# 一、项目搭建、结构配置与启用

## 项目创建

django-admin startproject vega自动生成vega项目，其中有项目基本结构

python manage.py startapp app\_tower生成名为‘app\_tower’的app应用

## 结构配置

1. 配置：

项目基本配置位于vega/vega/settings.py文件中



ALLOWED\_HOSTS用来配置应用访问ip

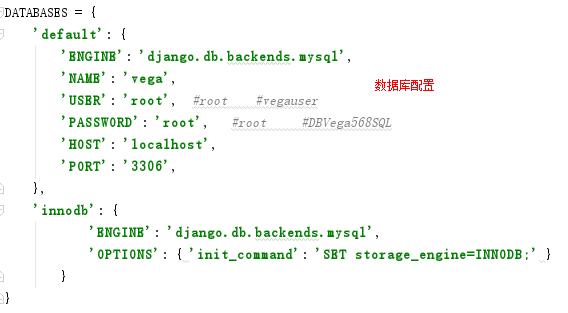


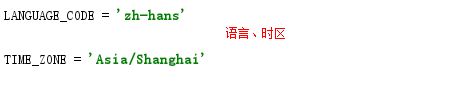
INSTALLED\_APPS用来加入自己创建的app应用（app\_tower）

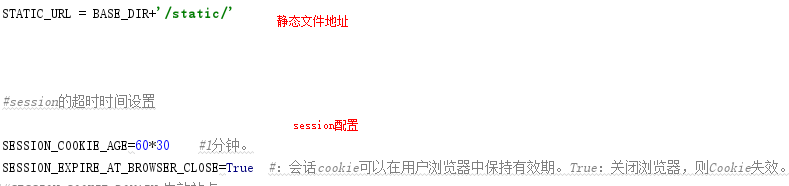


MIDDLEWARE\_CLASSES中间件配置，可加入自己所写中间件（LoinMiddleware）





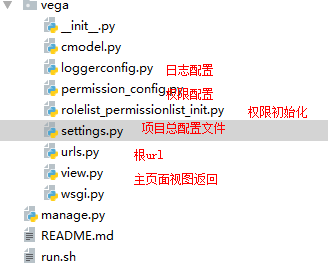




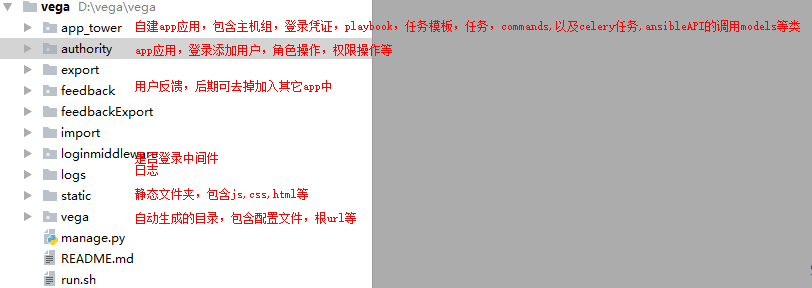


1. 项目结构：

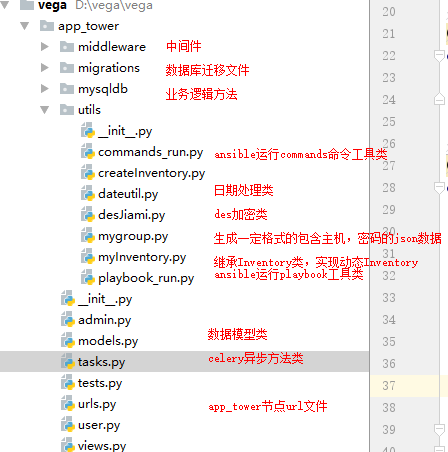
Vega/vega结构



项目总结构

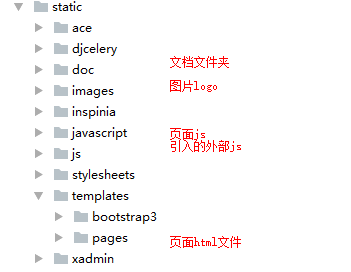


app\_tower为例app结构



以上是主要app app\_tower的结构，authority和feedback类似

静态文件：



## 项目启动

如vega/app\_tower/models.py文件中模型类有增加或改变，需要先生成数据迁移文件，同步数据库：

1. python manage.py makemigrations # 让 Django 知道我们在我们的模型有一些变更（当上传新版本时，要把所有app的migrations文件夹下的除\_\_init\_\_.py文件外的文件复制到新上传项目对应文件夹）  
     
   2. python manage.py migrate # 创建表结构

启动服务：

python manage.py runserver 0.0.0.0:80 &

再启动celery worker:

export PYTHONOPTIMIZE=1  
 python manage.py celery worker -c 4 --loglevel=info --autoreload &  
 #一次最多4个任务，其他将会排队

# 二、项目重要知识点介绍

## Celery异步任务队介绍

### 参考网址

celery在django中的使用：<http://www.cnblogs.com/znicy/p/5626040.html>

Celery定时任务：<http://blog.csdn.net/sicofield/article/details/50937338>

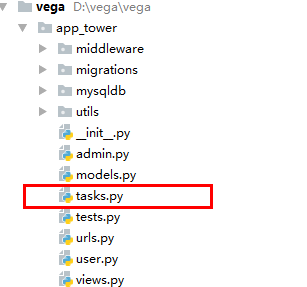
### 项目中配置

安装模块django==1.8、django-celery==3.1.17、celery==3.1.17、redis==2.10

如图，在vega/vega/settings.py中配置celery



在app根目录下建立tasks.py文件



tasks.py文件中引入task,并使用@task来使一个普通的方法变为celery异步方法

### celery方法的使用

如vega/app\_tower/mysqldb/jobTemplatedb.py中执行playbook和commands均需要用到celery方法



使用delay调用celery方法来创建任务，并送入消息队列

taskid = result.task\_id 获取任务的id

result = AsyncResult(taskid) 根据任务id获取任务

result.status 获取任务执行状态

revoke(taskid,terminate=True,signal=**'SIGKILL'**) 根据任务id停止任务

## Django 数据模型models介绍

### 参考网址

模型类介绍

<http://python.usyiyi.cn/translate/django_182/topics/db/models.html>

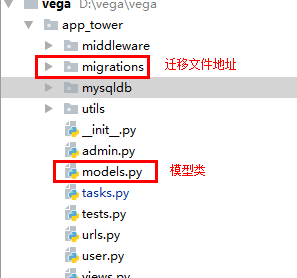
条件查询

<https://www.douban.com/note/301166150/>

外键查询，反向查询，多表联查

<http://python.usyiyi.cn/translate/django_182/topics/db/queries.html>

### 项目中配置



### 使用

类的编写：



其他知识可查看更详细文档

数据库操作：

**from** app\_tower.models **import** T\_Group引入类

增

group = T\_Group(NAME=form[**'name'**], DESCRIPTION=form[**'description'**], VARIABLES=form[**'variables'**],OWNER\_ID=OWNER\_ID,OWNER\_NAME=OWNER\_NAME,OWNER\_PROJECT\_ID=OWNER\_PROJECT\_ID,OWNER\_ALL=OWNER\_ALL,CREATE\_USER\_ID=request.session[**'userId'**] ,CREATE\_USER\_NAME=request.session[**'username'**]) 实 例化  
group.save() 存入数据库

注意只有使用了save方法后才执行插入操作

#批量插入

project\_user\_list = list()

for x in form['USERS']:

project\_user\_list.append(User.objects.get(id=x))

T\_PROJECT\_User\_ID.objects.bulk\_create(project\_user\_list)

删

inventorie = T\_Group.objects.get(id=form[**'id'**])

inventorie.delete()

###### #批量删除

T\_PROJECT\_User\_ID.objects.filter(PROJECT\_ID=project.id).delete()

改

inventorie = T\_Group.objects.get(id=form[**'id'**])  
inventorie.NAME = form[**'name'**]  
inventorie.DESCRIPTION = form[**'description'**]

inventorie.save()

查

普通查询所有

playbooks=playbook.objects.all(request) #all默认查所有可编写自己的manager方法

得到的playbooks是queryset类型，不能被方便使用，需要序列化  
playbooksList=serializers.serialize(**'json'**, playbooks, ensure\_ascii=False)  
true = True  
null = None  
false=False

eval(playbooksList) 得 到列表

条件查询，分页查询，排序查询

*# 通过objects这个模型管理器的all()获得所有数据行，相当于SQL中的SELECT \* FROM Test.objects.filter(name="runoob").order\_by("id")*t\_group\_List = T\_Group.objects.check\_own(request).filter(NAME\_\_contains=name).filter(DESCRIPTION\_\_contains=description).order\_by(orderBy)  
total=len(t\_group\_List)  
list = t\_group\_List[int(offset):int(offset) + int(limit)]

response\_data[**'rows'**] = serializers.serialize(**'json'**, list, ensure\_ascii=False)

通过filter链式操作实现条件查询，order\_by排序查询，[1,5]中括号分页查询

可参考

条件查询

<https://www.douban.com/note/301166150/>

外键查询，反向查询，多表联查

<http://python.usyiyi.cn/translate/django_182/topics/db/queries.html>

## ansible执行playbook,commands的API,封装类介绍

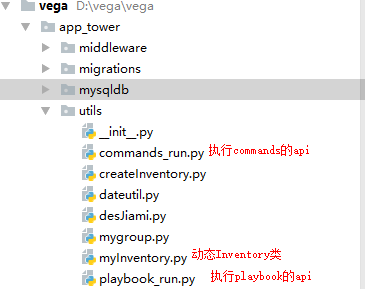
### 参考网址

<https://segmentfault.com/a/1190000008009639>

<http://www.jianshu.com/p/8558befb16c1>

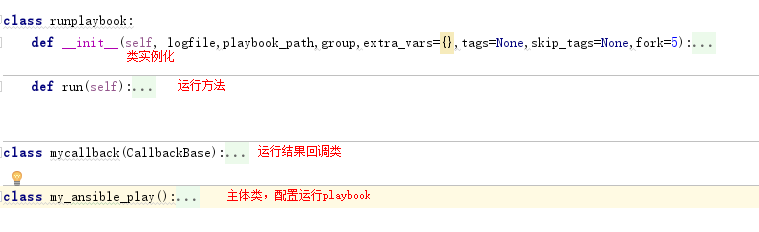
<http://blog.csdn.net/Allison_ywt/article/details/52880352>

### 项目中配置



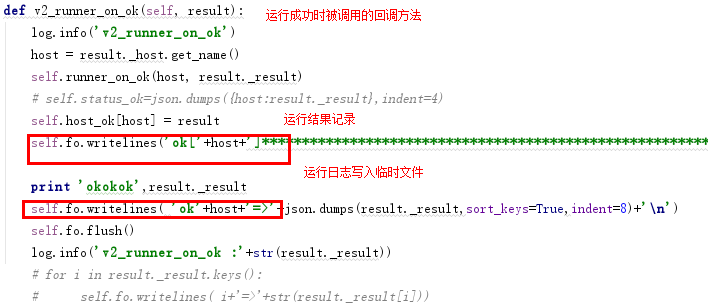
### 类介绍

playbook\_run.py文件



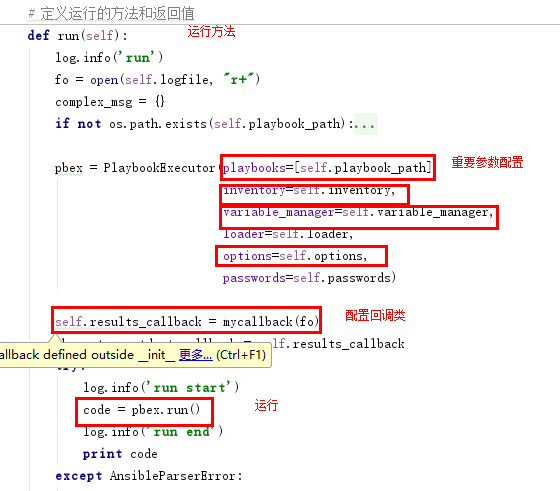














commands\_run.py文件



基本和playbook\_run.py文件一样，只是调用的api不一样，需要自己查看资料明白一些参数的含义

### 使用

两种ansible操作都是耗时操作，因此使用celery方法中调用



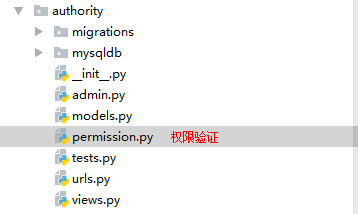
执行playbook类似，其中还要做一些记录入库操作，许多传递参数需要一一弄清楚

## Django装饰器介绍

### 参考网址

<https://mozillazg.github.io/2013/02/django-custom-decorators-optional-keyword-arguments-for-views.html>

### 项目中配置



本项目中用来装饰需要权限的方法，装饰器中逻辑为判断是否具有调用该方法的权限



### 使用

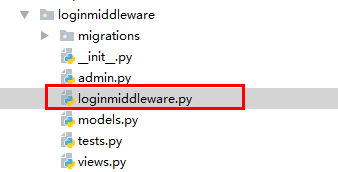


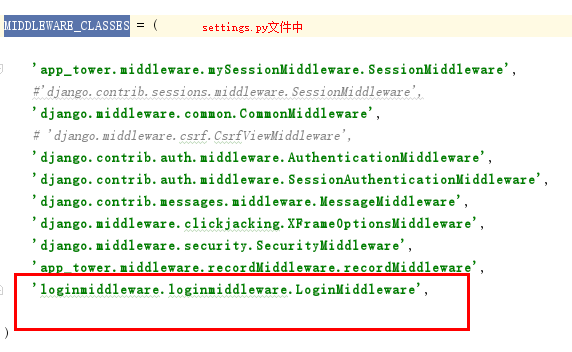
## 中间件介绍

### 参考网址

<http://python.usyiyi.cn/translate/django_182/topics/http/middleware.html>

### 项目中配置







## 权限的实现原理（角色权限、project项目组权限、菜单权限）

### 角色权限

<1>model类

用户表User与角色表RoleList是多对一的关系，User中使用外键连接RoleList；

RoleList与权限表PermissionList是多对多的关系，使用中间表



<2>permission\_config.py权限配置文件

在permission\_config.py文件中配置权限例：



初始化一些角色权限，例：

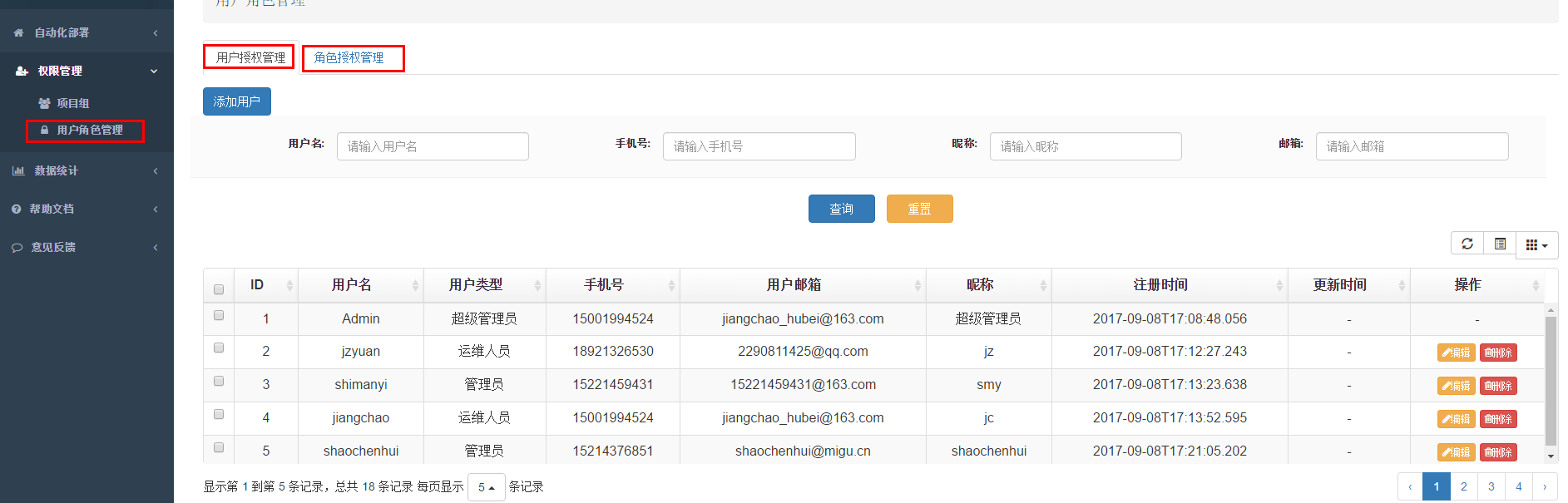


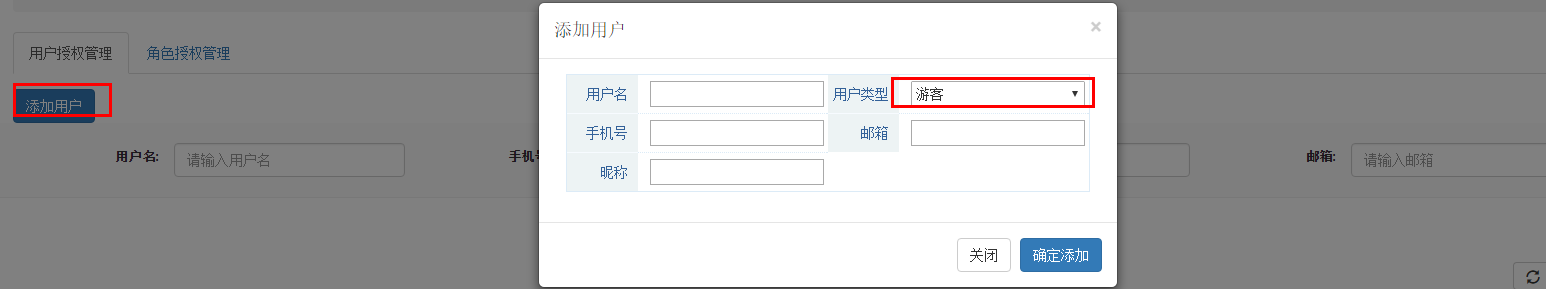
在rolelist\_permissionlist\_init.py文件中进行入库操作，在view.py中引入rolelist\_permissionlist\_init.py中类rolelist\_permission即可在项目启动时进行权限相关表的初始化入库操作。

<3>利用装饰器permission.py

在相关权限方法上加上@PermissionVerify()加入权限控制

<4>页面相关操作



添加用户或者编辑用户时可选择角色

可以给默认角色分配权限，也可以自定义角色自由分配权限



### project项目组权限

<1>model类配置管理器编写

可选择用户加入项目组，playbook、主机组、登录凭证、任务模板等在创建时选择可使用项目组，当使用这些资源时调用checkOwnManager类中check\_own方法，过滤不属于自己所在项目组的资源。在这些资源类中都加入如下字段，并根据页面选择使用权限来进行相应记录：

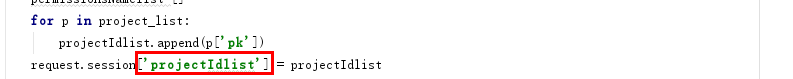


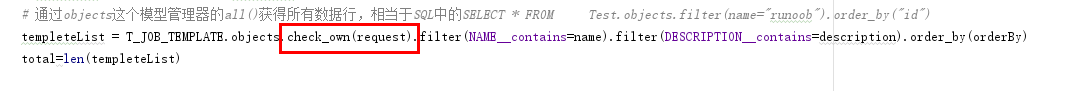
编辑管理类自定义查询方法：



<2>使用

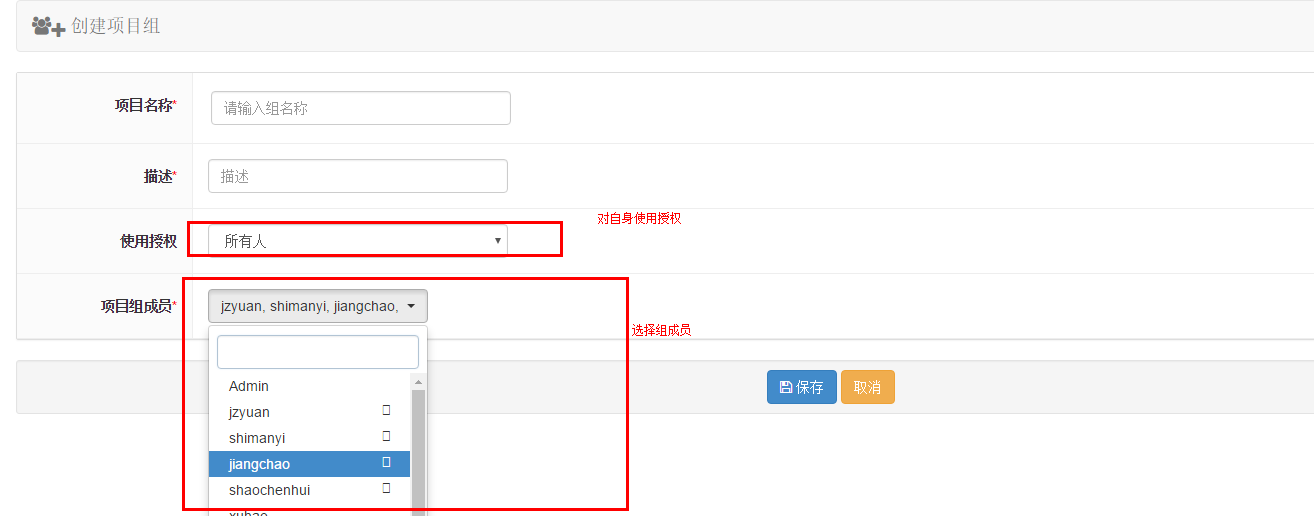
在登录时将用户所属项目组id加入request.session中：



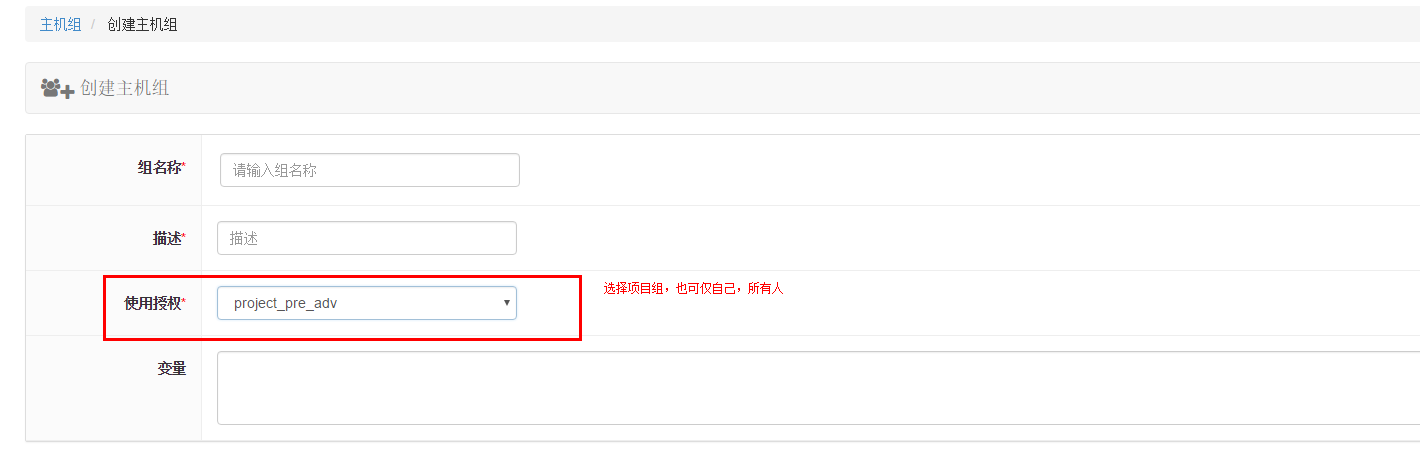
查询资源类时使用check\_own方法代替all：  


<3>页面相关操作

创建项目组，可对自己使用授权，一般为组长创建，推荐仅自己使用：



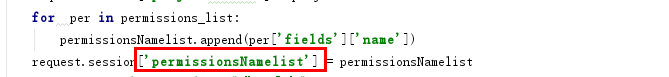
创建资源时，选择使用授权，提供仅自己，所有人，和项目组选项：



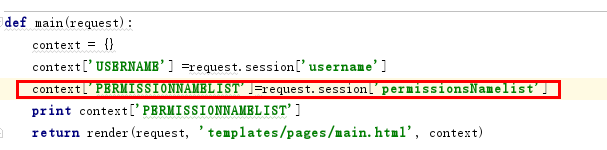
如此不同项目组之间可使用资源分开，实现项目组权限。

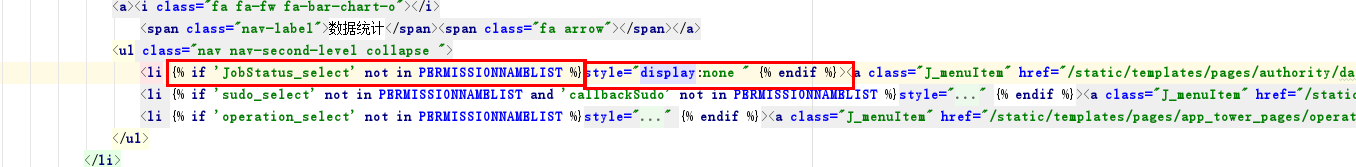
### 菜单权限

在登录时将用户拥有权限加入request.session中：



在main.html页面，使用模板标签，控制菜单显示和隐藏





## Bootstrap，js前端介绍

### 参考网址

菜鸟教程，快速入门查询：<http://www.runoob.com/bootstrap/bootstrap-tutorial.html>

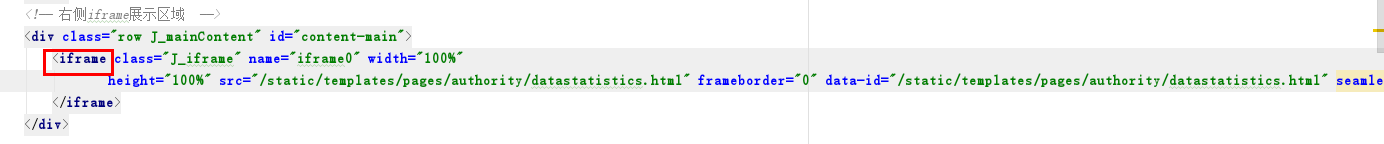
### 项目中配置

页面中按顺序引入<**link href="/static/js/bootstrap/css/bootstrap.min.css" rel="stylesheet"**>

<**script src="/static/js/bootstrap/js/jquery-3.2.1.min.js"**></**script**>  
<**script src="/static/js/bootstrap/js/bootstrap.min.js" type="text/javascript"**></**script**>

即可应用bootstrap相关工具。

主页面main.html布局采用xadmin的一些js和css,左边菜单栏，右上面导航栏，右下工作区域，采用iframe

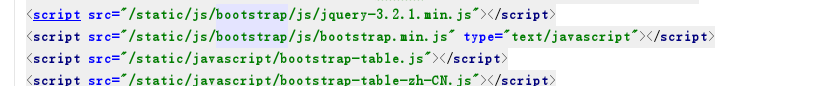


### 主要控件的使用

<1>表格bootstrap-table

页面引入



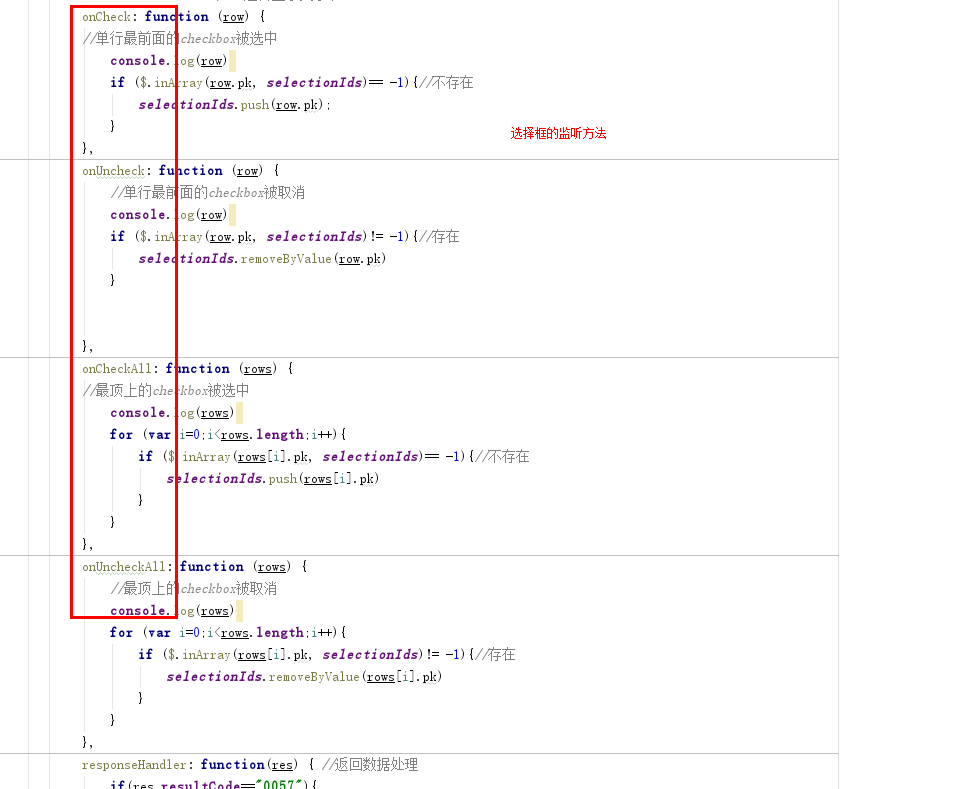


在需要展示表格的位置加入标签



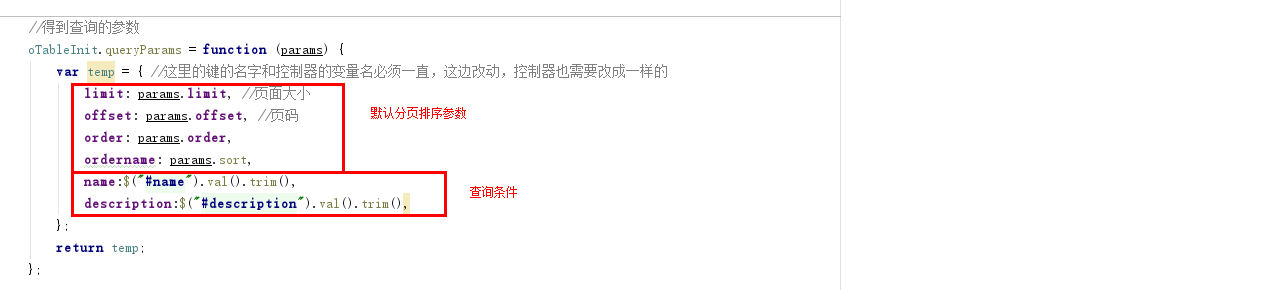
引入页面js后，配置如下







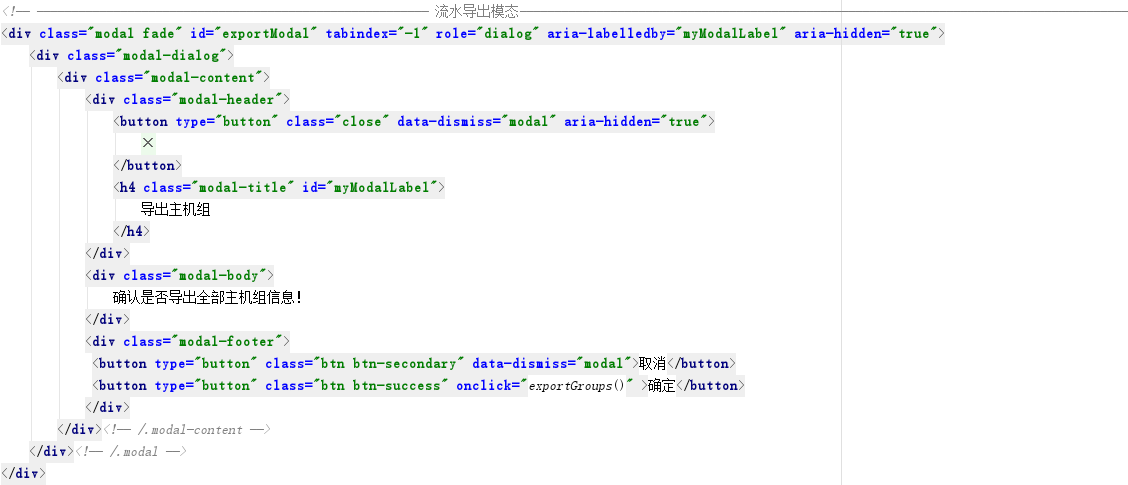




Bootstrap-table的更详细配置参数，方法，需自己查看相关文档

<2>模态框

Bootstrap的一种组件，页面遮罩弹出的功能框



可在js中控制显示$("#myModal").modal("show");也可在页面标签属性中配置显示隐藏

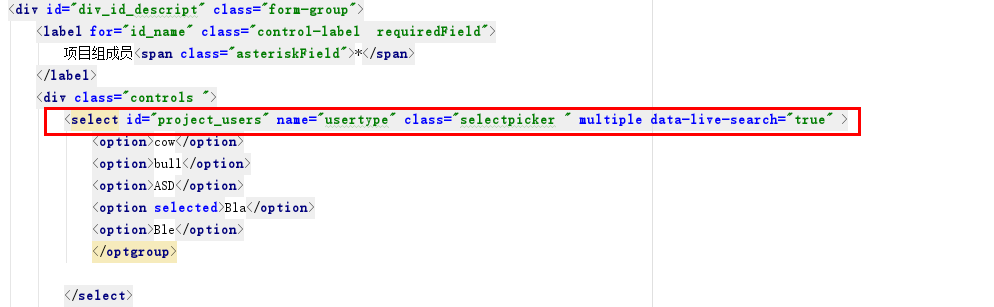
<3>多项选择下拉列表

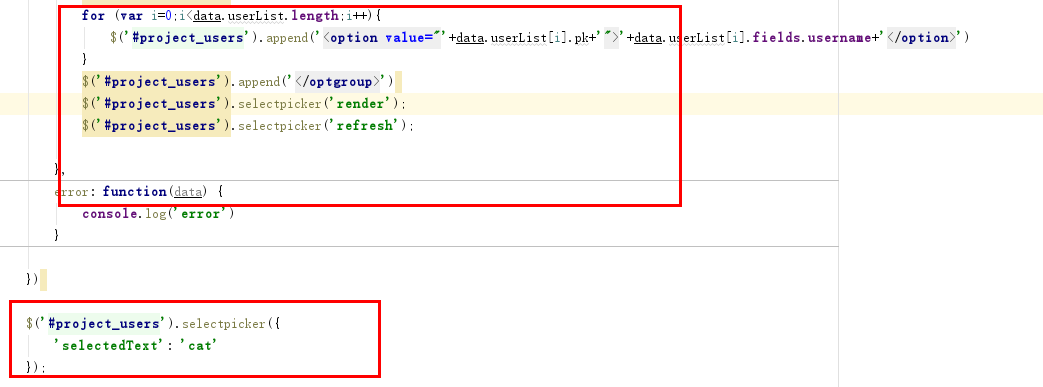
Bootstrap-select

项目组选择成员用到







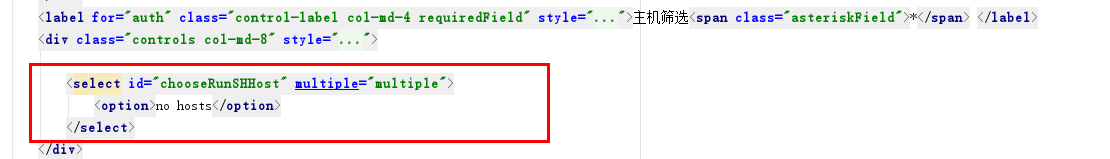


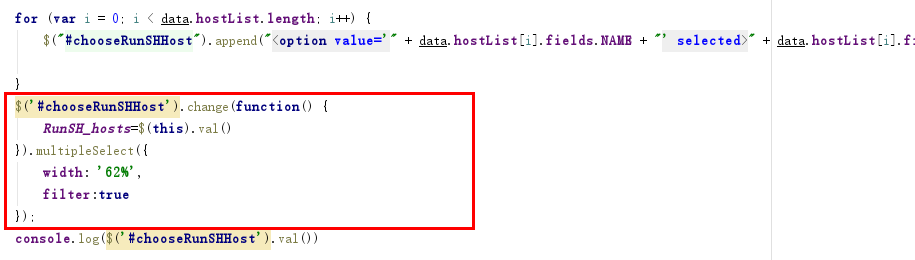
multiple-select

主机筛选用到









<4>双向列表

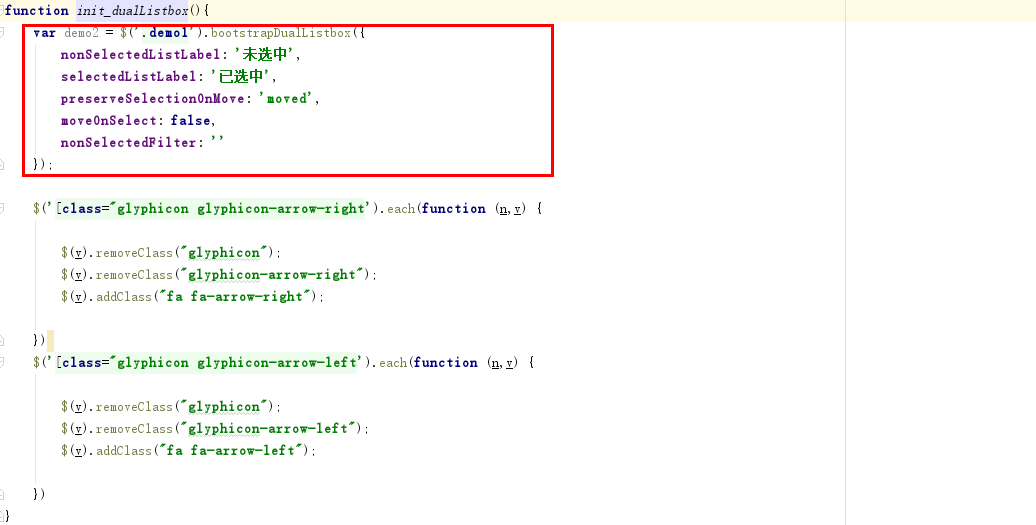
bootstrap-DualListbox

生成回滚主机组用到









<5>上传控件

Bootstrap-fileinput









## 日志配置

### 参考网址

<http://python.usyiyi.cn/translate/django_182/topics/logging.html>

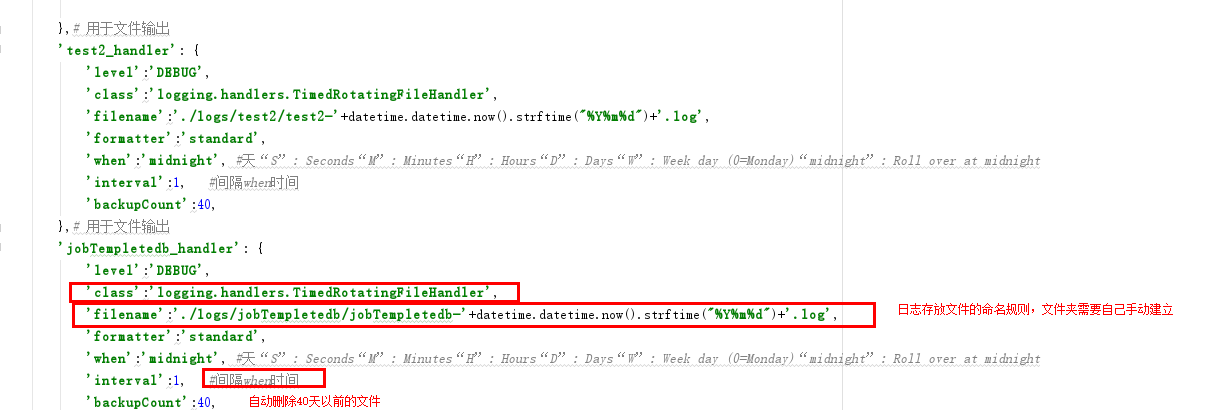
### 项目中配置

vega/vega/settings.py中引入

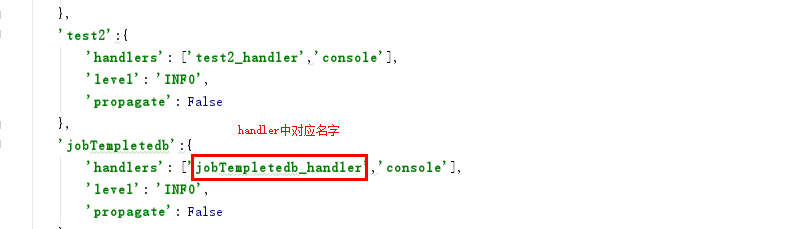


vega/vega/loggerconfig.py中

先配置handler



在配置loggers



### 使用



log.info()

log.error()

log.debug()等

# 举例说明项目前后端编写流程

### 表格的展示

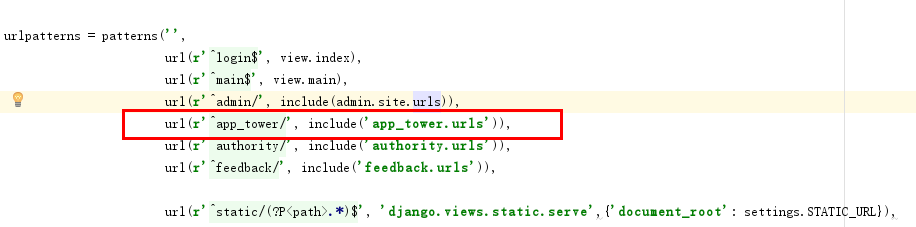
以主机组表格展示为例：

页面引入group.js，在js中配置表格所需参数：



后端配置路由：

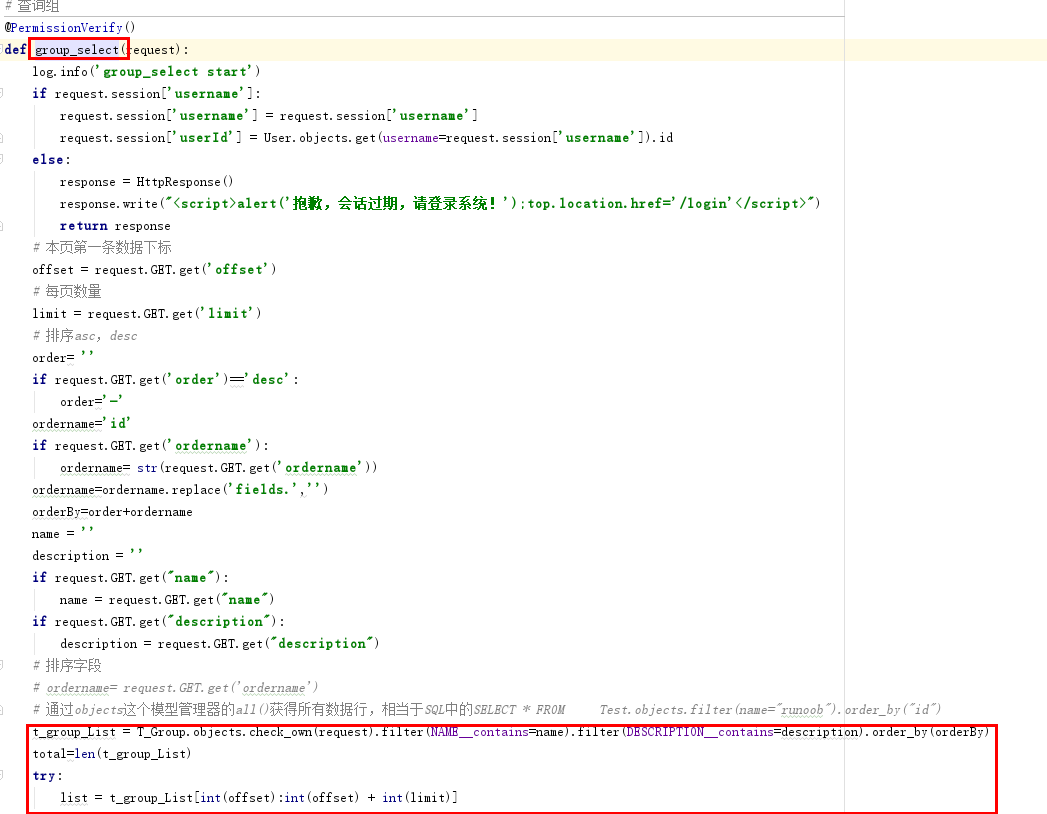
Vega/vega/urls.py主路由中



app\_tower/urls.py中



app\_tower/mysqldb/inventoriesdb.py中编写具体方法：



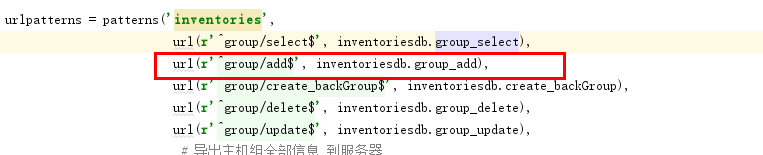
### 增删改ajax方法的调用

以添加主机组为例：

页面引入create\_group.js，在js中：



app\_tower/urls.py中



app\_tower/mysqldb/inventoriesdb.py中编写具体方法：

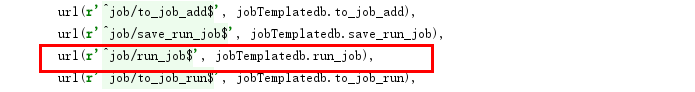


### 运行任务（playbook）

js传参



app\_tower/urls.py中配置路由



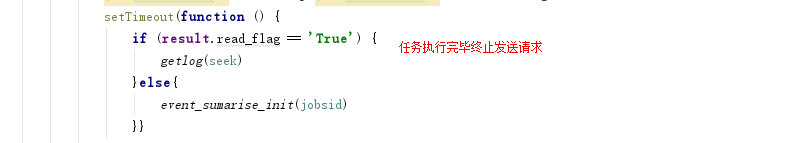
app\_tower/jobTempletedb.py中编写具体方法





返回成功后，js中跳转到任务执行页面，实时查看任务执行状态。根据任务信息定时发送请求获取任务执行情况，并展示执行日志

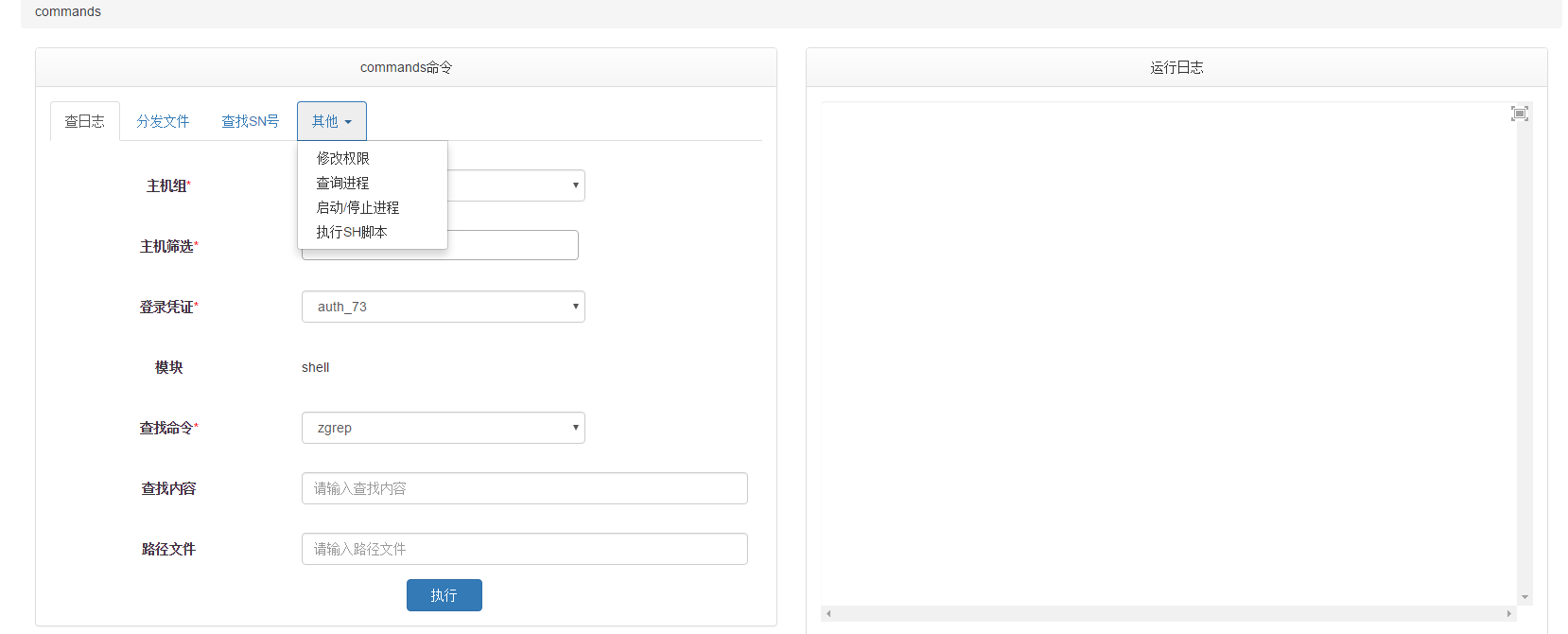




还提供停止任务按钮，生成执行结果概览，生成回滚主机组按钮功能

### 执行commands

ansible group -m [module name] -a “vars”

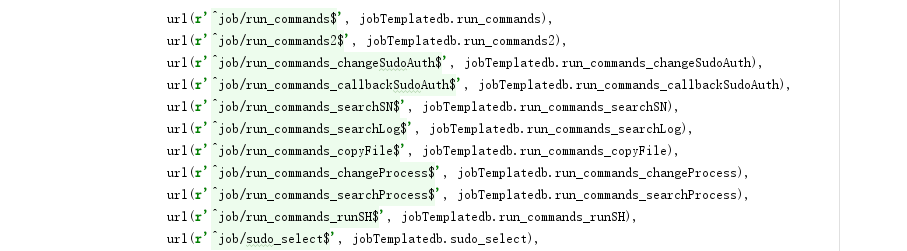


Commands.js中初始化该页面下拉框信息

发ajax请求



urls.py中编写路由



app\_tower/jobTempletedb.py中调用celery中执行commands方法

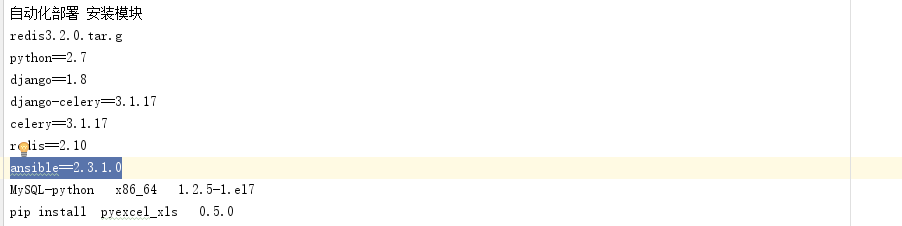


# 四、项目运行环境第三方模块版本

tower地址：

<https://tower.im/projects/9216060391c945dca76813527564ff83/docs/042a38800f394e7b843077b9f8df1d1c/>

README文件中：



# 重要参考网址

菜鸟教程（django,python,bootstrap快速入门）：

<http://www.runoob.com/python/python-tutorial.html>

Django1.8中文文档（十分详细，django各方面知识基本都能找到）：

<http://python.usyiyi.cn/translate/django_182/index.html>

celery在django中的使用：

<http://www.cnblogs.com/znicy/p/5626040.html>

Celery定时任务：

<http://blog.csdn.net/sicofield/article/details/50937338>

Ansible2.0的api使用：

<https://segmentfault.com/a/1190000008009639>

<http://www.jianshu.com/p/8558befb16c1>

<http://blog.csdn.net/Allison_ywt/article/details/52880352>

Ansible中文文档：

[http://www.ansible.com.cn/docs/intro\_dynamic\_inventory.html#static-groups-of-dynamic](http://www.ansible.com.cn/docs/intro_dynamic_inventory.html" \l "static-groups-of-dynamic)