台管理系统的可视化操作，我们再次使用ctrl+F5刷新后台管理系统的显示页面，可以得到如下结果：



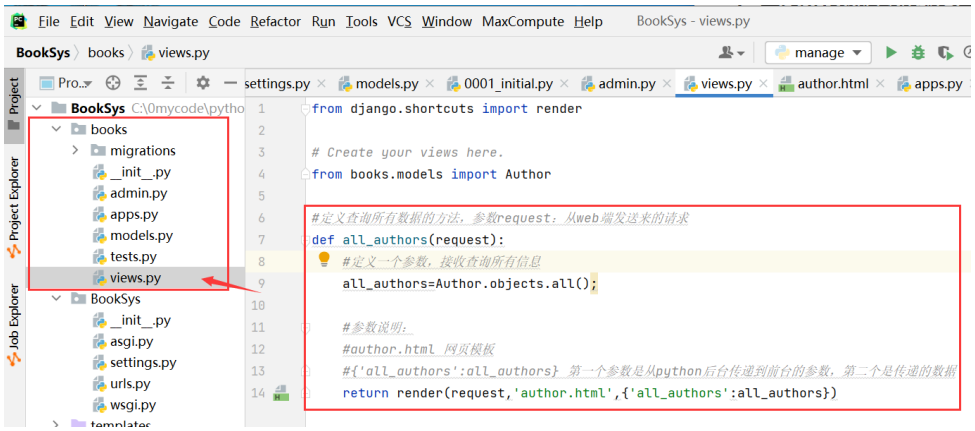
一、 查询所有数据到前端页面

1. 在数据库表中添加测试数据

在books_author表中添加如下数据

```
INSERT INTO `books_author`(`id`, `name`, `email`) VALUES (1, '张三', 'zhangsan@163.com');
INSERT INTO `books_author`(`id`, `name`, `email`) VALUES (2, '李四', 'lisi@163.com');
INSERT INTO `books_author`(`id`, `name`, `email`) VALUES (3, '王五', 'wangwu@163.com');
INSERT INTO `books_author`(`id`, `name`, `email`) VALUES (4, '赵六', 'zhaoliu@163.com');
INSERT INTO `books_author`(`id`, `name`, `email`) VALUES (5, '周七', 'zhouqi@163.com');
INSERT INTO `books_author`(`id`, `name`, `email`) VALUES (6, '王巴', 'wangba@163.com');
```

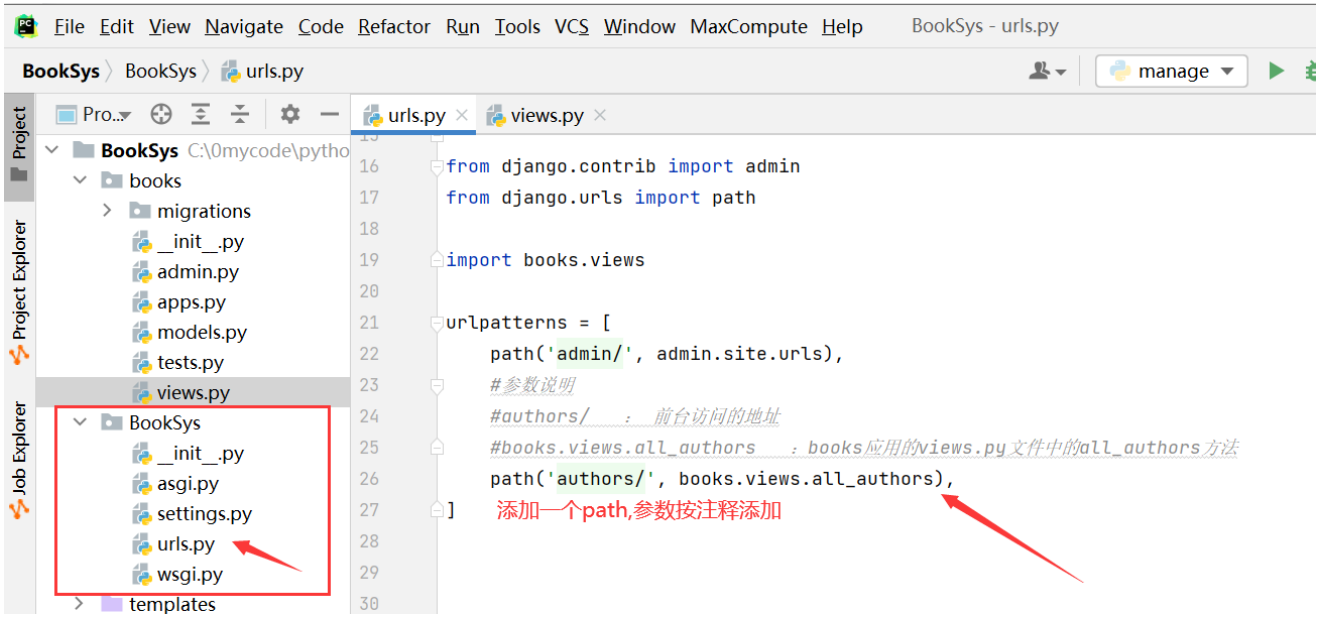
2. 在books应用的views.py中添加查询方法




```
from django.shortcuts import render

# Create your views here.
from books.models import Author

#定义查询所有数据的方法，参数request：从web端发送来的请求
def all_authors(request):
 #定义一个参数，接收查询所有信息
 all_authors=Author.objects.all();

 #参数说明：
 #author.html 网页模板
 #{'all_authors':all_authors} 第一个参数是从python后台传递到前台的参数，第二个是传递的数据
 return render(request,'author.html',{'all_authors':all_authors})
```

3. 添加地址映射

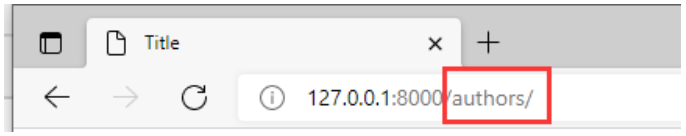
在项目的全局配置中的urls.py添加地址和方法进行映射



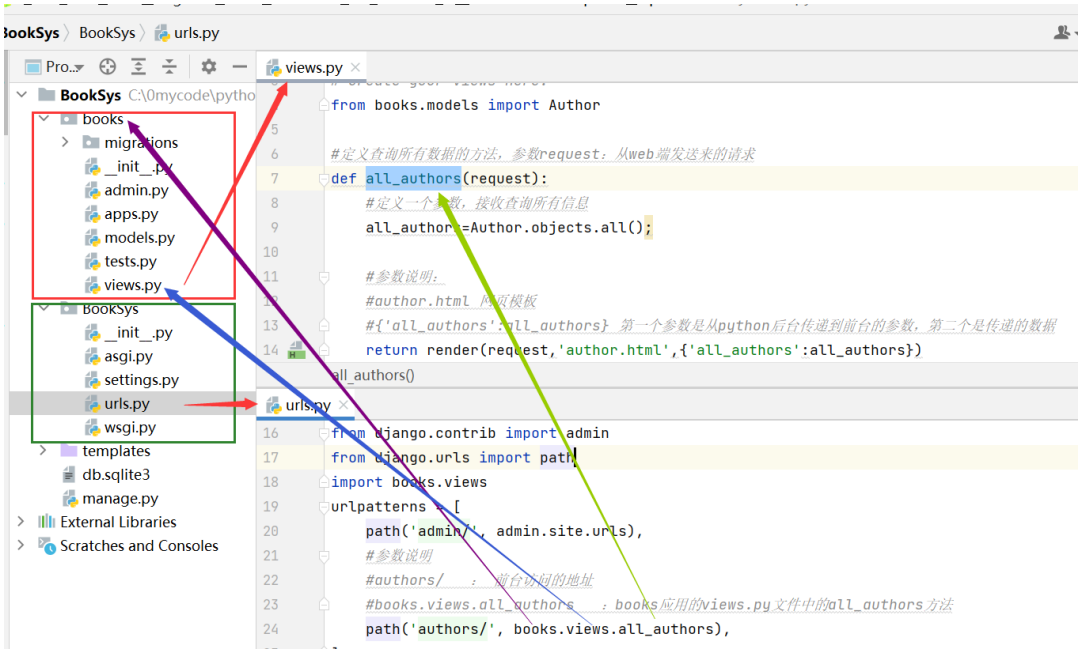

```python
from django.contrib import admin
from django.urls import path

import books.views

urlpatterns = [
 path('admin/', admin.site.urls),
 #参数说明
 #authors/ : 前台访问的地址
 #books.views.all_authors : books应用的views.py文件中的all_authors方法
 path('authors/', books.views.all_authors),
]
```

参数说明：

- authors/ 说明：前台访问的地址

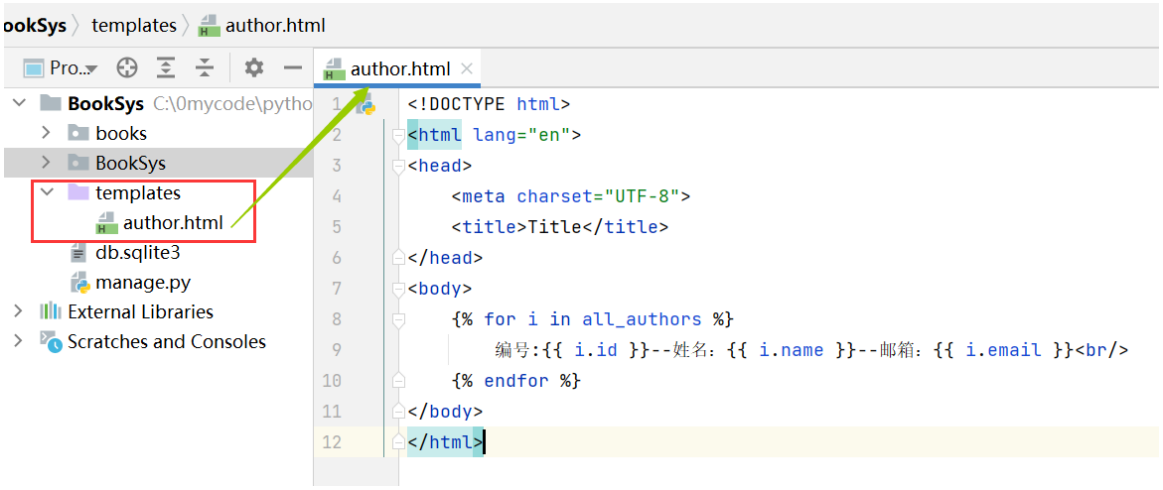


- books.views.all_authors：books应用的views.py文件中的all_authors方法



4. 模板添加

在templates中，添加author.html文档



模板代码

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Title</title>
</head>
<body>
 {% for i in all_authors %}
 编号:{{ i.id }}--姓名：{{ i.name }}--邮箱：{{ i.email }}<br/>
 {% endfor %}
</body>
</html>
```

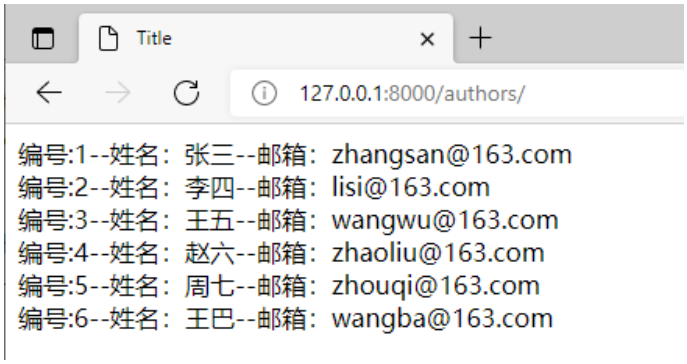
参数说明：

- all_authors 从后台传入到前台的数据参数名
- i 从列表中取一个对象

- i.id 对象i的id属性

5. 效果运行：

在浏览器中访问：http://127.0.0.1:8000/authors/



二、使用table标签格式化数据

1. 使用表格展示数据

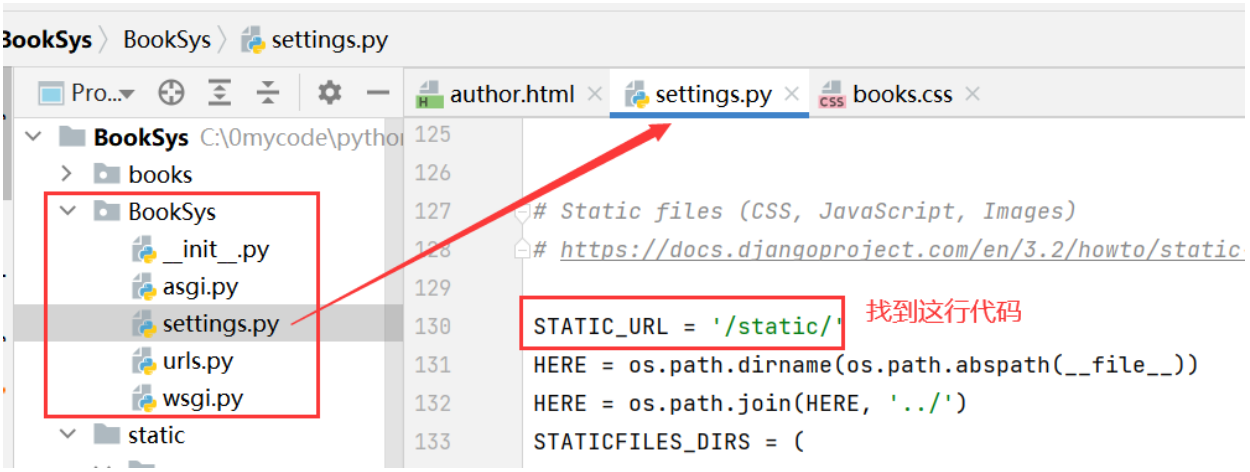
```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Title</title>
</head>
<body>

 <table>
  <tr>
  <th>编号</th>
  <th>姓名</th>
  <th>邮箱</th>
  <th>操作</th>
  </tr>
  {% for i in all_authors %}
  <tr>
  <td>{{ i.id }}</td>
  <td>{{ i.name }}</td>
  <td>{{ i.email }}</td>
  <td>
  <a href="javascript:alert('占位，修改功能待开发')">修改</a>
  <a href="javascript:alert('占位，删除功能待开发')">删除</a>
  <a href="javascript:alert('占位，查看详情功能待开发')">查看详情</a>
  </td>
  </tr>
  {% endfor %}
  </table>
</body>
</html>
```

2. 效果展示

编号	姓名	邮箱	操作
1	张三	zhangsan@163.com	修改 删除 查看详情
2	李四	lisi@163.com	修改 删除 查看详情
3	王五	wangwu@163.com	修改 删除 查看详情
4	赵六	zhaoliu@163.com	修改 删除 查看详情
5	周七	zhouqi@163.com	修改 删除 查看详情
6	王巴	wangba@163.com	修改 删除 查看详情

一、 使用css美化页面

1. Django默认的静态文件配置

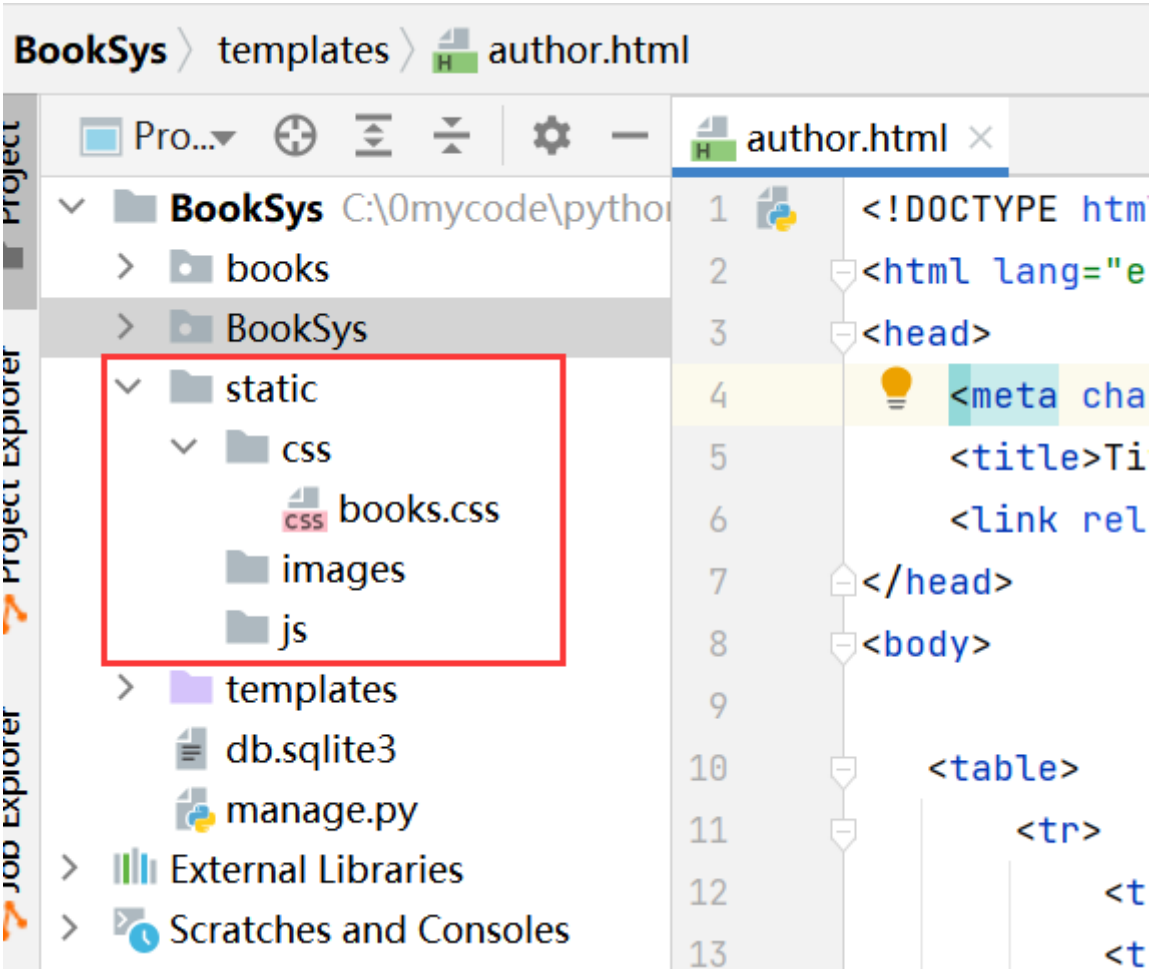


添加如下代码

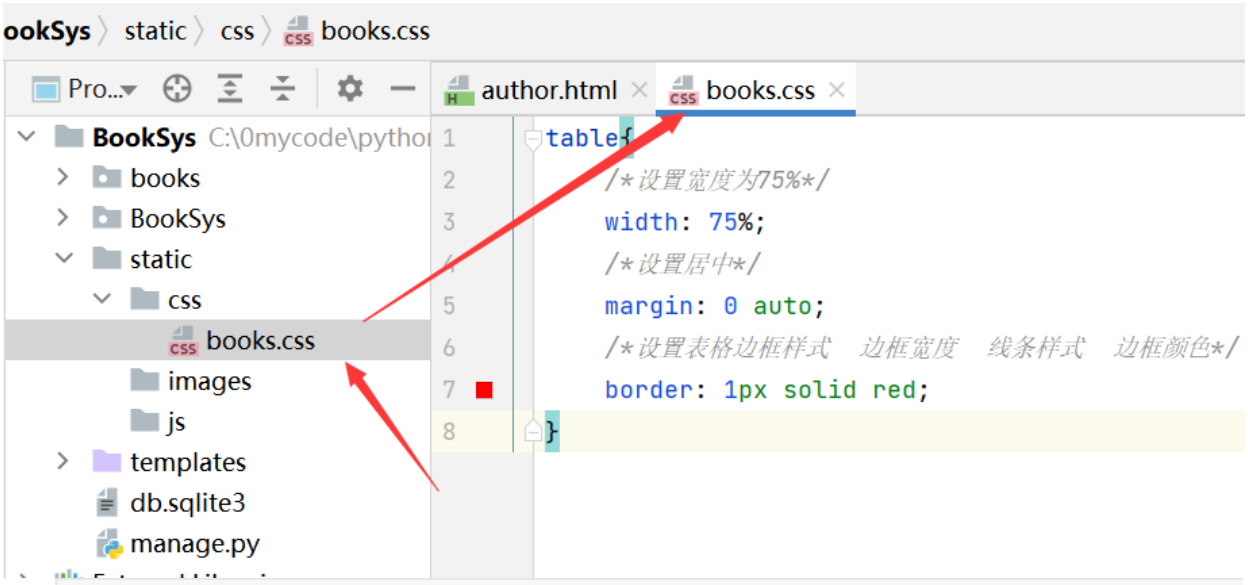
```
STATIC_URL = '/static/'
HERE = os.path.dirname(os.path.abspath(__file__))
HERE = os.path.join(HERE, '../')
STATICFILES_DIRS = (
 # Put strings here, like "/home/html/static" or "C:/www/django/static".
 # Always use forward slashes, even on Windows.
 # Don't forget to use absolute paths, not relative paths.
 os.path.join(HERE, 'static/'),
)
```

2. 配置项目的静态项目结构

在manager.py同层级下创建static文件夹, 里面放上css , js, image等文件或者文件夹



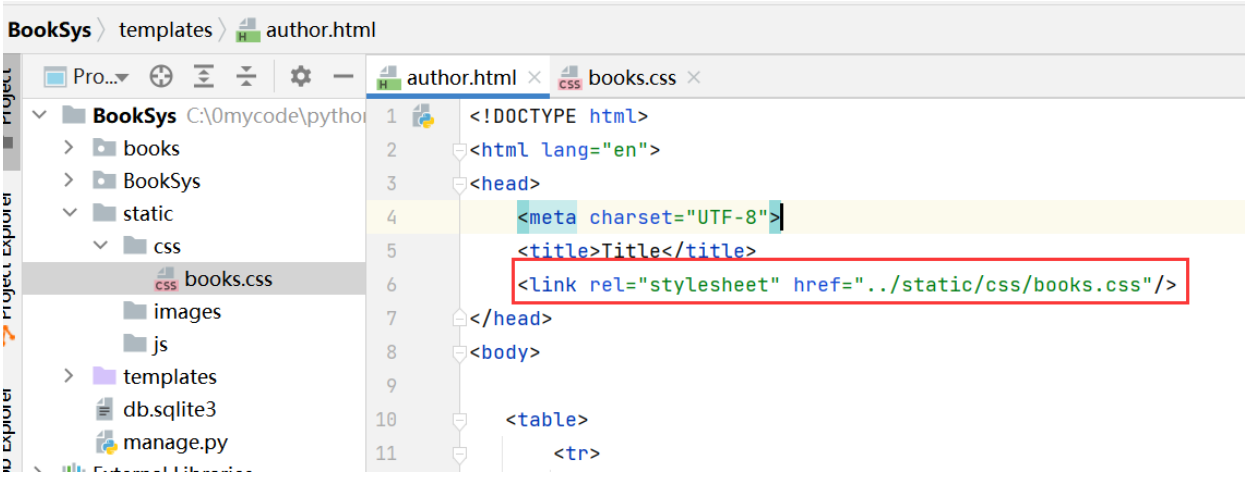
3. 创建books.css样式文件




```
table{
 /*设置宽度为75%*/
 width: 75%;
 /*设置居中*/
 margin: 0 auto;
```

```
6   /*设置表格边框样式 边框宽度 线条样式 边框颜色*/
7   border: 1px solid red;
8 }
```

4. 将创建的books.css导入到html中




```
1 <link rel="stylesheet" href="../static/css/books.css"/>
```

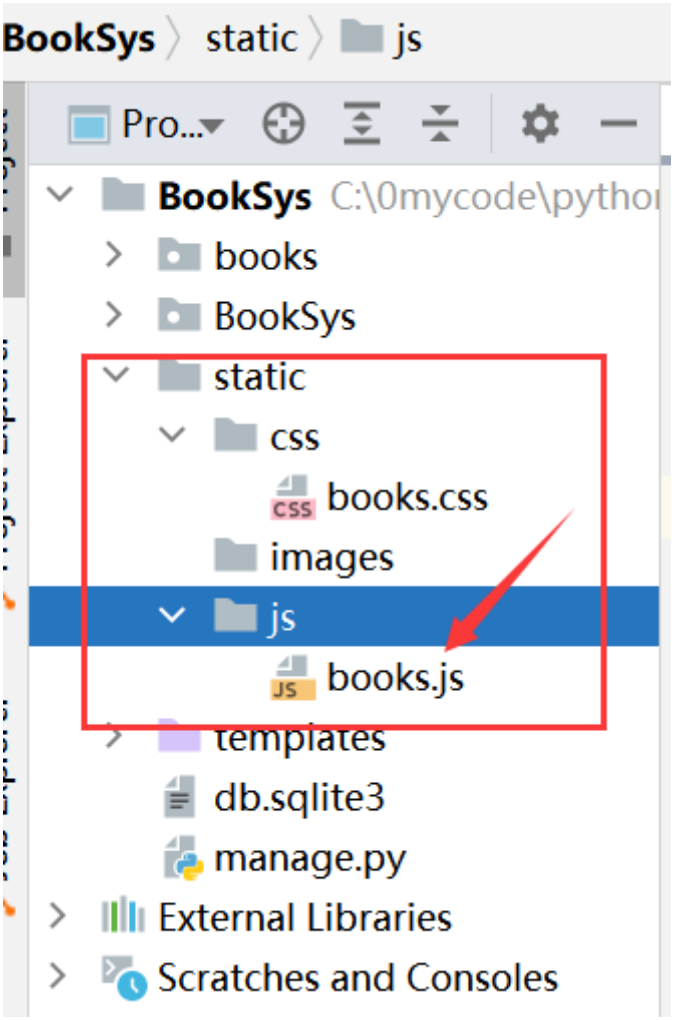
5. 运行效果

浏览器中访问：http://127.0.0.1:8000/authors/

编号	姓名	邮箱	操作
1	张三	zhangsan@163.com	修改 删除 查看详情
2	李四	lisi@163.com	修改 删除 查看详情
3	王五	wangwu@163.com	修改 删除 查看详情
4	赵六	zhaoliu@163.com	修改 删除 查看详情
5	周七	zhouqi@163.com	修改 删除 查看详情
6	王巴	wangba@163.com	修改 删除 查看详情

二、 使用Js为项目添加动态效果

1. 在static下的js文件夹中创建books.js文件



2. 添加js代码

```
1 alert("欢迎在Django中使用JavaScript!!!!");
```

3. 在浏览器中验证 JS是否引用成功

浏览器中访问：http://127.0.0.1:8000/authors/


127.0.0.1:8000/authors/
127.0.0.1:8000 显示
欢迎在Django中使用JavaScript!!!!
确定

一、 通过模板继承来实现页通用页面的共用

1. 模板继承的概念

模板继承是 Django 模板语言中最强大的部分。模板继承使你可以构建基本的“骨架”模板，将通用的功能或者属性写在基础模板中，也叫基类模板或者父模板。子模板可以继承父类模板，子模板继承后将自动拥有父类中的属性和方，我们还可以在子模板中对父模板进行重写，即重写父模板中方法或者属性，从而实现子模板的定制。模板继承大大提高了代码的可重用性，减轻开发人员的工作量。

2. 模板继承的应用

那么模板继承如何使用呢？它的使用场景有哪些呢？最典型的应用是 Web 站点的头部信息和尾部信息，比如 Web 站点的底部广告，每个网页都需要放底部广告，还有 Web 站点的头部导航栏，这些都可以使用模板继承来实现。

在模板继承中最常用了标签就是 {% block %} 与 {% extends %} 标签，其中 {% block% } 标签与 {% endblock %} 标签成对出现，而 {% extends %} 放在子模板的第一行且必须是模板中的第一个标签，标志着此模板继承自父模板，它们使用方法如下所示：

```
#定义父模板可被重写内容
{%block block_name%}
...可以被子模板覆盖的内容
{%endblock block_name%}
#继承父模板
{% extends '父模板名称' %}
#子模板重写父模板
{%block block_name%}
...子模板覆盖后呈现的新内容
{%endblock block_name%}
```

需要注意的是子模板不需要重写父模板中的所有 block 标签定义的内容，未重写时，子模板原封不动的使用父模板中的内容。下面我们通过一个简单的例子来看一下具体的实现过程。

二、 定义整体框架页面

```
<!DOCTYPE html>
<html>
```

```
<head>
<meta charset="utf-8">
<title></title>

<link rel="stylesheet" href="../static/css/books.css"/>
<script src="../static/js/books.js"></script>
</head>

 <body>
 <div>
 <header>
 <h3>音乐推荐系统</h3>
 </header>
 <menu>
 <a href="javascript:alert('占位：页面跳转')">作者列表</a>
 <button type="button" onclick="file_new()">New...</button>
 <button type="button" onclick="file_open()">Open...</button>
 <button type="button" onclick="file_save()">Save</button>
 <button type="button" onclick="file_new()">New...</button>
 <button type="button" onclick="file_open()">Open...</button>
 <button type="button" onclick="file_save()">Save</button>
 <button type="button" onclick="file_new()">New...</button>
 <button type="button" onclick="file_open()">Open...</button>
 <button type="button" onclick="file_save()">Save</button>
 </menu>
 </div>
 <div>
 <h1>网页中的可变内容</h1>
 {% block content %}
 <p>这是主体内容可以被子模板重写</p>
 {% endblock content %}
 </div>
 <div>
 <footer>
 <pre>
 脚部内容
 脚部内容
 脚部内容
 脚部内容
 </pre>
```

```
 </footer>
 </div>

 </body>
</html>
```

其中以下部分是可重写部分

```
{% block content %}
 <p>这是主体内容可以被子模板重写</p>
{% endblock content %}
```

二、 定义子页面

1. 核心代码说明：

- 子模版继承父模板

```
{% extends 'base.html' %}
```

- 使用block content包裹住子模版中新添加的内容

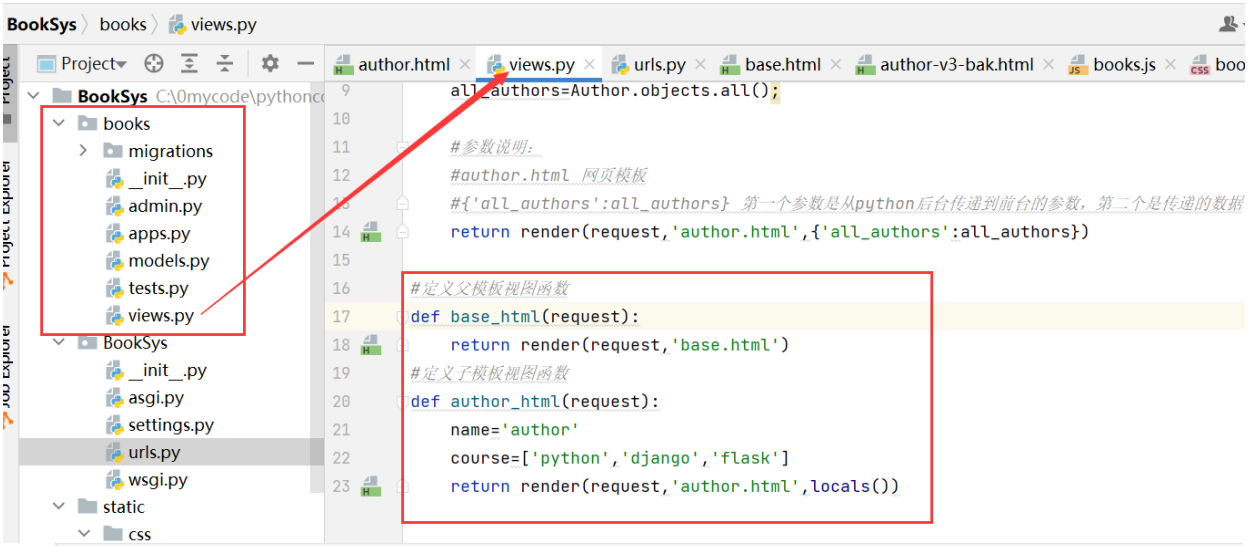
```
{% block content %}
……
{% endblock content %}
```

2. 完整代码

```
{% extends 'base.html' %}
{% block content %}
 <table>
 <tr>
 <th>编号</th>
 <th>姓名</th>
 <th>邮箱</th>
 <th>操作</th>
 </tr>
  {% for i in all_authors %}
 <tr>
 <td>{{ i.id }}</td>
 <td>{{ i.name }}</td>
 <td>{{ i.email }}</td>
 <td>
```

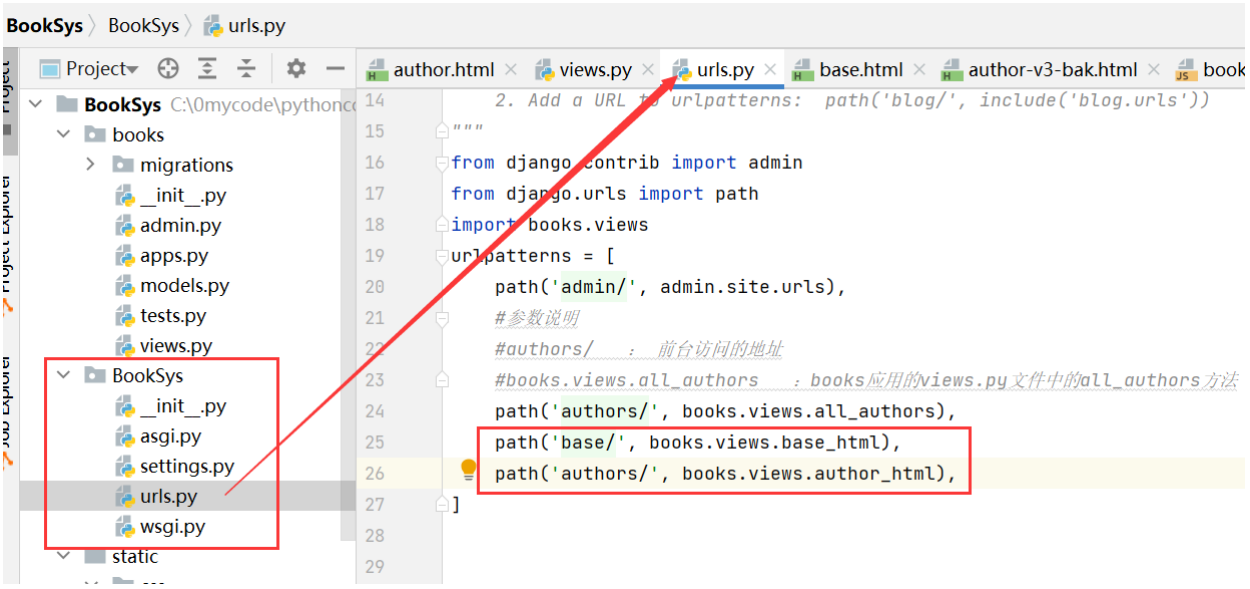
```
<a href="javascript:alert('占位，修改功能待开发')">修改</a>
<a href="javascript:alert('占位，删除功能待开发')">删除</a>
<a href="javascript:alert('占位，查看详情功能待开发')">查看详情</a>
</td>
</tr>
{% endfor %}
</table>
{% endblock content %}
```

三、 添加模板对应的视图函数




```
#定义父模板视图函数
def base_html(request):
 return render(request,'base.html')
#定义子模板视图函数
def author_html(request):
 name='author'
 course=['python','django','flask']
 return render(request,'author.html',locals())
```

四、 添加对应的地址映射




```
path('base/', books.views.base_html),
path('authors/', books.views.author_html),
```

五、 页面访问效果

1. 访问父模板

http://127.0.0.1:8000/base/

2. 访问子模板

http://127.0.0.1:8000/authors/

127.0.0.1:8000/authors/

音乐推荐系统

作者列表 New... Open... Save New... Open... Save New... Open... Save

网页中的可变内容

编号	姓名	邮箱	操作
1	张三	zhangsan@163.com	修改 删除 查看详情
2	李四	lisi@163.com	修改 删除 查看详情
3	王五	wangwu@163.com	修改 删除 查看详情
4	赵六	zhaoliu@163.com	修改 删除 查看详情
5	周七	zhouqi@163.com	修改 删除 查看详情
6	王巴	wangba@163.com	修改 删除 查看详情