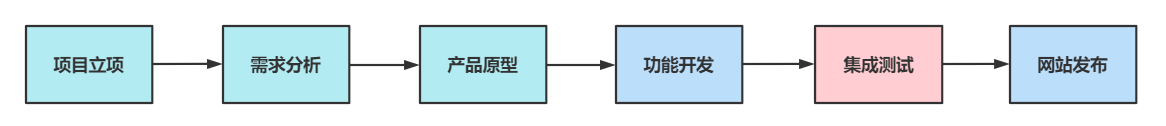
**《CMDB项目文档》**

|  |  |
| --- | --- |
| 作者信息 | 李振良（阿良），微信：k8init |
| 阿良教育官网 | [http://www.aliangedu.cn](http://www.aliangedu.cn/) |
| 说明 | 该文档有导航窗格，方便阅读，如果左侧没有显示，请检查word是否启用。  内部学员资料，请勿随意转载。 |



# 企业项目开发流程



# 1. 引言

## 1.1 项目背景

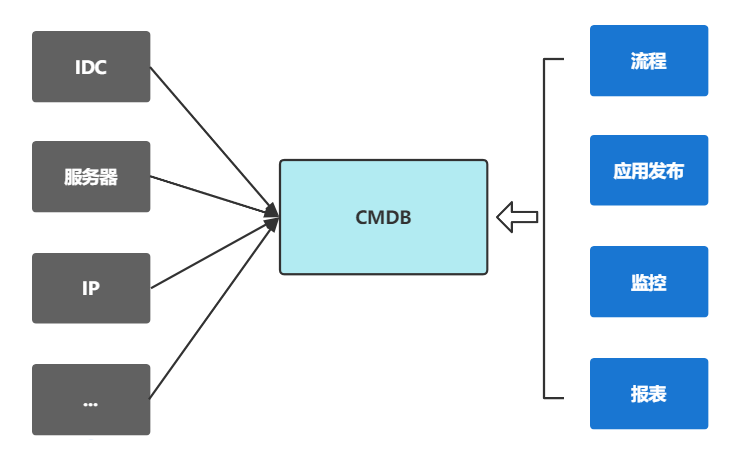
目前运维管理存在痛点：随着业务增长，服务器数量越来越多，资产信息通过Excel记录，人工管理低效，易于出错。

## 1.2 CMDB介绍

配置管理数据库（Configuration Management Database，CMDB），是一个逻辑数据库，包含了应用生命周期的信息，例如服务器、物理关系、通信关系、依赖关系等。

CMDB存储与管理企业IT架构中设备的各种配置信息，它与所有运维服务和应用发布流程都紧密相联，支持这些流程的运转、发挥配置信息的价值，同时依赖于相关流程保证数据的准确性。CMDB可以实现高度的自动化，减少人为错误的发生、降低人员成本，CMDB是实现运维自动化的基础。

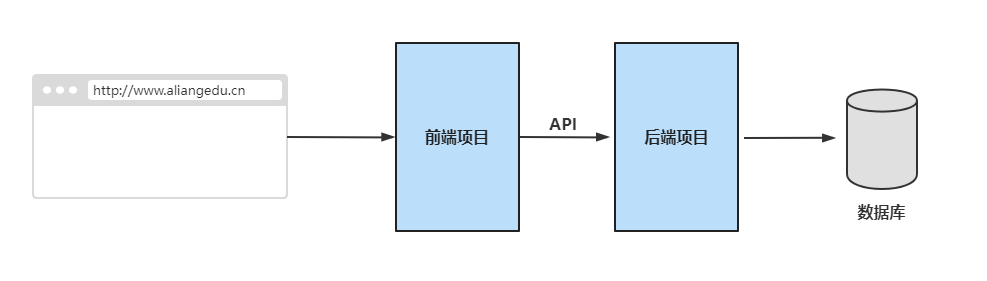
CMDB资产有：



CMDB数据存储需要注意的事项：

* CMDB的目的是为了在其他流程或应用之间共享数据的，如果一个应用或流程需要对某类数据单独使用的话，则不建议将这类数据存入CMDB中，存在自身应用即可。
* 动态数据不建议存储在CMDB中，例如CPU使用率、内存使用率，因为这类数据更新过于频繁。
* 如果没有任何流程、应用及人员，需要对特定的数据进行使用，则没有必要放到CMDB中存储。

## 1.3 技术选型



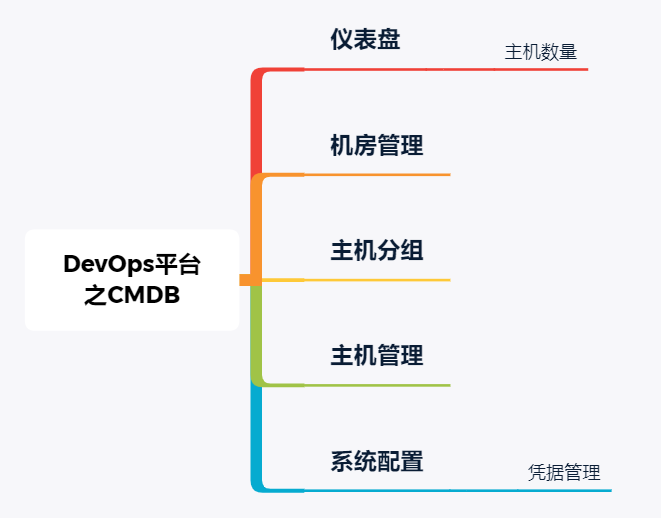
前端项目技术栈

* Vue
* Vue-router
* Element Plus
* Axios

后端项目技术栈

* Python
* Django DRF
* Mysql

## 1.4 整体设计



# 2. 数据库设计



## 2.1 机房管理

表名：cmdb\_idc

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 空 | 名称 |
| id | INTEGER | 否 | 自增长ID |
| name | VARCHAR(30)（unique） | 否 | 机房名称 |
| city | VARCHAR(20) | 否 | 城市 |
| provider | VARCHAR(20) | 否 | 运营商 |
| note | TEXT | 是 | 备注 |
| create\_time | DATETIME | 否 | 创建时间 |

## 2.2 主机分组

表名：cmdb\_server\_group

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 空 | 名称 |
| id | INTEGER | 否 | 自增长ID |
| name | VARCHAR(30)  （unique） | 否 | 分组名称 |
| note | TEXT | 是 | 备注 |
| create\_time | DATETIME | 否 | 创建时间 |

## 2.3 主机管理

表名：cmdb\_server

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 空 | 名称 |
| id | INTEGER | 否 | 自增长ID |
| idc | IDC表一对多关系 | 否 | IDC机房 |
| server\_group | 分组表多对多关系，默认“Default”组 | 否 | 主机分组 |
| credential | 凭据表一对多 | 否 | 凭据ID |
| name | VARCHAR(30) | 否 | 名称，默认与主机名一样 |
| hostname | VARCHAR(30)，unique  （唯一索引） | 否 | 主机名，唯一标识符 |
| ssh\_ip | VARCHAR(40) | 否 | SSH IP |
| ssh\_port | INTEGER | 否 | SSH端口 |
|  |  |  |  |
| machine\_type | VARCHAR(20) | 是 | 机器类型（虚拟机、云主机、物理机） |
| os\_version | VARCHAR(30) | 是 | 系统版本 |
| public\_ip | JSON | 是 | 公网IP（列表存储，会有多个ip） |
| private\_ip | JSON | 否 | 内网IP（列表存储） |
| cpu\_num | VARCHAR(10) | 是 | CPU数量 |
| cpu\_model | VARCHAR(100) | 是 | CPU型号 |
| memory | VARCHAR(30) | 是 | 内存 |
| disk | JSON | 是 | 硬盘（列表存储，包含设备、容量、硬盘类型） |
| put\_shelves\_date | DATE | 是 | 上架日期，默认为系统启动时间 |
| off\_shelves\_date | DATE | 是 | 下架日期 |
| expire\_datetime | DATETIME | 是 | 租约过期时间 |
| is\_verified | VARCHAR(10) | 是 | SSH验证状态（已验证，未验证） |
| note | TEXT | 是 | 备注 |
| update\_time | DATETIME | 是 | 更新时间 |
| create\_time | DATETIME | 否 | 创建时间 |

## 3.2 系统配置：凭据管理

表名：system\_config\_credential

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 空 | 名称 |
| id | INTEGER | 否 | 自增长ID |
| name | VARCHAR(30) | 否 | 名称 |
| auth\_mode | VARCHAR(30) | 否 | 认证方式，key、pass |
| username | VARCHAR(20) | 否 | 用户名 |
| password | VARCHAR(30) | 是 | 密码 |
| private\_key | TEXT | 是 | 私钥 |
| note | TEXT | 是 | 备注 |
| update\_time | DATETIME | 否 | 更新时间 |
| create\_time | DATETIME | 否 | 创建时间 |

# 3. API平台开发

## 3.1 接口设计

|  |  |  |  |
| --- | --- | --- | --- |
| 请求路径 | http方法 | 功能 | 备注 |
| /api/cmdb/idc/ | get,post,put,delete | 查看，创建，更新，删除 | IDC机房 |
| /api/cmdb/server\_group/ | get,post,put,delete | 查看，创建，更新，删除 | 主机分组 |
| /api/cmdb/server/ | get,post,put,delete | 查看，创建，更新，删除 | 服务器 |
| /api/cmdb/create\_host | post | 创建 | 新建主机 |
| /api/cmdb/host\_collect | get | SSH连接采集主机配置 | SSH连接采集主机配置 |
| /api/cmdb/excel\_create\_host | get，post | 下载excel模板文件，提交文件 | excel导入主机 |
| /api/cmdb/tencent\_cloud | get | 调用腾讯云ECS API获取 | 腾讯云云主机导入 |
| /api/cmdb/aliyun\_cloud | get | 调用阿里云ECS API获取 | 阿里云云主机导入 |

## 3.2 API平台雏形

### 1、Django基本配置

创建应用：

python manage.py startapp cmdb

添加APP：

INSTALLED\_APPS = [  
 ...  
 'rest\_framework',  
 'cmdb',  
]

在settings.py配置mysql数据库：

DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.mysql',  
 'NAME': 'devops',  
 'USER': 'root',  
 'PASSWORD': '123456',  
 'HOST': '192.168.31.62',  
 'PORT': '3306',  
 }  
}

使用pip工具安装：

pip3 install pymysql

指定数据库驱动，在devops\_api/\_\_init\_\_.py文件，添加如下：

import pymysql  
pymysql.install\_as\_MySQLdb()

使用docker启动一个mysql实例：

docker run -d \

--name db \

-p 3306:3306 \

-v mysqldata:/var/lib/mysql \

-e MYSQL\_ROOT\_PASSWORD=123456 \

mysql:5.7 --character-set-server=utf8

### 2、定义数据库模型

cmdb/models.py

from django.db import models

class Idc(models.Model):

name = models.CharField(max\_length=30, unique=True, verbose\_name="机房名称")

city = models.CharField(max\_length=20, verbose\_name="城市")

provider = models.CharField(max\_length=30, verbose\_name="运营商")

note = models.TextField(blank=True, null=True, verbose\_name="备注")

create\_time = models.DateTimeField(auto\_now\_add=True, verbose\_name="创建时间")

class Meta:

db\_table = "cmdb\_idc"

verbose\_name\_plural = "IDC机房"

ordering = ('-id',)

def \_\_str\_\_(self):

return self.name

class ServerGroup(models.Model):

name = models.CharField(max\_length=30, unique=True, verbose\_name="分组名称")

note = models.TextField(blank=True, null=True, verbose\_name="备注")

create\_time = models.DateTimeField(auto\_now\_add=True, verbose\_name="创建时间")

class Meta:

db\_table = "cmdb\_server\_group"

verbose\_name\_plural = "主机分组"

ordering = ('-id',)

def \_\_str\_\_(self):

return self.name

class Server(models.Model):

idc = models.ForeignKey(Idc, on\_delete=models.PROTECT, verbose\_name="IDC机房")

server\_group = models.ManyToManyField(ServerGroup, default="Default", verbose\_name="主机分组")

name = models.CharField(max\_length=30, blank=True, verbose\_name="名称")

hostname = models.CharField(max\_length=30, unique=True, verbose\_name="主机名")

ssh\_ip = models.GenericIPAddressField(verbose\_name="SSH IP")

ssh\_port = models.IntegerField(verbose\_name="SSH端口")

note = models.TextField(blank=True, null=True, verbose\_name="备注")

machine\_type = models.CharField(max\_length=30, blank=True, choices=(('vm','虚拟机'),('cloud\_vm','云主机'),('physical\_machine','物理机')), verbose\_name="机器类型")

os\_version = models.CharField(max\_length=50, blank=True, null=True, verbose\_name="系统版本")

public\_ip = models.CharField(max\_length=100, blank=True, null=True, verbose\_name="公网IP")

private\_ip = models.CharField(max\_length=100, blank=True, null=True, verbose\_name="内网IP")

cpu\_num = models.CharField(max\_length=10, blank=True, null=True, verbose\_name="CPU")

cpu\_model = models.CharField(max\_length=100, blank=True, null=True, verbose\_name="CPU型号")

memory = models.CharField(max\_length=30, blank=True, null=True, verbose\_name="内存")

disk = models.CharField(max\_length=200,blank=True, null=True, verbose\_name="硬盘")

put\_shelves\_date = models.DateField(null=True, blank=True, verbose\_name="上架日期")

off\_shelves\_date = models.DateField(null=True, blank=True, verbose\_name="下架日期")

expire\_datetime = models.DateTimeField(blank=True, null=True, verbose\_name="租约过期时间")

is\_verified = models.CharField(max\_length=10, blank=True, choices=(('verified','已验证'),('unverified','未验证')), default='unverified', verbose\_name="SSH验证状态")

update\_time = models.DateTimeField(auto\_now\_add=True, verbose\_name="更新时间")

create\_time = models.DateTimeField(auto\_now\_add=True, verbose\_name="创建时间")

class Meta:

db\_table = "cmdb\_server"

verbose\_name\_plural = "主机管理"

ordering = ('-id',)

def \_\_str\_\_(self):

return self.hostname

生成数据库：

python manage.py makemigrations

python manage.py migrate

### 3、创建序列化文件

cmdb/serializers.py

from cmdb.models import Idc, ServerGroup, Server   
from rest\_framework import serializers  
  
class IdcSerializer(serializers.ModelSerializer):  
 """  
 IDC机房序列化类  
 """  
 class Meta:  
 model = Idc  
 fields = "\_\_all\_\_"  
 read\_only\_fields = ("id",) # 仅用于序列化（只读）字段，反序列化（更新）可不传  
  
class ServerGroupSerializer(serializers.ModelSerializer):  
 """  
 主机分组序列化类  
 """  
  
 class Meta:  
 model = ServerGroup  
 fields = "\_\_all\_\_"  
 read\_only\_fields = ("id",)   
  
class ServerSerializer(serializers.ModelSerializer):  
 """  
 服务器序列化类  
 """  
 # idc = IdcSerializer() # 一对多  
  
 class Meta:  
 model = Server  
 fields = "\_\_all\_\_"  
 read\_only\_fields = ("id",)

### 4、定义视图

cmdb/views.py

from rest\_framework.viewsets import ModelViewSet

# 序列化

from cmdb.models import Idc, ServerGroup, Server

from cmdb.serializers import IdcSerializer, ServerGroupSerializer, ServerSerializer

class IdcViewSet(ModelViewSet):

queryset = Idc.objects.all()

serializer\_class = IdcSerializer

class ServerGroupViewSet(ModelViewSet):

queryset = ServerGroup.objects.all()

serializer\_class = ServerGroupSerializer

class ServerViewSet(ModelViewSet):

queryset = Server.objects.all()

serializer\_class = ServerSerializer

### 5、定义路由

devops\_api/urls.py

from django.contrib import admin  
from django.urls import path, include  
  
urlpatterns = [  
 # path('admin/', admin.site.urls),  
]  
  
from cmdb.views import IdcViewSet, ServerGroupViewSet, ServerViewSet  
from rest\_framework import routers  
router = routers.DefaultRouter()  
router.register(r'idc', IdcViewSet, basename="idc")  
router.register(r'server\_group', ServerGroupViewSet, basename="server\_group")  
router.register(r'server', ServerViewSet, basename="server")  
  
urlpatterns += [  
 path('api/cmdb/', include(router.urls))  
]

### 6、凭据管理

创建应用：

python manage.py startapp system\_config

添加APP：

INSTALLED\_APPS = [  
 ...  
 'cmdb',

'system\_config',  
]

system\_config/models.py

from django.db import models  
  
class Credential(models.Model):  
 auth\_choice = (  
 (1, "密码"),  
 (2, "秘钥")  
 )  
 name = models.CharField(max\_length=30, verbose\_name="凭据名称")  
 username = models.CharField(max\_length=20, verbose\_name="用户名")  
 auth\_mode = models.IntegerField(choices=auth\_choice, default=1, verbose\_name="认证方式")  
 password = models.CharField(max\_length=50, blank=True, verbose\_name="密码")  
 private\_key = models.TextField(blank=True, verbose\_name="私钥")  
 note = models.TextField(blank=True, verbose\_name="备注")  
 create\_time = models.DateTimeField(auto\_now\_add=True, verbose\_name="创建时间")  
 update\_time = models.DateTimeField(auto\_now=True, verbose\_name="更新时间")  
  
 class Meta:  
 db\_table = "system\_config\_credential"  
 verbose\_name\_plural = "凭据管理"  
 ordering = ('-id',)  
  
 def \_\_str\_\_(self):  
 return self.name

序列化

from rest\_framework import serializers  
from system\_config.models import Credential  
  
class CredentialSerializer(serializers.ModelSerializer):  
  
 class Meta:  
 model = Credential  
 fields = "\_\_all\_\_"  
 read\_only\_fields = ("id",)

视图

from rest\_framework.viewsets import ModelViewSet  
from system\_config.serializers import CredentialSerializer  
from system\_config.models import Credential  
  
class CredentialViewSet(ModelViewSet):  
  
 queryset = Credential.objects.all()  
 serializer\_class = CredentialSerializer

路由

在下面增加：

from system\_config.views import CredentialViewSet  
router = routers.DefaultRouter()  
router.register(r'credential', CredentialViewSet, basename="credential")  
urlpatterns += [  
 path('api/config/', include(router.urls))  
]

### 7、增加分页、过滤、搜索、排序

#### 分页

定义分页与修改分页返回结果。

devops\_api/libs/pagination.py

from rest\_framework.pagination import PageNumberPagination

from rest\_framework.response import Response

from collections import OrderedDict

class MyPagination(PageNumberPagination):

page\_size = 6 # 默认每页显示多少条

page\_query\_param = 'page\_num' # 指定查询第几页（页码），默认 page

page\_size\_query\_param = 'page\_size' # 定义每页显示多少条

max\_page\_size = 50 # 每页最多显示多少条

def get\_paginated\_response(self, data):

code = 200

msg = "成功"

return Response(OrderedDict([

('code', code),

('msg', msg),

('count', self.page.paginator.count),

# ('next', self.get\_next\_link()),

# ('previous', self.get\_previous\_link()),

('data', data)

]))

settings.py

# DRF配置（针对所有API有效）  
REST\_FRAMEWORK = {  
 'DEFAULT\_PAGINATION\_CLASS': 'libs.pagination.MyPagination'  
}

#### 过滤、搜索和排序

pip install django-filter

添加APP：

INSTALLED\_APPS = [

...

'django\_filters',

]

from rest\_framework import filters  
from django\_filters.rest\_framework import DjangoFilterBackend  
  
class IdcViewSet(ModelViewSet):  
 queryset = Idc.objects.all()  
 serializer\_class = IdcSerializer  
 filter\_backends = [filters.SearchFilter, filters.OrderingFilter, DjangoFilterBackend] # 指定过滤器  
 search\_fields = ('name',) # 指定可搜索的字段  
 filter\_fields = ('city',)

### 8、启用Token认证

#### 登录

settings.py

添加APP：

INSTALLED\_APPS = [

...

'rest\_framework.authtoken'

...

]

REST\_FRAMEWORK = {  
 # 认证  
 'DEFAULT\_AUTHENTICATION\_CLASSES': [  
 'rest\_framework.authentication.SessionAuthentication',  
 'rest\_framework.authentication.TokenAuthentication',  
 ],  
 # 权限  
 'DEFAULT\_PERMISSION\_CLASSES': [  
 'rest\_framework.permissions.IsAuthenticated' # 登录后就能访问所有API  
 ],  
}

devops\_api/urls.py

re\_path('^api/login/$', token\_auth.CustomAuthToken.as\_view()),

devops\_api/libs/token\_auth.py

from rest\_framework.authtoken.views import ObtainAuthToken  
from rest\_framework.authtoken.models import Token  
from rest\_framework.response import Response

class CustomAuthToken(ObtainAuthToken):  
  
 def post(self, request, \*args, \*\*kwargs):  
 serializer = self.serializer\_class(data=request.data,  
 context={'request': request})  
 if serializer.is\_valid():  
 user = serializer.validated\_data['user']  
 token, created = Token.objects.get\_or\_create(user=user)  
 res = {'code': 200,'msg': '认证成功','token': token.key,'username': user.username,}  
 return Response(res)  
 else:  
 res = {'code': 500,'msg': '用户名或密码错误！',}  
 return Response(res)

生成authtoken\_token表

python manage.py migrate

登录api忽略认证。

from rest\_framework.permissions import AllowAny

permission\_classes = (AllowAny,) # AllowAny 允许所有用户（登录不需要身份认证）

#### 退出

前端删除本地token即可。

#### 修改密码

devops\_api/urls.py

from libs import token\_auth

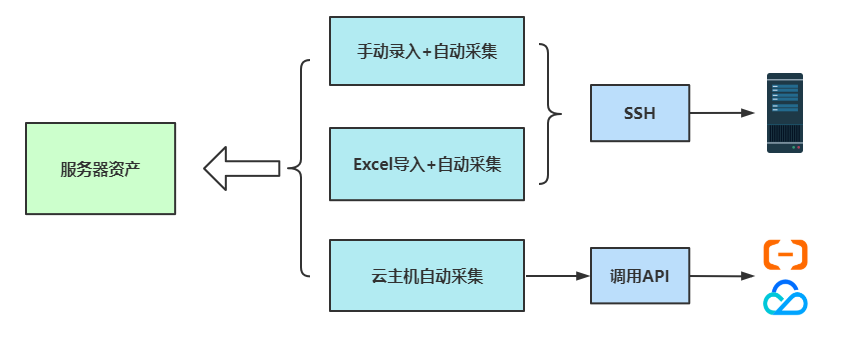
re\_path('^api/change\_password/$', token\_auth.ChangeUserPasswordView.as\_view()),

我们是基于django用户认证实现登录，因此判断密码是否正确和修改密码需要使用django自带的make\_password加密和check\_password验证。

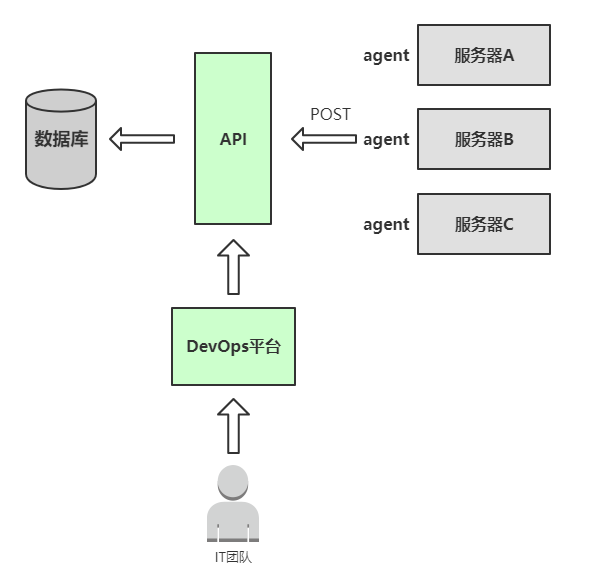
devops\_api/libs/token\_auth.py

from rest\_framework.views import APIView  
from django.contrib.auth.models import User  
from django.contrib.auth.hashers import make\_password, check\_password  
from rest\_framework.permissions import AllowAny  
class ChangeUserPasswordView(APIView):  
 permission\_classes = (AllowAny,) # AllowAny 允许所有用户（登录不需要身份认证）  
 def post(self, request):  
 # username = request.user  
 username = 'aliang'  
 old\_password = request.data.get("old\_password")  
 new\_password = request.data.get("new\_password")  
 try:  
 user = User.objects.get(username=username)  
 except:  
 res = {'code': 500, 'msg': '用户不存在！'}  
 return Response(res)  
  
 if check\_password(old\_password, user.password):  
 user.password = make\_password(new\_password)  
 user.save()  
 res = {'code': 200, 'msg': '修改密码成功'}  
 else:  
 res = {'code': 500, 'msg': '原密码不正确！'}  
 return Response(res)

## 3.3 服务器信息采集



前端采集功能实现流程图



服务器自动上报或主动采集工作流程图

### 1、采集方式

* Agent方式：在每台服务器部署，周期采集并提交API。也可以下发任务。速度快。
  + 缺点：提前部署
  + 应用场景：适合服务器数量多
* SSH访问：通过paramiko连接各个机器，执行命令，获取数据并提交API。
  + 缺点：慢
  + 应用场景：适合服务器数量少
* Ansible类工具：也是基于ssh通信，功能完善，速度快，开发成本低。
  + 缺点：依赖工具
  + 应用场景：适合熟悉ansible的

### 2、服务器配置采集脚本

#### 采集内容及采集方式

|  |  |  |
| --- | --- | --- |
| 字段 | 名称 | 获取方式 |
| hostname | 主机名 | socket模块获取 |
| machine\_type | 机器类型（虚拟机、云主机、物理机） | 从dmesg中提取标识 |
| os\_version | 系统版本 | /etc/issue |
| public\_ip | 公网IP地址 | 调用接口判断公网还是内网（用户也会传递），云主机无需判断 |
| [intranet](javascript:;)\_ip | 内网IP地址 |
| cpu\_num | CPU数量 | /proc/cpuinfo |
| cpu\_model | CPU型号 | /proc/cpuinfo |
| memory | 内存 | /proc/meminfo |
| disk | 硬盘 | lsblk |
| put\_shelves\_date | 上架日期 | 默认以系统启动时间，后期人工再改 |

#### 采集脚本

#!/usr/bin/python  
# coding: utf-8  
# describe：CMDB采集脚本，对python版本和执行用户没要求  
# 解决python执行编码问题  
import sys  
try:  
 reload(sys) # py3没有  
 sys.setdefaultencoding('utf8')  
except:  
 pass  
  
import socket, fcntl, struct  
from datetime import datetime, date, timedelta  
import os, json  
  
try:  
 from urllib import request  
except:  
 import urllib2 as request  
import logging  
  
# 当前目录  
BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))  
  
# 日志配置  
log\_file = os.path.join(BASE\_DIR, "collect.log")  
logging.basicConfig(level=logging.INFO,filename=log\_file,format="%(asctime)s - [%(levelname)s] %(message)s")  
  
def get\_ip\_address(nic):  
 s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
 socket.inet\_ntoa(fcntl.ioctl(s.fileno(),0x8915,struct.pack('256s', nic[:15]))[20:24])  
  
class GetData():  
 def \_\_init\_\_(self):  
 self.result = {}  
 # 解析文件  
 def parse\_file(self, file, name):  
 with open(file) as f:  
 for line in f.readlines():  
 key, value = line.split(":")  
 key = key.strip()  
 value = value.strip()  
 if key == name:  
 return value  
 def hostname(self):  
 hostname = socket.gethostname()  
 hostname\_backup = '/tmp/.hostname'  
 if os.path.isfile(hostname\_backup) and os.path.getsize(hostname\_backup) != 0:  
 with open(hostname\_backup) as f:  
 hostname = f.read().strip()  
 else:  
 with open(hostname\_backup, 'w') as f:  
 f.write(hostname)  
 return hostname  
 def machine\_type(self):  
 result = os.popen("dmesg |grep -i virtual |grep -ci hardware")  
 if int(result.read()) >= 1:  
 type = "physical\_machine" # 物理机  
 else:  
 result = os.popen("dmesg |grep -i virtual |grep -ci kvm")  
 if int(result.read()) >= 1:  
 type = "cloud\_vm" # 云主机  
 else:  
 type = "vm" # 虚拟机  
 return type  
 # 获取系统版本，兼容centos7和Ubuntu  
 def os\_version(self):  
 with open("/etc/issue") as f:  
 if f.readline().strip() == "\S":  
 with open("/etc/redhat-release") as f:  
 os\_version = f.readline().strip()  
 else:  
 os\_version = f.readline().strip()  
 return os\_version  
 # 系统启动时间  
 def system\_up\_time(self):  
 with open("/proc/uptime") as f:  
 s = f.read().split(".")[0] # 启动有多少秒  
 up\_time = datetime.now() - timedelta(seconds=float(s)) # 当前时间减去启动秒  
 return date.strftime(up\_time, '%Y-%m-%d')  
 def public\_ip(self):  
 private\_ip = self.private\_ip()  
 ip\_api\_url = ['http://ip.renfei.net', 'http://ifconfig.me/ip']  
 ip\_list = []  
 try:  
 req = request.Request(url=ip\_api\_url[0])  
 res = request.urlopen(req)  
 ip = json.loads(res.read().decode())['clientIP']  
 except:  
 req = request.Request(url=ip\_api\_url[1])  
 res = request.urlopen(req)  
 ip = res.read().decode()  
 if ip in private\_ip:  
 ip.append(ip)  
 return ip\_list  
 else:  
 ip\_list.append('%s(NAT)' %ip)  
 return ip\_list  
 def private\_ip(self):  
 nic\_prefix = ['eth', 'en', 'em'] # 常见网卡名前缀  
 ip\_list = []  
 with open("/proc/net/dev") as f:  
 for s in f.readlines():  
 name = s.split(':')[0].strip()  
 for p in nic\_prefix:  
 if name.startswith(p):  
 result = os.popen("ip addr show %s |awk -F'[ /]' '/inet /{print $6}'" %name)  
 ip\_list.append(result.read().strip())  
 return ip\_list  
 def cpu\_num(self):  
 cpu = self.parse\_file("/proc/cpuinfo", "cpu cores")  
 return "%s核" %cpu  
 def cpu\_model(self):  
 model = self.parse\_file("/proc/cpuinfo", "model name")  
 return model  
 def memory(self):  
 total = self.parse\_file("/proc/meminfo", "MemTotal")  
 total = round(float(total.split()[0]) / 1024 / 1024, 1) # 转GB单位  
 return "%sG" %total  
 def disk(self):  
 disk = []  
 result = os.popen("lsblk |awk '$6~/disk/{print $1,$4,$5}'")  
 for d in result.read().strip().split('\n'):  
 d = d.split()  
 device = d[0]  
 size = d[1]  
 type = "HDD" if d[2] == 0 else "SSD"  
 disk.append({'device': '/dev/%s' %device, 'size': size, 'type': type})  
 return disk  
 def get\_all(self):  
 """  
 这里字段必须与API对应  
 """  
 self.result = {  
 "hostname": self.hostname(),  
 "machine\_type": self.machine\_type(),  
 "os\_version": self.os\_version(),  
 # "public\_ip": self.public\_ip(),  
 "private\_ip": self.private\_ip(),  
 "cpu\_num": self.cpu\_num(),  
 "cpu\_model": self.cpu\_model(),  
 "memory": self.memory(),  
 "disk": self.disk(),  
 "put\_shelves\_date": self.system\_up\_time(), # 上架时间默认设置系统启动时间  
 }  
 json\_data = json.dumps(self.result)  
 return json\_data  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 data = GetData()  
 try:  
 print(data.get\_all())  
 except Exception as e:  
 result = {'code': 500, 'msg': '采集脚本执行失败！错误：%s' %e}  
 print(json.dumps(result))

#### 采集脚本什么时候工作？

* **新建主机并同步实现（SSH）**

填写基本信息，确保主机名与目标主机一致->点击确认->请求测试接口(带上凭据id)，不通先关闭窗口，提示要操作什么，例如检查ip和端口，通的话修改数据库字段已验证，请求调用采集接口自动上报。

* **管理员点击同步实现（SSH）**
* **周期性自动执行上报（Agent）**
* 在第一次新建主机时候上传脚本并配置定时任务。
* 在装机后系统初始化自动配置
* 后期用ansible批量主机配置

### 3、在远程主机执行命令和上传文件

有了采集脚本，接下来就是如何让脚本能目标主机执行进行采集并获取入库。

这里采用paramiko实现ssh连接目标主机并执行采集脚本。

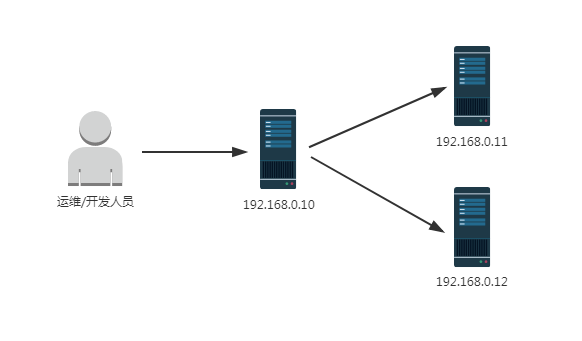
paramiko模块是基于Python实现的SSH远程安全连接，用于SSH远程执行命令、文件传输等功能。

首先pip安装：

pip3 install paramiko

为更好学习该模块，我们下面写几个具体的示例来熟悉它的常用用法。

拓扑图：



#### SSH密码认证远程执行命令

import paramiko

hostname = '192.168.0.11'

port = 22

username = 'root'

password = '123456'

# 绑定实例

ssh = paramiko.SSHClient()

# AutoAddPolicy()自动添加主机keys

ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

# 连接主机信息

ssh.connect(hostname, port, username, password, timeout=5)

# 执行Shell命令，结果分别保存在标准输入，标准输出和标准错误

stdin, stdout, stderr = ssh.exec\_command('ls -l')

stdout = stdout.read()

error = stderr.read()

# 判断stderr输出是否为空，为空则打印运行结果，不为空打印报错信息

if not error:

print(stdout)

else:

print(error)

ssh.close()

#### SSH密钥认证远程执行命令

口令是普遍的鉴权策略，为了提高安全性，还会用密钥对认证。

首选生成秘钥对：

# ssh-keygen # 采用默认配置，一直回车即可

# ls .ssh/

id\_rsa id\_rsa.pub

将id\_rsa.pub公钥追加到目标服务器/root/.ssh/authorized\_keys文件中。

import paramiko

import sys

hostname = '192.168.0.11'

port = 22

username = 'root'

key\_file = '/root/.ssh/id\_rsa'

# 将列表元素以空格拼接

cmd = " ".join(sys.argv[1:])

def ssh\_command(command):

ssh = paramiko.SSHClient()

# 指定key文件

key = paramiko.RSAKey.from\_private\_key\_file(key\_file)

ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

# 使用key登录

ssh.connect(hostname, port, username, pkey=key)

stdin, stdout, stderr = ssh.exec\_command(command)

result = stdout.read()

error = stderr.read()

if not error:

print(result)

else:

print(error)

ssh.close()

if \_\_name\_\_ == "\_\_main\_\_":

ssh\_command (cmd)

#### 上传文件到远程服务器

import os, sys

import paramiko

hostname = '192.168.0.11'

port = 22

username = 'root'

password = '123456'

local\_path = '/root/test.txt'

remote\_path = '/opt/test.txt'

try:

s = paramiko.Transport((hostname, port))

s.connect(username = username, password=password)

#key = paramiko.RSAKey.from\_private\_key(key\_file)

#transport.connect(username=username, pkey=key)

except Exception as e:

print(e)

sftp = paramiko.SFTPClient.from\_transport(s)

# 使用put()方法把本地文件上传到远程服务器

sftp.put(local\_path, remote\_path)

#### 封装SSH模块验证

import paramiko  
from io import StringIO # py2 from StringIO import StringIO  
import os  
  
class SSH():  
 def \_\_init\_\_(self, ip, port, username, password=None, key=None):  
 self.ip = ip  
 self.port = port  
 self.username = username  
 self.password = password  
 self.key = key  
 def command(self, shell):  
 # 绑定实例  
 ssh = paramiko.SSHClient()  
 # 允许连接不在known\_hosts文件上的主机  
 ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())  
 try:  
 if self.password:  
 ssh.connect(hostname=self.ip, port=self.port, username=self.username, password=self.password, timeout=5)  
 else:  
 cache = StringIO(self.key) # 将字符串通过StringIO转为file对象（self.key内容是从数据库查询的文本）  
 key = paramiko.RSAKey.from\_private\_key(cache) # 接收file对象  
 # 使用key登录  
 ssh.connect(hostname=self.ip, port=self.port, username=self.username, pkey=key)  
 # 执行Shell命令，结果分别保存在标准输入，标准输出和标准错误  
 stdin, stdout, stderr = ssh.exec\_command(shell)  
 stdout = stdout.read()  
 error = stderr.read()  
 # 判断stderr输出是否为空，为空则打印运行结果，不为空打印报错信息  
 ssh.close()  
 if not error:  
 return stdout  
 else:  
 return error  
 except Exception as e:  
 return "SSH连接失败：%s" % e  
  
 def scp(self, local\_file, remote\_file):  
 # 绑定实例  
 ts = paramiko.Transport((self.ip, self.port))  
 try:  
 if self.password:  
 ts.connect(username=self.username, password=self.password)  
 else:  
 cache = StringIO(self.key)  
 key = paramiko.RSAKey.from\_private\_key(cache)  
 ts.connect(username=self.username, pkey=key)  
 sftp = paramiko.SFTPClient.from\_transport(ts)  
 try:  
 sftp.put(localpath=local\_file, remotepath=remote\_file)  
 ts.close()  
 return "上传文件成功"  
 except Exception as e:  
 return "上传文件失败：%s" %e  
 except Exception as e:  
 return "SSH连接失败：%s" % e  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 ssh = SSH('192.168.31.73', 22, 'root', '123.com')  
 local\_file=os.path.join(os.getcwd(),'client\_collect\_agent.py')  
 result = ssh.command('ls -l')  
 # result = ssh.scp(local\_file, '/tmp/agent.py')  
 print(result)

视图：

from system\_config.models import Credential  
from utils.ssh import SSH  
class SshConnView(APIView):  
 def get(self, request):  
  
 # 前端传递  
 ip = "192.168.31.73"  
 port = 22  
 username = "root"  
 credential\_id = '7452496b-21a1-4f5f-be78-e3d7094738c8'  
  
 # 获取密码或者私钥  
 secret = Credential.objects.get(credential\_id=credential\_id)  
 password = secret.password # 解密  
 private\_key = secret.private\_key  
 if secret.password:  
 print('密码')  
 ssh = SSH(ip, port, username, password=password)  
 result = ssh.command('ls')  
 else:  
 print('秘钥')  
 ssh = SSH(ip, port, username, key=private\_key)  
 result = ssh.command('ls')  
 res = {'code': 200, 'msg': '成功', 'data': result}  
 return Response(res)

### 4、新建主机功能

#### 路由

from cmdb.views import ExcelCreateHostView, CreateHostView  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 re\_path('^api/login/$', token\_auth.CustomAuthToken.as\_view()),  
 re\_path('^api/docs/$', schema\_view),  
 re\_path('^api/cmdb/create\_host/$', CreateHostView.as\_view()),  
 re\_path('^api/cmdb/excel\_create\_host/$', ExcelCreateHostView.as\_view()),  
]

#### 视图

class CreateHostView(APIView):  
 def post(self, request):  
 idc\_id = int(request.data.get('idc')) # 机房id  
 server\_group\_id\_list = request.data.get('server\_group') # 分组id  
 name = request.data.get('name')  
 hostname = request.data.get('hostname')  
 ssh\_ip = request.data.get('ssh\_ip')  
 ssh\_port = int(request.data.get('ssh\_port'))  
 credential\_id = int(request.data.get('credential'))  
 note = request.data.get('note')  
  
 # 如果主机存在返回  
 server = Server.objects.filter(hostname=hostname)  
 if server:  
 result = {'code': 500, 'msg': '主机已存在！'}  
 return Response(result)  
  
 # 通过凭据ID获取用户名信息  
 credential = Credential.objects.get(id=credential\_id)  
 username = credential.username  
 if credential.auth\_mode == 1:  
 password = credential.password  
 ssh = SSH(ssh\_ip, ssh\_port, username, password=password)  
 else:  
 private\_key = credential.private\_key  
 ssh = SSH(ssh\_ip, ssh\_port, username, key=private\_key)  
  
 test = ssh.test() # 测试SSH连接通过  
 if test['code'] == 200:  
 client\_agent\_name = "host\_collect.py"  
 local\_file = os.path.join(settings.BASE\_DIR, 'cmdb', 'files', client\_agent\_name)  
 remote\_file = os.path.join(settings.CLIENT\_COLLECT\_DIR, client\_agent\_name) # 这个工作路径在setting里配置  
 ssh.scp(local\_file, remote\_file=remote\_file)  
 ssh.command('chmod +x %s' % remote\_file)  
 result = ssh.command('python %s' % remote\_file)  
  
 if result['code'] == 200: # 采集脚本执行成功  
 data = json.loads(result['data'])  
  
 if hostname != data['hostname']:  
 result = {'code': 500, 'msg': '填写的主机名与目标主机不一致，请核对后再提交！'}  
 return Response(result)  
  
 # 1.基本主机信息入库（人工录入）  
 idc = Idc.objects.get(id=idc\_id) # 根据id查询IDC  
 server\_obj = Server.objects.create(  
 idc=idc,  
 name=name if name else hostname,  
 hostname=hostname,  
 ssh\_ip=ssh\_ip,  
 ssh\_port=ssh\_port,  
 is\_verified='verified',

credential=credential,  
 note=note  
 )  
 # 添加对对多字段  
 for group\_id in server\_group\_id\_list:  
 group = ServerGroup.objects.get(id=group\_id) # 根据id查询分组  
 server\_obj.server\_group.add(group) # 将服务器添加到分组  
  
 # 2.主机配置入库（自动采集）  
 server.update(\*\*data)  
 result = {'code': 200, 'msg': '添加主机成功并同步配置'}  
 else:  
 result = {'code': 500, 'msg': '采集主机配置失败！错误：%s' % result['msg']}  
 else:  
 result = {'code': 500, 'msg': 'SSH连接异常！错误：%s' % test['msg']}  
  
 return Response(result)

#### 采集脚本

cmdb/files/host\_collect.py

#!/usr/bin/python  
# coding: utf-8  
# describe：CMDB采集脚本，对python版本和执行用户没要求  
# 解决python执行编码问题  
import sys  
try:  
 reload(sys) # py3没有  
 sys.setdefaultencoding('utf8')  
except:  
 pass  
  
import socket, fcntl, struct  
from datetime import datetime, date, timedelta  
import os, json  
  
try:  
 from urllib import request  
except:  
 import urllib2 as request  
import logging  
  
# 当前目录  
BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))  
  
# 日志配置  
log\_file = os.path.join(BASE\_DIR, "collect.log")  
logging.basicConfig(level=logging.INFO,filename=log\_file,format="%(asctime)s - [%(levelname)s] %(message)s")  
  
def get\_ip\_address(nic):  
 s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
 socket.inet\_ntoa(fcntl.ioctl(s.fileno(),0x8915,struct.pack('256s', nic[:15]))[20:24])  
  
class GetData():  
 def \_\_init\_\_(self):  
 self.result = {}  
 # 解析文件  
 def parse\_file(self, file, name):  
 with open(file) as f:  
 for line in f.readlines():  
 key, value = line.split(":")  
 key = key.strip()  
 value = value.strip()  
 if key == name:  
 return value  
 def hostname(self):  
 hostname = socket.gethostname()  
 hostname\_backup = '/tmp/.hostname'  
 if os.path.isfile(hostname\_backup) and os.path.getsize(hostname\_backup) != 0:  
 with open(hostname\_backup) as f:  
 hostname = f.read().strip()  
 else:  
 with open(hostname\_backup, 'w') as f:  
 f.write(hostname)  
 return hostname  
 def machine\_type(self):  
 result = os.popen("dmesg |grep -i virtual |grep -ci hardware")  
 if int(result.read()) >= 1:  
 type = "physical\_machine" # 物理机  
 else:  
 result = os.popen("dmesg |grep -i virtual |grep -ci kvm")  
 if int(result.read()) >= 1:  
 type = "cloud\_vm" # 云主机  
 else:  
 type = "vm" # 虚拟机  
 return type  
 # 获取系统版本，兼容centos7和Ubuntu  
 def os\_version(self):  
 with open("/etc/issue") as f:  
 if f.readline().strip() == "\S":  
 with open("/etc/redhat-release") as f:  
 os\_version = f.readline().strip()  
 else:  
 os\_version = f.readline().strip()  
 return os\_version  
 # 系统启动时间  
 def system\_up\_time(self):  
 with open("/proc/uptime") as f:  
 s = f.read().split(".")[0] # 启动有多少秒  
 up\_time = datetime.now() - timedelta(seconds=float(s)) # 当前时间减去启动秒  
 return date.strftime(up\_time, '%Y-%m-%d')  
 def public\_ip(self):  
 private\_ip = self.private\_ip()  
 ip\_api\_url = ['http://ip.renfei.net', 'http://ifconfig.me/ip']  
 ip\_list = []  
 try:  
 req = request.Request(url=ip\_api\_url[0])  
 res = request.urlopen(req)  
 ip = json.loads(res.read().decode())['clientIP']  
 except:  
 req = request.Request(url=ip\_api\_url[1])  
 res = request.urlopen(req)  
 ip = res.read().decode()  
 if ip in private\_ip:  
 ip.append(ip)  
 return ip\_list  
 else:  
 ip\_list.append('%s(NAT)' %ip)  
 return ip\_list  
 def private\_ip(self):  
 nic\_prefix = ['eth', 'en', 'em'] # 常见网卡名前缀  
 ip\_list = []  
 with open("/proc/net/dev") as f:  
 for s in f.readlines():  
 name = s.split(':')[0].strip()  
 for p in nic\_prefix:  
 if name.startswith(p):  
 result = os.popen("ip addr show %s |awk -F'[ /]' '/inet /{print $6}'" %name)  
 ip\_list.append(result.read().strip())  
 return ip\_list  
 def cpu\_num(self):  
 cpu = self.parse\_file("/proc/cpuinfo", "cpu cores")  
 return "%s核" %cpu  
 def cpu\_model(self):  
 model = self.parse\_file("/proc/cpuinfo", "model name")  
 return model  
 def memory(self):  
 total = self.parse\_file("/proc/meminfo", "MemTotal")  
 total = round(float(total.split()[0]) / 1024 / 1024, 1) # 转GB单位  
 return "%sG" %total  
 def disk(self):  
 disk = []  
 result = os.popen("lsblk |awk '$6~/disk/{print $1,$4,$5}'")  
 for d in result.read().strip().split('\n'):  
 d = d.split()  
 device = d[0]  
 size = d[1]  
 type = "HDD" if d[2] == 0 else "SSD"  
 disk.append({'device': '/dev/%s' %device, 'size': size, 'type': type})  
 return disk  
 def get\_all(self):  
 """  
 这里字段必须与API对应  
 """  
 self.result = {  
 "hostname": self.hostname(),  
 "machine\_type": self.machine\_type(),  
 "os\_version": self.os\_version(),  
 # "public\_ip": self.public\_ip(),  
 "private\_ip": self.private\_ip(),  
 "cpu\_num": self.cpu\_num(),  
 "cpu\_model": self.cpu\_model(),  
 "memory": self.memory(),  
 "disk": self.disk(),  
 "put\_shelves\_date": self.system\_up\_time(), # 上架时间默认设置系统启动时间  
 }  
 json\_data = json.dumps(self.result)  
 return json\_data  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 data = GetData()  
 try:  
 print(data.get\_all())  
 except Exception as e:  
 result = {'code': 500, 'msg': '采集脚本执行失败！错误：%s' %e}  
 print(json.dumps(result))

### 5、Excel导入功能

安装excel解析模块：

pip install xlrd==1.2.0

from cmdb.views import ExcelCreateHostView  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 re\_path('^api/login/$', token\_auth.CustomAuthToken.as\_view()),  
 re\_path('^api/docs/$', schema\_view),  
 re\_path('^api/cmdb/excel\_create\_host/$', ExcelCreateHostView.as\_view())  
]

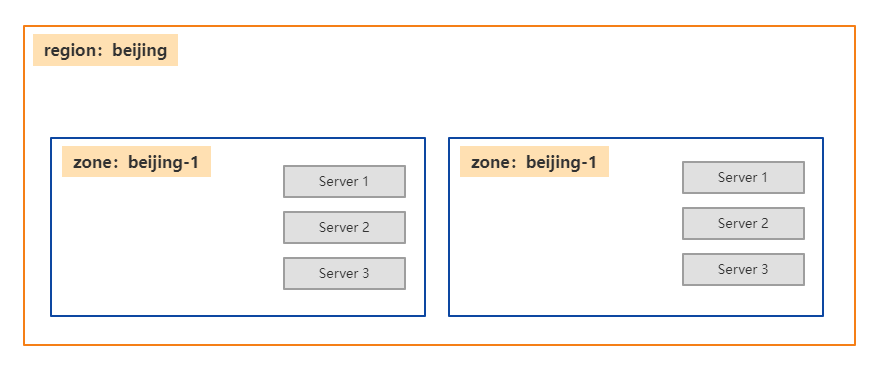
class ExcelCreateHostView(APIView):  
 # 下载主机导入模板.xlsx  
 def get(self, request):  
 from django.conf import settings  
 import os  
 from django.http import FileResponse  
 file\_name = 'host.xlsx'  
 file\_path = os.path.join(settings.BASE\_DIR, 'cmdb', 'files', file\_name)  
 response = FileResponse(open(file\_path, 'rb'))  
 response['Content-Type'] = 'application/octet-stream'  
 response['Content-Disposition'] = 'attachment; filename=%s' %file\_name  
 # result = {'code': 200, 'msg': '获取文件成功'}  
 return response  
 # 导入  
 def post(self, request):  
 import xlrd  
 excel\_file\_obj = request.data['file']  
 idc\_id = int(request.data.get('idc'))  
 server\_group\_id = int(request.data.get('server\_group'))  
 try:  
 data = xlrd.open\_workbook(filename=None, file\_contents=excel\_file\_obj.read())  
 except Exception:  
 result = {'code': 500, 'msg': '请上传Excel文件！'}  
 return Response(result)  
 table = data.sheets()[0] # 打开第一个工作表  
 nrows = table.nrows # 获取表的行数  
 # ncole = table.ncols # 获取列数  
  
 idc = Idc.objects.get(id=idc\_id)  
 server\_group = ServerGroup.objects.get(id=server\_group\_id)  
 try:  
 for i in range(nrows): # 循环行  
 if i != 0: # 跳过标题行  
 name = table.row\_values(i)[0]  
 hostname = table.row\_values(i)[1]  
 ssh\_ip = table.row\_values(i)[2]  
 ssh\_port = table.row\_values(i)[3]  
 note = table.row\_values(i)[4]  
  
 server = Server.objects.create(  
 idc=idc,  
 name=name,  
 hostname=hostname,  
 ssh\_ip=ssh\_ip,  
 ssh\_port=ssh\_port,  
 note=note  
 )  
 server.server\_group.add(server\_group)  
 result = {'code': 200, 'msg': '导入成功'}  
 except Exception as e:  
 result = {'code': 500, 'msg': '导入异常！%s' %e}  
  
 return Response(result)

浏览器测试下载，apipost测试上传并带值。



### 6、云主机导入功能

云主机采集很方便，无需agent脚本，直接通过云平台API获取即可。



地区、可用区和服务器之间关系图

#### 阿里云

在线API调试平台：<https://api.aliyun.com/>

获取AceessKey文档：

<https://help.aliyun.com/document_detail/175967.html>

获取AceessKey地址：<https://ram.console.aliyun.com/manage/ak>

pip install aliyun-python-sdk-ecs -i <https://mirrors.aliyun.com/pypi/simple>

devops\_api/libs/aliyun\_cloud.py

from aliyunsdkcore.client import AcsClient  
from aliyunsdkecs.request.v20140526 import DescribeRegionsRequest, DescribeInstancesRequest, DescribeZonesRequest, DescribeDisksRequest  
import json  
  
class AliCloud():  
 def \_\_init\_\_(self, secret\_id, secret\_key):  
 self.secret\_id = secret\_id  
 self.secret\_key = secret\_key  
  
 def region\_list(self):  
 client = AcsClient(self.secret\_id, self.secret\_key)  
 req = DescribeRegionsRequest.DescribeRegionsRequest() # 获取地区  
 try:  
 resp = client.do\_action\_with\_exception(req)  
 resp = json.loads(resp.decode())  
 resp = {'code': 200, 'data': resp}  
 return resp  
 except Exception as e:  
 return {'code': '500', 'msg': e.get\_error\_msg()}  
  
 def zone\_list(self, region\_id):  
 client = AcsClient(self.secret\_id, self.secret\_key)  
 req = DescribeZonesRequest.DescribeZonesRequest()  
 req.add\_query\_param('RegionId', region\_id)  
 try:  
 resp = client.do\_action\_with\_exception(req)  
 resp = json.loads(resp.decode())  
 resp = {'code': 200, 'data': resp}  
 return resp  
 except Exception as e:  
 return {'code': '500', 'msg': e.get\_error\_msg()}  
  
 def instance\_list(self, region\_id):  
 client = AcsClient(self.secret\_id, self.secret\_key)  
 req = DescribeInstancesRequest.DescribeInstancesRequest()  
 req.add\_query\_param('RegionId', region\_id)  
 try:  
 resp = client.do\_action\_with\_exception(req)  
 resp = json.loads(resp.decode())  
 resp = {'code': 200, 'data': resp}  
 return resp  
 except Exception as e:  
 return {'code': '500', 'msg': e.get\_error\_msg()}  
  
 def instance\_disk(self, instance\_id):  
 client = AcsClient(self.secret\_id, self.secret\_key)  
 req = DescribeDisksRequest.DescribeDisksRequest()  
 req.add\_query\_param('InstanceId', instance\_id)  
 try:  
 resp = client.do\_action\_with\_exception(req)  
 resp = json.loads(resp.decode())  
 resp = {'code': 200, 'data': resp}  
 return resp  
 except Exception as e:  
 return {'code': '500', 'msg': e.get\_error\_msg()}  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 cloud = AliCloud("LTAI5t9TByf4T3KQwn3Ct5rZ", "bm7E7p0uKJjtsqvclPysHj2jJfOwWs")  
 # result = cloud.region\_list()  
 # result = cloud.zone\_list('cn-hangzhou')  
 # result = cloud.instance\_list("cn-hangzhou")  
 # result = cloud.instance\_disk("i-bp1g28isv8irjtrwdxf4")  
 # print(result)

测试凭据：

LTAI5t9TByf4T3KQwn3Ct5rZ

bm7E7p0uKJjtsqvclPysHj2jJfOwWs

class AliyunCloudView(APIView):  
 """  
 阿里云获取云主机信息  
 """  
 def get(self, request):  
 """  
 返回所有区域（region）  
 """  
 secret\_id = request.query\_params.get('secret\_id')  
 secret\_key = request.query\_params.get('secret\_key')  
  
 cloud = AliyunCloud("LTAI5t9TByf4T3KQwn3Ct5rZ", "bm7E7p0uKJjtsqvclPysHj2jJfOwWs")  
 # cloud = AliyunCloud(secret\_id, secret\_key)  
 region\_result = cloud.region\_list()  
  
 code = region\_result['code']  
 if code == 200:  
 # 二次处理，固定字段名  
 region = []  
 for r in region\_result['data']['Regions']['Region']:  
 region.append({"region\_id": r['RegionId'], 'region\_name': r['LocalName']})  
 res = {'code': code, 'msg': '获取区域列表成功！', 'data': region}  
 else:  
 res = {'code': code, 'msg': region\_result['msg']}  
 return Response(res)  
  
 def post(self, request):  
 """  
 根据区域名称创建机房，再导入云主机（绑定机房）到数据库  
 """  
 # 凭据、IDC机房、主机分组、SSH连接地址（IP、端口）  
 secret\_id = request.data.get('secret\_id')  
 secret\_key = request.data.get('secret\_key')  
 server\_group\_id = int(request.data.get('server\_group'))  
 region\_id = request.data.get('region') # 区域用于机房里的城市  
 ssh\_ip = request.data.get('ssh\_ip') # 用户选择使用内网（private）还是公网（public），下面判断对应录入  
 ssh\_port = int(request.data.get('ssh\_port'))  
  
 cloud = AliyunCloud(secret\_id, secret\_key)  
 instance\_result = cloud.instance\_list(region\_id)  
  
 instance\_list = []  
 if instance\_result['code'] == 200:  
 instance\_list = instance\_result['data']['Instances']['Instance']  
 if len(instance\_list) == 0:  
 res = {'code': 500, 'msg': '该区域未发现云主机，请重新选择！'}  
 return Response(res)  
 elif instance\_result['code'] == 500:  
 res = {'code': 500, 'msg': '%s' %instance\_result['msg']}  
 return Response(res)  
  
 # InstanceSet中可用区字段值是英文，例如 ap-beijing-1  
 # 先获取可用区英文与中文对应，下面遍历主机再获取中文名  
 zone\_result = cloud.zone\_list(region\_id)  
 zone\_dict = {}  
 for z in zone\_result['data']['Zones']['Zone']:  
 zone\_dict[z['ZoneId']] = z['LocalName']  
  
 # 获取主机所在可用区  
 # 可用区用于机房里的机房名称  
 zone\_set = set()  
 for host in instance\_list:  
 zone = host['ZoneId'] # 可用区，例如 ap-beijing-1  
 zone\_set.add(zone\_dict[zone]) # 获取中文名  
  
 # 根据可用区创建机房  
 for zone in zone\_set:  
 # 如果存在不创建  
 idc = Idc.objects.filter(name=zone)  
 if not idc:  
 city = ""  
 region\_list = cloud.region\_list()['data']['Regions']['Region']  
 for r in region\_list: # 获取区域对应中文名  
 if r['RegionId'] == region\_id:  
 city = r['LocalName']  
 Idc.objects.create(  
 name=zone,  
 city=city,  
 provider="阿里云"  
 )  
  
 # 导入云主机信息到数据库  
 for host in instance\_list:  
 zone = host['ZoneId']  
 instance\_id = host['InstanceId'] # 实例ID  
 # hostname = host['HostName']  
 instance\_name = host['InstanceName'] # 机器名称  
 os\_version = host['OSName']  
  
 private\_ip\_list = host['NetworkInterfaces']['NetworkInterface'][0]['PrivateIpSets']['PrivateIpSet']  
 private\_ip = []  
 for ip in private\_ip\_list:  
 private\_ip.append(ip['PrivateIpAddress'])  
  
 public\_ip = host['PublicIpAddress']['IpAddress']  
 cpu = "%s核" %host['Cpu']  
 memory = "%sG" %(int(host['Memory']) / 1024)  
  
 # 硬盘信息需要单独获取  
 disk = []  
 disk\_list = cloud.instance\_disk(instance\_id)['data']['Disks']['Disk']  
 for d in disk\_list:  
 disk.append({'device': d['Device'], 'size': '%sG' %d['Size'], 'type': None})  
  
 create\_date = time.strftime("%Y-%m-%d",time.strptime(host['CreationTime'] , "%Y-%m-%dT%H:%MZ"))  
 # 2022-01-30T04:51Z 需要转换才能存储  
 expired\_time = time.strftime("%Y-%m-%d %H:%M:%S",time.strptime(host['ExpiredTime'], "%Y-%m-%dT%H:%MZ"))  
  
 # 创建服务器  
 idc\_name = zone\_dict[zone]  
 idc = Idc.objects.get(name=idc\_name) # 一对多  
  
 if ssh\_ip == "public":  
 ssh\_ip = public\_ip[0]  
 elif ssh\_ip == "private":  
 ssh\_ip = private\_ip[0]  
  
 data = {'idc': idc,  
 'name': instance\_name,  
 'hostname': instance\_id,  
 'ssh\_ip': ssh\_ip,  
 'ssh\_port': ssh\_port,  
 'machine\_type': 'cloud\_vm',  
 'os\_version': os\_version,  
 'public\_ip': public\_ip,  
 'private\_ip': private\_ip,  
 'cpu\_num': cpu,  
 'memory': memory,  
 'disk': disk,  
 'put\_shelves\_date': create\_date,  
 'expire\_datetime': expired\_time,  
 'is\_verified': 'verified'}  
 # 如果instance\_id不存在才创建  
 server = Server.objects.filter(hostname=instance\_id)  
 if not server:  
 server = Server.objects.create(\*\*data)  
 # 分组多对多  
 group = ServerGroup.objects.get(id=server\_group\_id) # 根据id查询分组  
 server.server\_group.add(group) # 将服务器添加到分组  
 else:  
 server.update(\*\*data)  
  
 res = {'code': 200, 'msg': '导入云主机成功'}  
 return Response(res)

#### 腾讯云

在线API调试平台：<https://console.cloud.tencent.com/api/explorer>

API文档：<https://cloud.tencent.com/document/product/296/19850>

Python SDK：<https://github.com/TencentCloud/tencentcloud-sdk-python>

获取AceessKey地址：<https://console.cloud.tencent.com/cam/capi>

pip install tencentcloud-sdk-python -i https://mirrors.tencent.com/pypi/simple/

devops\_api/libs/tencent\_cloud.py

from tencentcloud.common import credential  
from tencentcloud.common.exception.tencent\_cloud\_sdk\_exception import TencentCloudSDKException  
from tencentcloud.cvm.v20170312 import cvm\_client, models  
  
class TCloud():  
 def \_\_init\_\_(self, secret\_id, secret\_key):  
 self.secret\_id = secret\_id  
 self.secret\_key = secret\_key  
 self.cred = credential.Credential(self.secret\_id, self.secret\_key)  
  
 def region\_list(self):  
 client = cvm\_client.CvmClient(self.cred, None)  
 req = models.DescribeRegionsRequest() # 获取地区  
 try:  
 resp = client.DescribeRegions(req) # resp=[{"Region": "ap-guangzhou", "RegionName": "华南地区(广州)", "RegionState": "AVAILABLE"}, ]  
 resp.code = 200  
 return resp  
 except TencentCloudSDKException as e:  
 return {'code': '500', 'msg': e.message}  
  
 def zone\_list(self, region\_id):  
 client = cvm\_client.CvmClient(self.cred, region\_id)  
 req = models.DescribeZonesRequest()  
 try:  
 resp = client.DescribeZones(req)  
 resp.code = 200  
 return resp  
 except TencentCloudSDKException as e:  
 return {'code': '500', 'msg': e.message}  
  
 def instance\_list(self, region\_id):  
 client = cvm\_client.CvmClient(self.cred, region\_id) # 获取北京区域  
 req = models.DescribeInstancesRequest()  
 # req.InstanceIds = "ins-1511w4tn" #根据实例id获取  
 try:  
 resp = client.DescribeInstances(req)  
 resp.code = 200  
 return resp  
 except TencentCloudSDKException as e:  
 return {'code': '500', 'msg': e.message}  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 cloud = TCloud("AKIDJaox2BMemv5A92hKm5EE8tUuHTfmnU6a", " XzJVZVIzYzHVboxyukV9f2IVbZROLBsT")  
 # result = cloud.region\_list()  
 # result = cloud.zone\_list()  
 # result = cloud.instance\_list("ap-beijing")  
 # print(result)

测试凭据：

AKIDJaox2BMemv5A92hKm5EE8tUuHTfmnU6a

XzJVZVIzYzHVboxyukV9f2IVbZROLBsT

class TencentCloudView(APIView):  
 """  
 腾讯云获取云主机信息  
 """  
 def get(self, request):  
 """  
 返回所有区域  
 """  
 secret\_id = request.query\_params.get('secret\_id')  
 secret\_key = request.query\_params.get('secret\_key')  
  
 # cloud = TencentCloud("AKIDKUP3QWNxGI4ZnkQpwnMgiTxA7mAH8i02", "nJYWDtJSKwGJ5aokPownMegRu61f27wU")  
 cloud = TencentCloud(secret\_id, secret\_key)  
 region\_result = cloud.region\_list()  
  
 code = region\_result.code  
 if code == 200:  
 # 二次处理，固定字段名，与阿里云的一致  
 region = []  
 for r in region\_result.RegionSet:  
 region.append({"region\_id": r.Region, 'region\_name': r.RegionName})  
 res = {'code': code, 'msg': '获取区域列表成功！', 'data': region}  
 else:  
 res = {'code': code, 'msg': region\_result.msg}  
 return Response(res)  
  
 def post(self, request):  
 """  
 根据区域名称创建机房，再导入云主机（关联机房）到数据库  
 """  
 secret\_id = request.data.get('secret\_id')  
 secret\_key = request.data.get('secret\_key')  
 server\_group\_id = int(request.data.get('server\_group'))  
 region\_id = request.data.get('region')  
 ssh\_ip = request.data.get('ssh\_ip')  
 ssh\_port = int(request.data.get('ssh\_port'))  
  
 cloud = TencentCloud(secret\_id, secret\_key)  
 instance\_result = cloud.instance\_list(region\_id)  
  
 instance\_list = []  
 if instance\_result.code == 200:  
 instance\_list = instance\_result.InstanceSet  
 if instance\_result.TotalCount == 0:  
 res = {'code': 500, 'msg': '该区域未发现云主机，请重新选择！'}  
 return Response(res)  
 elif instance\_result.code == 500:  
 res = {'code': 500, 'msg': '%s' %instance\_result['msg']}  
 return Response(res)  
  
 # InstanceSet中可用区字段值是英文  
 # 先获取可用区英文与中文对应，下面遍历主机再获取中文名  
 zone\_result = cloud.zone\_list(region\_id)  
 zone\_dict = {}  
 for z in zone\_result.ZoneSet:  
 zone\_dict[z.Zone] = z.ZoneName  
  
 # 获取主机所在可用区  
 # 可用区，机房表：机房名称  
 host\_zone\_set = set()  
 for host in instance\_list:  
 zone = host.Placement.Zone # 可用区，例如 ap-beijing-1  
 host\_zone\_set.add(zone\_dict[zone]) # 获取中文名  
  
 # 根据可用区创建机房  
 for zone in host\_zone\_set:  
 # 如果存在不创建  
 idc = Idc.objects.filter(name=zone)  
 if not idc:  
 city = ""  
 region\_result = cloud.region\_list()  
 for r in region\_result.RegionSet: # 获取区域对应中文名  
 if r.Region == region\_id:  
 city = r.RegionName  
  
 Idc.objects.create(  
 name=zone,  
 city=city,  
 provider="腾讯云"  
 )  
  
 # 导入云主机信息到数据库  
 for host in instance\_list:  
 zone = host.Placement.Zone  
 instance\_id = host.InstanceId # 实例ID  
 instance\_name = host.InstanceName # 机器名称  
  
 os\_version = host.OsName  
 private\_ip = host.PrivateIpAddresses  
 public\_ip = host.PublicIpAddresses  
 cpu = "%s核" %host.CPU  
 memory = "%sG" %host.Memory  
  
 disk = [{'device': 'None', 'size': host.SystemDisk.DiskSize, 'type': 'None'}] # 默认保存是系统盘  
 data\_list = host.DataDisks  
 if data\_list:  
 for d in data\_list:  
 disk.append({'device': 'None', 'size': d.DiskSize, 'type': 'None'})  
  
 create\_date = time.strftime("%Y-%m-%d",time.strptime(host.CreatedTime, "%Y-%m-%dT%H:%M:%SZ"))  
 expired\_time = time.strftime("%Y-%m-%d %H:%M:%S",time.strptime(host.ExpiredTime, "%Y-%m-%dT%H:%M:%SZ"))  
 # state = host.InstanceState  
  
 # 创建服务器  
 idc\_name = zone\_dict[zone]  
 idc = Idc.objects.get(name=idc\_name)  
  
 if ssh\_ip == "public":  
 ssh\_ip = public\_ip[0] # 使用第一个IP连接  
 elif ssh\_ip == "private":  
 ssh\_ip = private\_ip[0]  
  
 # 如果instance\_id不存在才创建  
 data = {'idc': idc,  
 'name': instance\_name,  
 'hostname': instance\_id,  
 'ssh\_ip': ssh\_ip,  
 'ssh\_port': ssh\_port,  
 'machine\_type': 'cloud\_vm',  
 'os\_version': os\_version,  
 'public\_ip': public\_ip,  
 'private\_ip': private\_ip,  
 'cpu\_num': cpu,  
 'memory': memory,  
 'disk': disk,  
 'put\_shelves\_date': create\_date,  
 'expire\_datetime': expired\_time,  
 'is\_verified': 'verified'}  
 server = Server.objects.filter(hostname=instance\_id)  
 if not server:  
 server = Server.objects.create(\*\*data)  
 # 分组多对多  
 group = ServerGroup.objects.get(id=server\_group\_id) # 根据id查询分组  
 server.server\_group.add(group) # 将服务器添加到分组  
 else:  
 server.update(\*\*data)  
  
 res = {'code': 200, 'msg': '导入云主机成功'}  
 return Response(res)

### 7、同步按钮功能

from cmdb.views import ExcelCreateHostView, CreateHostView, HostCollectView  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 re\_path('^api/login/$', token\_auth.CustomAuthToken.as\_view()),  
 re\_path('^api/docs/$', schema\_view),  
 re\_path('^api/cmdb/create\_host/$', CreateHostView.as\_view()),  
 re\_path('^api/cmdb/excel\_create\_host/$', ExcelCreateHostView.as\_view()),  
 re\_path('^api/cmdb/host\_collect/$', HostCollectView.as\_view()),  
]

class HostCollectView(APIView):

def get(self, request):

hostname = request.query\_params.get('hostname')

server = Server.objects.get(hostname=hostname)

ssh\_ip = server.ssh\_ip

ssh\_port = server.ssh\_port

# 未绑定凭据并且没有选择凭据

credential\_id = request.query\_params.get('credential\_id')

if not server.credential and not credential\_id:

result = {'code': 500, 'msg': '未发现凭据，请选择！'}

return Response(result)

elif server.credential:

credential\_id = int(server.credential.id)

elif credential\_id:

credential\_id = int(request.query\_params.get('credential\_id'))

credential = Credential.objects.get(id=credential\_id)

username = credential.username

if credential.auth\_mode == 1:

password = credential.password

ssh = SSH(ssh\_ip, ssh\_port, username, password=password)

else:

private\_key = credential.private\_key

ssh = SSH(ssh\_ip, ssh\_port, username, key=private\_key)

# 先SSH基本测试

test = ssh.test()

if test['code'] == 200:

client\_agent\_name = "host\_collect.py"

local\_file = os.path.join(settings.BASE\_DIR, 'cmdb', 'files', client\_agent\_name)

remote\_file = os.path.join(settings.CLIENT\_COLLECT\_DIR, client\_agent\_name) # 这个工作路径在setting里配置

ssh.scp(local\_file, remote\_file=remote\_file)

ssh.command('chmod +x %s' % remote\_file)

result = ssh.command('python %s' % remote\_file)

if result['code'] == 200: # 采集脚本执行成功

# 再进一步判断客户端采集脚本提交结果

data = json.loads(result['data'])

Server.objects.filter(hostname=hostname).update(\*\*data)

# 更新凭据ID

server.credential = credential

server.is\_verified = 'verified'

server.save()

result = {'code': 200, 'msg': '主机配置同步成功'}

else:

result = {'code': 500, 'msg': '主机配置同步失败！错误：%s' %result['msg']}

else:

result = {'code': 500, 'msg': 'SSH连接异常！错误：%s' %test['msg']}

return Response(result)

## 3.4 接口文档

### 1、启用接口文档

pip install django-rest-swagger

INSTALLED\_APPS = [  
 'rest\_framework\_swagger',  
]

from rest\_framework\_swagger.views import get\_swagger\_view  
schema\_view = get\_swagger\_view(title='接口文档')  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 re\_path('^api/login/$', token\_auth.CustomAuthToken.as\_view()),  
 re\_path('^api/docs/$', schema\_view),  
]

REST\_FRAMEWORK = {  
 # API接口文档  
 'DEFAULT\_SCHEMA\_CLASS': 'rest\_framework.schemas.coreapi.AutoSchema',

}

### 2、文档示例

请求路径：/api/cmdb/idc/

请求方法：

* 新增（post）：/api/cmdb/idc/
* 查询所有（get）：/api/cmdb/idc/
* 查询单个（get）：/api/cmdb/idc/<id>
* 更新单个（put）：/api/cmdb/idc/<id>
* 删除单个（delete）：/api/cmdb/idc/<id>

请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| 键 | 值 | 说明 | 是否必填 |
| ordering | =name 正排序  =-name 倒排序 | 排序 | 否 |
| search | 字符串 | 搜索 | 否 |
| name | 字符串 | 过滤 | 否 |
| page\_size | 数字 | 每页显示条数 | 否 |
| page\_num | 数字 | 当前页码 | 否 |

响应参数

|  |  |
| --- | --- |
| 参数 | 说明 |
| code | 状态码，200返回成功，500返回失败 |
| msg | 消息 |
| count | 总条数 |
| data | 数据 |

响应数据

{

"code": 200,

"msg": "成功",

"count": 2,

"data": [

{

"id": 1,

"name": "酒仙桥兆维",

"city": "北京",

"provider": "某公司",

"note": null,

"create\_time": "2022-06-06T16:35:45.389410"

},

{

"id": 2,

"name": "浦东电信",

"city": "上海",

"provider": "某公司",

"note": null,

"create\_time": "2022-06-06T16:20:06.958453"

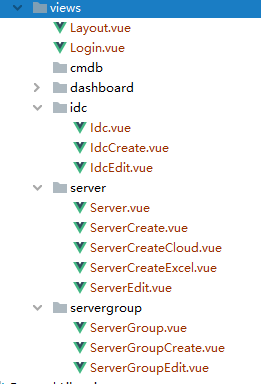
},

]

}

# 五、前端开发

## 5.1 登录页面



### 1、布局



<template>  
 <div class="main">  
 <div class="login\_box">  
 <div class="title">  
 DevOps运维平台  
 </div>  
 <div class="login\_form">  
 <el-form label-width="30px">  
 <el-form-item>  
 <el-input  
 :prefix-icon="User"  
 placeholder="用户名"  
 ></el-input>  
 </el-form-item>  
 <el-form-item>  
 <el-input  
 :prefix-icon="Lock"  
 placeholder="密码"  
 type="password"  
 show-password  
 ></el-input>  
 </el-form-item>

<el-form-item class="btn">  
 <el-button type="primary" @click="onSubmit" >登录</el-button>  
 </el-form-item>  
 </el-form>  
  
 </div>  
 </div>  
 </div>  
</template>  
  
<script>  
 // input里加图标必须单独按需导入  
 import { User, Lock } from "@element-plus/icons-vue";  
 export default {  
 // 需要用setup导入  
 setup() {  
 return {  
 User,Lock  
 }  
 }  
 }  
</script>  
  
<style scoped>  
 .main {  
 background-image: url("../assets/img/login\_background.png");  
 background-size: 100% 100%;  
 height: 100%;  
 }  
 .login\_box {  
 width: 400px;  
 height: 300px;  
 background-color: #FFFFFF;  
 box-shadow: 0 5px 20px 0 #e8e8e8;  
 border-radius: 20px;  
 position: absolute;  
 top: 0;  
 bottom: 0;  
 right: 0;  
 left: 0;  
 margin: auto;  
 }  
 .title {  
 font-size: 20px;  
 font-weight: bold;  
 color: #409eff;  
 text-align: center;  
 margin-top: 30px;  
 }  
 .login\_form {  
 margin-top: 40px;  
 margin-right: 30px;  
 }  
 .btn {  
 margin-left: 36%;  
 }  
 .btn .el-button {  
 width: 140px;  
 }  
</style>

* label-width：表单左侧距离
* show-password：可切换显示隐藏的密码框

### 2、数据绑定

<el-form :model="form" :rules="rules" label-width="30px">  
 <el-form-item prop="username">  
 <el-input  
 :prefix-icon="User"  
 placeholder="用户名"  
 v-model="form.username"  
 ></el-input>  
 </el-form-item>  
 <el-form-item prop="password">  
 <el-input  
 :prefix-icon="Lock"  
 placeholder="密码"  
 type="password"  
 show-password  
 v-model="form.password"  
 ></el-input>  
 </el-form-item>

<el-form-item class="btn">  
 <el-button type="primary" @click="onSubmit" >登录</el-button>  
 </el-form-item>  
</el-form>

export default {  
 data() {  
 return {  
 form: {  
 username: '',  
 password: ''  
 },  
 rules: {  
 username: [  
 {required: true, message: '请输入用户名', trigger: 'blur'},  
 {min: 3, message: '用户名长度应不小于3个字符', trigger: 'blur'}  
 ],  
 password: [  
 {required: true, message: '请输入密码', trigger: 'blur'},  
 {min: 6, message: '用户名长度应不小于6个字符', trigger: 'blur'}  
 ]  
 }  
 }  
 },

methods: {  
 onSubmit() {  
 // 提交前预验证  
 this.$refs.form.validate((valid) => { //回调函数中valid布尔值  
 if (valid) {  
 // 消息提示  
 this.$message.success('登录成功')  
 } else {  
 this.$message.warning('用户名或密码格式错误！')  
 }  
 })  
 }  
},

### 3、提交后端

思路：

1. 将输入的用户名和密码POST提交到API
2. 如果API返回状态码为200说明认证成功
3. 将token保存在sessionStorage中
4. 认证成功跳转到首页，否则提示用户名或密码不正确。

methods: {  
 onSubmit() {  
 // 提交前预验证  
 this.$refs.ruleFormRef.validate((valid) => { // 回调函数中valid布尔值  
 if (valid) {  
 this.$axios.post('http://192.168.31.108:8001/api/login/', this.form)  
 .then(res => {  
 if (res.data.code == 200) {  
 this.$message.success('登录成功');  
 // 保存token到会话存储  
 window.sessionStorage.setItem("token", res.data.token);  
 this.$router.push("/home") //导航跳转到首页  
 // window.location.href = "#/dashboard"  
 } else {  
 this.$message.warning('用户名或密码不正确！')  
 }  
 })

.catch((error) => {  
 this.$message.error('服务端接口请求错误！'+ error)  
 })

} else {  
 this.$message.warning('用户名或密码格式错误！')  
 }  
 })  
 }  
},

提交会出现错误，F12调试查看：

Access to XMLHttpRequest at 'http://192.168.31.108:8001/api/login/' from origin 'http://192.168.31.108:8080' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.

提示涉及跨域访问，配置django放行跨域请求：

pip install django-cors-headers

settings.py



### 4、axios二次封装

现在已经在实际场景中用到了axios请求API，你会发现，例如每次请求写完整的API地址，但前部分都是固定的（例如域名发生变化，那需要一个个修改，会非常麻烦），后续登录后所有请求API都要header携带token，还有catch错误处理。这些都是固定，如果每次都要写显得臃肿很多。所以我们需要axios二次封装，简化使用。

src/api/http.js

import axios from "axios";  
import { ElMessage } from 'element-plus';  
  
const instance = axios.create({  
 baseURL: 'http://192.168.31.108:8001/api',  
 timeout: 3000,  
 // headers: {'X-Custom-Header': 'foobar'}  
});  
  
// 拦截器：请求拦截  
instance.interceptors.request.use(config => {  
 // 在请求被发送之前做些什么  
 const token = window.sessionStorage.getItem('token');  
 if (token) {  
 config.headers = {  
 'Authorization': 'token ' + token  
 };  
 }  
 return config;  
}, error => {  
 // 处理请求错误  
 return Promise.reject(error)  
});  
  
// 拦截器：响应拦截  
instance.interceptors.response.use(response => {  
 // console.log('响应拦截处理');  
 if(response.data.code != 200) {  
 ElMessage.warning(response.data.msg) // 这里应根据后端返回消息显示  
 }  
 return response  
}, error => {  
 // 处理响应错误（catch）  
 ElMessage.error('请求服务端接口错误：' + error.message);  
 return Promise.reject(error)  
});  
  
// 导出实例  
export default instance

请求拦截：每一次请求去判断是否有token，如果token存在则在请求头加上这个token。

响应拦截：所有响应不等于200的状态码都提示消息

main.js

import axios from './api/http'

app.config.globalProperties.$http = axios; //在app=createApp(App)下面

登录：

methods: {  
 onSubmit() {  
 // 提交前预验证  
 this.$refs.form.validate((valid) => { // 回调函数中valid布尔值  
 if (valid) {  
 this.$http.post('/login/', this.form)  
 .then(res => {  
 if (res.data.code == 200) {  
 this.$message.success('登录成功');  
 // 保存token到会话存储，使用用户名和token作为key确保唯一  
 window.sessionStorage.setItem('token', res.data.token);  
 window.sessionStorage.setItem('username', res.data.username);  
 this.$router.push("/dashboard") //导航跳转到首页  
 }  
 })  
 } else {  
 this.$message.error('用户名或密码格式错误！')  
 }  
 })  
 }  
},

### 5、路由导航守卫控制访问权限

目前虽然已经实现登录功能，即使没有登录情况下直接访问任何页面都还可以访问的，我们希望如果用户没有登录情况下，访问任何页面都重新导航到登录页面。

// 添加导航守卫  
router.beforeEach((to, from, next) => {  
 // 如果用户访问登录页，直接放行  
 if(to.path == '/login') {  
 return next()  
 }  
 // 从sessionStorage获取token值  
 const token = window.sessionStorage.getItem('token');  
 // 如果没有获取到token值，跳转到登录页  
 if (!token) {  
 return next('/login')  
 }  
 // 正常跳转  
 next()  
});  
  
export default router;

* to：即将要进入的目标，是一个路由对象
* from：当前导航正要离开的路由，也是一个路由对象
* next：可选，是一个方法

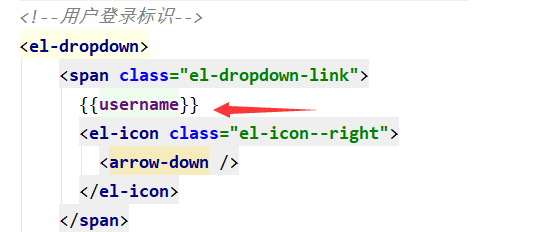
## 5.2 后台基本布局

### 1、显示用户名

Login.vue



Layout.vue





### 2、退出

基于token的方法实现退出比较简单，只需要删除本地的token即可，这样，后续的请求就获取不了token携带发送。

Home.vue

<template #dropdown>  
 <el-dropdown-menu>  
 <el-dropdown-item>个人中心</el-dropdown-item>  
 <el-dropdown-item @click="logout">退出登录</el-dropdown-item>  
 </el-dropdown-menu>  
</template>

logout() {  
 window.sessionStorage.clear()  
 this.$router.push('/login')  
}

### 3、修改密码

#### 对话框与表单



<el-dropdown-item @click="UserPasswordDialog = true">修改密码</el-dropdown-item>

<!--修改密码对话框-->  
 <el-dialog  
 v-model="UserPasswordDialog"  
 width="30%"  
 title="修改密码"  
 :before-close="handleClose"  
 >  
 <el-form :model="UserPasswordForm" label-position=“right” label-width="100px" :rules="rules" ref="UserPasswordForm">  
 <el-form-item label="原密码：" prop="old\_password">  
 <el-input  
 v-model="UserPasswordForm.old\_password"  
 type="password"  
 show-password  
 ></el-input>  
 </el-form-item>  
 <el-form-item label="新密码：" prop="new\_password">  
 <el-input  
 v-model="UserPasswordForm.new\_password"  
 type="password"  
 show-password  
 ></el-input>  
 </el-form-item>  
 <el-form-item label="再次确认：" prop="confirm\_password">  
 <el-input  
 v-model="UserPasswordForm.confirm\_password"  
 type="password"  
 show-password  
 ></el-input>  
 </el-form-item>  
 </el-form>  
 <template #footer>  
 <span class="dialog-footer">  
 <el-button @click="UserPasswordDialog = false">取消</el-button>  
 <el-button type="primary" @click="changePasswordSubmit">确定</el-button>  
 </span>  
 </template>  
 </el-dialog>

return {  
 memus: [],  
 UserPasswordDialog: false,  
 UserPasswordForm: {  
 old\_password: '',  
 new\_password: '',  
 confirm\_password: ''  
 },  
 rules: {  
 old\_password: [  
 {required: true, message: '请输入原密码', trigger: 'blur'},  
 {min: 6, message: '用户名长度应不小于6个字符', trigger: 'blur'}  
 ],  
 new\_password: [  
 {required: true, message: '请输入新密码', trigger: 'blur'},  
 {min: 6, message: '用户名长度应不小于6个字符', trigger: 'blur'},  
 ],  
 confirm\_password: [  
 {required: true, message: '请确认新密码', trigger: 'blur'},  
 {min: 6, message: '用户名长度应不小于6个字符', trigger: 'blur'},  
 ]  
 }  
}

**methods**

changePasswordSubmit() {  
 // 提交前预验证  
 this.$refs.UserPasswordForm.validate((valid) => { // 回调函数中valid布尔值  
 console.log(this.UserPasswordForm)  
 if (valid) {  
 this.$axios.post('http://192.168.31.108:8001/api/change\_password/', this.UserPasswordForm)  
 .then(res => {  
 if (res.data.code == 200) {  
 this.$message.success('修改成功');  
 this.UserPasswordDialog = false;  
 } else {  
 this.$message.warning('修改密码失败！') // 这里应根据后端返回消息显示  
 }  
 })  
 .catch((error) => {  
 this.$message.error('服务端接口请求错误！'+ error)  
 })  
 } else {  
 this.$message.warning('密码格式错误！')  
 }  
 });  
}

#### 判断两次新密码一致（自定义效验规则）

在data.return上面增加效验方法：

data(){  
 const checkNewOldPassword = (rule, value, callback) => {  
 if (value == this.UserPasswordForm.old\_password) {  
 callback(new Error('新密码不能与旧密码一样！'))  
 } else {

return callback()

}  
 };  
 const checkNewPassword = (rule, value, callback) => {  
 if (value != this.UserPasswordForm.new\_password) {  
 callback(new Error('两次输入密码不一致！'))  
 } else {

return callback()

}  
 };

return {}

}

规则里增加：

new\_password: [  
 {required: true, message: '请输入新密码', trigger: 'blur'},  
 {min: 6, message: '用户名长度应不小于6个字符', trigger: 'blur'},  
 {validator: checkNewOldPassword, trigger: 'blur'}  
 ],  
 confirm\_password: [  
 {required: true, message: '请确认新密码', trigger: 'blur'},  
 {min: 6, message: '用户名长度应不小于6个字符', trigger: 'blur'},  
 {validator: checkNewPassword, trigger: 'blur'}  
 ]

## 5.4 主机管理

### 1、机房管理

#### 1、表格展示

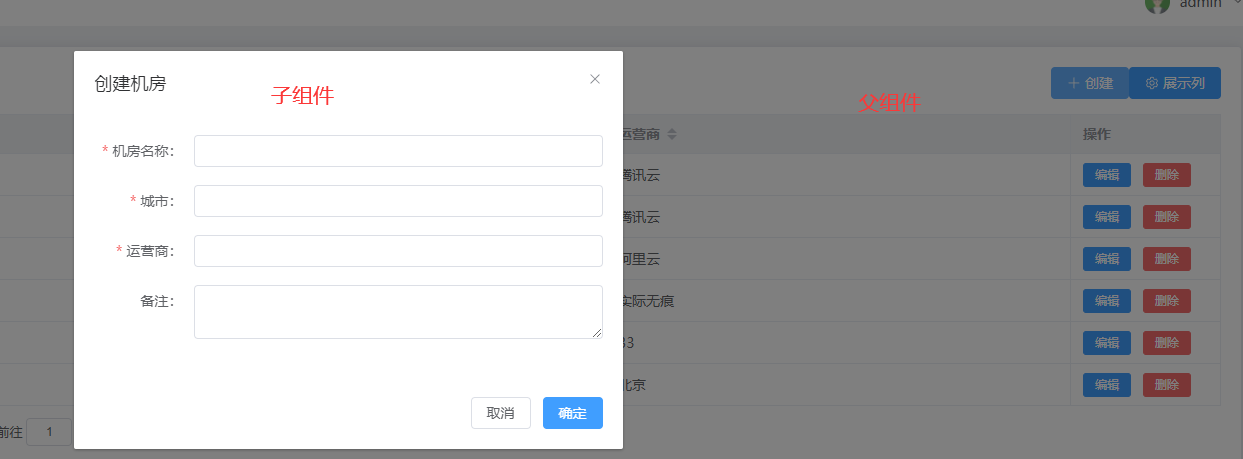
<template>  
 <el-card class="box-card">  
  
 <el-table  
 ref="Table"  
 :data="tableData"  
 @selection-change="handleSelectionChange"  
 border  
 prop  
 style="width: 100%"  
 :header-cell-style="{ background: '#F0F2F5' }"  
 >  
 <el-table-column type="selection" width="55" /> <!--启用多选-->  
 <el-table-column prop="name" label="机房名称" width="180" sortable/>  
 <el-table-column prop="city" label="城市" width="180" sortable/>  
 <el-table-column prop="provider" label="运营商" sortable/>  
 <el-table-column prop="note" label="备注" />  
 <el-table-column prop="create\_time" label="创建时间" sortable/>  
  
 <!--操作栏-->  
 <el-table-column label="操作" align="left">  
 <template #default="scope">  
 <el-button  
 size="small"  
 type="primary"  
 @click="handleEdit(scope.$index, scope.row)"  
 >编辑</el-button>  
 <el-button  
 size="small"  
 type="danger"  
 @click="handleDelete(scope.$index, scope.row)"  
 >删除</el-button  
 >  
 </template>  
 </el-table-column>  
  
 </el-table>  
  
 </el-card>  
</template>  
  
<script>  
export default {  
 data() {  
 return {  
 tableData: [],  
 }  
 },  
 mounted() {  
 this.getData()  
 },  
 methods: {  
 getData() {  
 this.$http.get('/cmdb/idc/')  
 .then(res => {  
 this.tableData = res.data.data  
 })  
 }  
 },  
};  
</script>

#### 2、分页

<template>  
 <el-card class="box-card">  
  
 <el-table  
 …  
 </el-table>  
  
 <!--分页-->  
 <div style="margin-top: 20px">  
 <el-pagination  
 v-model:currentPage="currentPage"  
 :page-sizes="[10, 15, 20, 25, 30]"  
 :page-size="pageSize"  
 layout="total, sizes, prev, pager, next, jumper"  
 :total="total"  
 background  
 @size-change="handleSizeChange"  
 @current-change="handleCurrentChange"  
 >  
 </el-pagination>  
 </div>  
  
 </el-card>  
</template>  
  
<script>  
export default {  
 data() {  
 return {  
 tableData: [],  
 currentPage: 1, // 默认开始页面  
 pageSize: 10, // 默认每页的数据条数  
 total: 0, // 总数据条数  
 urlParams: { // URL查询参数，传递服务端，放这里方便修改值  
 page\_num: 1,  
 page\_size: this.pageSize  
 }  
 }  
 },  
 created() {  
 // 加载页面时获取数据  
 this.getData()  
 },  
 methods: {  
 getData() {  
 this.$http.get('/cmdb/idc/',{params: this.urlParams})  
 .then(res => {  
 this.tableData = res.data.data;  
 this.total = res.data.count;  
 })  
 },  
 // 分页：监听【选择每页数量】的事件  
 handleSizeChange(pageSize) {  
 // console.log(pageSize)  
 this.pageSize = pageSize; // 重新设置分页显示  
 this.urlParams.page\_size = pageSize; // 将最新值设置到数据里，用于下面用新值重新获取数据  
 this.getData()  
 },  
 // 分页：监听【点击页码】的事件  
 handleCurrentChange(currentPage) {  
 // console.log(currentPage)  
 this.currentPage = currentPage; // 重新设置分页显示  
 this.urlParams.page\_num = currentPage;  
 this.getData()  
 }  
 },  
};  
</script>

#### 3、编辑

随着页面功能越来越多，代码量就会越来越大，因此可以将一部分对话框代码解耦，有利于维护。



创建子组件 IdcEdit.vue

<!--操作栏：编辑对话框-->  
<el-dialog  
 :model-value="visible"  
 width="30%"  
 title="修改机房信息"  
 @close="dialogClose"  
 >  
 <el-form :model="form" ref="formRef" :rules="formRules" label-position=“right” label-width="100px">  
 <el-form-item label="机房名称：" prop="name">  
 <el-input v-model="form.name"></el-input>  
 </el-form-item>  
 <el-form-item label="城市：" prop="city">  
 <el-input v-model="form.city"></el-input>  
 </el-form-item>  
 <el-form-item label="运营商：" prop="provider">  
 <el-input v-model="form.provider"></el-input>  
 </el-form-item>  
 <el-form-item label="备注：">  
 <el-input v-model="form.note" type="textarea"></el-input>  
 </el-form-item>  
 </el-form>  
  
 <template #footer>  
 <span class="dialog-footer">  
 <el-button @click="dialogClose">取消</el-button>  
 <el-button type="primary" @click="submit">确定</el-button>  
 </span>  
 </template>  
</el-dialog>

Idc导入子组件：

import IdcEdit from './IdcEdit';

components: {  
 IdcEdit,  
}

传递打开窗口和当前行：

<IdcEdit v-model:visible="editDialogVisible" :rowId="rowId"></IdcEdit>

editDialogVisible: false, // 编辑对话框显示与隐藏  
rowId: '', // 传递给子组件当前行ID

// 操作栏：编辑显示  
handleEdit(index, row) {  
 this.editDialogVisible = true;  
 this.rowId = row.id; // 传递给子组件当前行ID  
},

子组件

script>  
 export default {  
 name: "IdcEdit",  
 props: {  
 visible: Boolean,  
 rowId: Number, // 当前行ID  
 },  
 data() {  
 return {  
 form: '',  
 formRules: {  
 name: [  
 {required: true, message: '请输入机房名称', trigger: 'blur'},  
 {min: 2, message: '机房名称长度应不小于2个字符', trigger: 'blur'}  
 ],  
 city: [  
 {required: true, message: '请输入城市', trigger: 'blur'},  
 {min: 2, message: '城市长度应不小于2个字符', trigger: 'blur'}  
 ],  
 provider: [  
 {required: true, message: '请输入运营商', trigger: 'blur'},  
 {min: 2, message: '运营商长度应不小于2个字符', trigger: 'blur'}  
 ]  
 },  
 }  
 },  
 methods: {  
 submit() {  
 this.$refs.formRef.validate((valid) => {  
 if (valid) {  
 this.$http.put('/cmdb/idc/' + this.rowId + '/', this.form)  
 .then(res => {  
 if (res.data.code == 200){  
 this.$message.success('修改成功');  
 this.$parent.getData(); // 调用父组件方法，更新数据  
 this.dialogClose() // 关闭窗口  
 }  
 })  
 } else {  
 this.$message.error('格式错误！')  
 }  
 })  
 },  
 // 点击关闭，子组件通知父组件更新属性  
 dialogClose() {  
 this.$emit('update:visible', false) // 父组件必须使用v-model  
 }  
 },  
 // 监听窗口打开，  
 watch: {  
 visible() {  
 if(this.visible) { // 关闭窗口不请求  
 this.$http.get('/cmdb/idc/' + this.rowId + '/')  
 .then(res => {  
 if (res.data.code == 200) {  
 this.form = res.data.data;  
 }  
 });  
 }  
  
 }  
 }  
 }  
</script>

#### 4、删除

// 操作栏：删除  
handleDelete(index, row) {  
 console.log(index, row);  
 this.$confirm("你确定要删除选中的吗？", "提示", {  
 confirmButtonText: "确定",  
 cancelButtonText: "取消",  
 type: "warning"  
 })  
 .then(() => { // 点击确定  
 this.$http.delete('/cmdb/idc/'+ row.id + '/')  
 .then(res => {  
 if(res.data.code == 200) {  
 this.$message.success('删除成功');  
 this.tableData.splice(index, 1); // 根据表格索引临时删除数据  
 }  
 });  
 })  
},

#### 5、搜索



卡片里面最上面添加：

<div style="margin-bottom: 10px;display: flex;align-items: center;justify-content: space-between">  
 <div>  
 <el-row>  
 <el-col :span="12">  
 <el-input  
 v-model="urlParams.search"  
 placeholder="请输入关键字"  
 @keyup.enter="onSearch"  
 clearable  
 @clear="onSearch"  
 class="search"  
 />  
 </el-col>  
 <el-col :span="2">  
 <el-button type="primary" @click="onSearch"><el-icon><search /></el-icon>&nbsp;搜索</el-button>  
 </el-col>  
 </el-row>  
 </div>  
 <div>  
 预留创建按钮区域  
 </div>  
</div>

URL查询参数增加search字段：

urlParams: { // URL查询参数，传递服务端，放这里方便修改值  
 page\_num: 1,  
 page\_size: 10,  
 search: '', // 如果没有输入关键字，默认为空  
},

// 查询提交（携带search重新查询）  
onSearch() {  
 this.getData()  
}

#### 6、新建

IdcCreate.vue

<template>  
 <el-dialog  
 :model-value="visible"  
 width="30%"  
 title="创建机房"  
 @close="dialogClose"  
 >  
 <el-form :model="form" ref="formRef" :rules="formRules" label-position=“right” label-width="100px" >  
 <el-form-item label="机房名称：" prop="name">  
 <el-input v-model="form.name"></el-input>  
 </el-form-item>  
 <el-form-item label="城市：" prop="city">  
 <el-input v-model="form.city"></el-input>  
 </el-form-item>  
 <el-form-item label="运营商：" prop="provider">  
 <el-input v-model="form.provider"></el-input>  
 </el-form-item>  
 <el-form-item label="备注：">  
 <el-input v-model="form.note" type="textarea"></el-input>  
 </el-form-item>  
 </el-form>  
  
 <template #footer>  
 <span class="dialog-footer">  
 <el-button @click="dialogClose">取消</el-button>  
 <el-button type="primary" @click="submit">确定</el-button>  
 </span>  
 </template>  
 </el-dialog>  
  
</template>  
  
<script>  
 export default {  
 name: "IdcCreate",  
 props: {  
 visible: Boolean,  
 },  
 data() {  
 return {  
 form: {  
 'name': '',  
 'city': '',  
 'provider': '',  
 'note': ''  
 },  
 formRules: {  
 name: [  
 {required: true, message: '请输入机房名称', trigger: 'blur'},  
 {min: 2, message: '机房名称长度应不小于2个字符', trigger: 'blur'}  
 ],  
 city: [  
 {required: true, message: '请输入城市', trigger: 'blur'},  
 {min: 2, message: '城市长度应不小于2个字符', trigger: 'blur'}  
 ],  
 provider: [  
 {required: true, message: '请输入运营商', trigger: 'blur'},  
 {min: 2, message: '运营商长度应不小于2个字符', trigger: 'blur'}  
 ]  
 },  
 }  
 },  
 methods: {  
 submit() {  
 this.$refs.formRef.validate((valid) => {  
 if (valid) {  
 console.log(this.form)  
 this.$http.post('/cmdb/idc/', this.form)  
 .then(res => {  
 if (res.data.code == 200){  
 this.$message.success('创建成功');  
 this.$parent.getData(); // 调用父组件方法，更新数据  
 this.dialogClose() // 关闭窗口  
 }  
 })  
 } else {  
 this.$message.error('格式错误！')  
 }  
 })  
 },  
 // 点击关闭，子组件通知父组件更新属性  
 dialogClose() {  
 this.$emit('update:visible', false) // 父组件必须使用v-model  
 }  
 }  
 }  
</script>

Idc.vue

<div style="margin-bottom: 10px;display: flex;align-items: center;justify-content: space-between">  
 <div>  
 <el-row>  
 <el-col :span="12">  
 <el-input  
 v-model="urlParams.search"  
 placeholder="请输入关键字"  
 @keyup.enter="onSearch"  
 clearable  
 @clear="onSearch"  
 class="search"  
 />  
 </el-col>  
 <el-col :span="2">  
 <el-button type="primary" @click="onSearch"><el-icon><search /></el-icon>&nbsp;搜索</el-button>  
 </el-col>  
 </el-row>  
 </div>  
 <div>  
 <el-button type="primary" @click="createDialogVisible = true"><el-icon><plus /></el-icon>&nbsp;创建</el-button>  
 </div>  
</div>

import IdcCreate from "./IdcCreate";

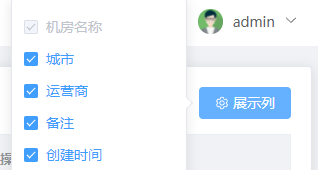
components: {  
 IdcEdit,  
 IdcCreate  
}

<IdcCreate v-model:visible="createDialogVisible"></IdcCreate>

由于api默认创建返回是创建的数据，前端无法作出判断,所以自定义响应数据。

class IdcViewSet(ModelViewSet):  
 queryset = Idc.objects.all()  
 serializer\_class = IdcSerializer  
 filter\_backends = [filters.SearchFilter, filters.OrderingFilter, DjangoFilterBackend] # 指定过滤器  
 search\_fields = ('name',) # 指定可搜索的字段  
 filter\_fields = ('city',)  
  
 def create(self, request, \*args, \*\*kwargs):  
 idc = Idc.objects.filter(name=request.data.get('name'))  
 if not idc:  
 serializer = self.get\_serializer(data=request.data)  
 serializer.is\_valid(raise\_exception=True)  
 self.perform\_create(serializer)  
 res = {'code': 200, 'msg': '创建成功'}  
 else:  
 res = {'code': 500, 'msg': '名称已存在！'}  
 return Response(res)

#### 7、展示列



1. **展示列弹出框**

<!--展示列弹出框-->  
<el-popover placement="left" :width="100" trigger="click">  
 <template #reference>  
 <el-button type="primary"><el-icon><setting /></el-icon>&nbsp;展示列</el-button>  
 </template>  
 <el-checkbox v-model="showColumn.name" disabled>机房名称</el-checkbox>  
 <el-checkbox v-model="showColumn.city">城市</el-checkbox>  
 <el-checkbox v-model="showColumn.provider">运营商</el-checkbox>  
 <el-checkbox v-model="showColumn.note">备注</el-checkbox>  
 <el-checkbox v-model="showColumn.create\_time">创建时间</el-checkbox>  
</el-popover>

1. **配置数据双向绑定**

showColumn: {  
 name: true,  
 city: true,  
 provider: true,  
 note: true,  
 create\_time: true  
},

表格字段根据布尔值动态显示

<el-table-column prop="name" label="机房名称" width="180" v-if="showColumn.name" sortable/>

1. **进一步完善**



完善：增加取消和确认按钮，保存到浏览器本地存储进行持久化

增加按钮和点击才会显示：

<div class="header-bar-right">  
 <el-button type="primary" @click="createDialogVisible = true"><el-icon><plus /></el-icon>&nbsp;创建</el-button>  
  
 <!--展示列弹出框-->  
 <el-popover placement="left" :width="100" v-model:visible='columnVisible'>  
 <template #reference>  
 <el-button type="primary" @click="columnVisible = true"><el-icon><setting /></el-icon>&nbsp;展示列</el-button>  
 </template>  
 <el-checkbox v-model="showColumn.name" disabled>机房名称</el-checkbox>  
 <el-checkbox v-model="showColumn.city">城市</el-checkbox>  
 <el-checkbox v-model="showColumn.provider">运营商</el-checkbox>  
 <el-checkbox v-model="showColumn.note">备注</el-checkbox>  
 <el-checkbox v-model="showColumn.create\_time">创建时间</el-checkbox>  
 <div style="text-align: right; margin: 0">  
 <el-button size="small" type="text" @click="columnVisible = false">取消</el-button>  
 <el-button size="small" type="primary" @click="saveColumn">确认</el-button>  
 </div>  
 </el-popover>  
</div>

在加载时判断本地是否有历史存储展示列字段：

columnVisible: false, // 可展示列显示与隐藏  
 showColumn: { // 字段默认是否展示  
 name: true,  
 city: true,  
 provider: true,  
 note: true,  
 create\_time: true  
 },  
 }  
},

mounted() {

// 加载页面时获取数据

this.getData();

// 从浏览器本地存储获取历史存储展示

const columnSet = localStorage.getItem(this.$route.path + '-columnSet');

if(columnSet) {

this.showColumn = JSON.parse(columnSet)

}  
},

保存到浏览器本地存储进行持久化：

saveColumn() {  
 // 将可显示的字段存储到浏览器本地存储  
 localStorage.setItem(this.$route.path + '-columnSet', JSON.stringify(this.showColumn));  
 this.columnVisible = false;  
}

### 2、主机分组

与机房管理一样，只是调用接口不同。

* 修改路由
* 修改接口地址
* 修改字段
* 修改展示列
* 修改对话框标题
* 修改子组件名称

### 3、主机管理

功能开发思路：

1. 基本布局与数据表格展示
2. 表格内容优化展示
3. 操作栏（编辑、删除、同步）
4. 查询
5. 新建（新建、excel、云主机）

#### 基本布局

##### 修改字段

<el-table-column type="selection" width="55"/> <!--启用多选-->  
<el-table-column prop="name" label="名称" width="130" fixed="left" sortable v-if="showColumn.name"/>  
<el-table-column prop="hostname" label="主机名" width="130" sortable v-if="showColumn.hostname"/>  
<el-table-column prop="idc" label="IDC机房" sortable v-if="showColumn.idc"/>  
<el-table-column prop="server\_group" label="主机分组" width="150" sortable v-if="showColumn.server\_group"/>  
<el-table-column prop="machine\_type" label="机器类型" width="110" sortable v-if="showColumn.machine\_type"/>  
<el-table-column prop="os\_version" label="系统版本" sortable :show-overflow-tooltip="true" v-if="showColumn.os\_version"/>  
<el-table-column prop="public\_ip" label="公网IP" width="180" sortable v-if="showColumn.public\_ip"/>  
<el-table-column prop="cpu\_num" label="CPU" width="80" sortable v-if="showColumn.cpu\_num"/>  
<el-table-column prop="cpu\_model" label="CPU型号" sortable v-if="showColumn.cpu\_model"/>  
<el-table-column prop="memory" label="内存" width="80" sortable v-if="showColumn.memory"/>  
<el-table-column prop="disk" label="硬盘" width="200" sortable v-if="showColumn.disk"/>  
<el-table-column prop="put\_shelves\_date" label="上架日期" sortable v-if="showColumn.put\_shelves\_date"/>  
<el-table-column prop="off\_shelves\_date" label="下架日期" sortable v-if="showColumn.off\_shelves\_date"/>  
<el-table-column prop="expire\_datetime" label="租约过期时间" sortable v-if="showColumn.expire\_datetime"/>  
<el-table-column prop="is\_verified" label="SSH状态" width="120" sortable v-if="showColumn.is\_verified"/>  
<el-table-column prop="note" label="备注" v-if="showColumn.note"/>  
<el-table-column prop="update\_time" label="更新时间" sortable v-if="showColumn.update\_time"/>  
<el-table-column prop="create\_time" label="创建时间" sortable v-if="showColumn.create\_time"/>

##### 修改展示列

<el-checkbox v-model="showColumn.name" disabled>名称</el-checkbox>  
<el-checkbox v-model="showColumn.hostname" disabled>主机名</el-checkbox>  
<el-checkbox v-model="showColumn.idc">IDC机房</el-checkbox>  
<el-checkbox v-model="showColumn.server\_group">主机分组</el-checkbox>  
<el-checkbox v-model="showColumn.machine\_type">机器类型</el-checkbox>  
<el-checkbox v-model="showColumn.os\_version">系统版本</el-checkbox>  
<el-checkbox v-model="showColumn.public\_ip">公网IP</el-checkbox>  
<el-checkbox v-model="showColumn.private\_ip">私有IP</el-checkbox>  
<el-checkbox v-model="showColumn.cpu\_num">CPU数量</el-checkbox>  
<el-checkbox v-model="showColumn.cpu\_model">CPU型号</el-checkbox>  
<el-checkbox v-model="showColumn.memory">内存</el-checkbox>  
<el-checkbox v-model="showColumn.disk">硬盘</el-checkbox>  
<el-checkbox v-model="showColumn.put\_shelves\_date">上架日期</el-checkbox>  
<el-checkbox v-model="showColumn.off\_shelves\_date">下架日期</el-checkbox>  
<el-checkbox v-model="showColumn.expire\_datetime">租约过期时间</el-checkbox>  
<el-checkbox v-model="showColumn.is\_verified">SSH验证状态</el-checkbox>  
<el-checkbox v-model="showColumn.note">备注</el-checkbox>  
<el-checkbox v-model="showColumn.update\_time">更新时间</el-checkbox>  
<el-checkbox v-model="showColumn.create\_time">创建时间</el-checkbox>

name: true,  
hostname: true,  
idc: true,  
server\_group: true,  
machine\_type: false,  
os\_version: true,  
public\_ip: true,  
private\_ip: true,  
cpu\_num: true,  
cpu\_model: false,  
memory: true,  
disk: true,  
put\_shelves\_date: false,  
off\_shelves\_date: false,  
expire\_datetime: false,  
is\_verified: false,  
update\_time: false,  
create\_time: false,  
note: false

##### 修改编辑对话框

<el-dialog  
 :model-value="visible"  
 width="40%"  
 title="修改主机"  
 @close="dialogClose"  
 >  
 <el-form :model="form" ref="formRef" :rules="formRules" label-position=“right” label-width="100px" >  
 <el-form-item label="机器名称：" prop="name">  
 <el-input v-model="form.name" placeholder="例如：测试机"></el-input>   
 </el-form-item>  
 <el-form-item label="主机名称：" prop="hostname">  
 <el-input v-model="form.hostname" disabled></el-input>  
 </el-form-item>  
 <el-form-item label="IDC机房：" prop="idc">  
 <el-input v-model="form.idc"></el-input>  
 </el-form-item>  
 <el-form-item label="主机分组：" prop="server\_group">  
 <el-input v-model="form.server\_group"></el-input>  
 </el-form-item>  
 <el-form-item label="SSH IP：">  
 <el-input v-model="form.ssh\_ip"></el-input>  
 </el-form-item>  
 <el-form-item label="SSH 端口：">  
 <el-input v-model="form.ssh\_port" ></el-input>  
 </el-form-item>  
 <el-form-item label="机器类型：" prop="machine\_type">  
 <el-input v-model="form.machine\_type"></el-input>  
 </el-form-item>  
 <el-form-item label="系统版本：" prop="os\_version">  
 <el-input v-model="form.os\_version"></el-input>  
 </el-form-item>  
 <el-form-item label="公网IP：" prop="public\_ip">  
 <el-input v-model="form.public\_ip"></el-input>  
 </el-form-item>  
 <el-form-item label="私有IP：" prop="private\_ip">  
 <el-input v-model="form.private\_ip"></el-input>  
 </el-form-item>  
 <el-form-item label="CPU数量：" prop="cpu\_num">  
 <el-input v-model="form.cpu\_num"></el-input>  
 </el-form-item>  
 <el-form-item label="CPU型号：" prop="cpu\_model">  
 <el-input v-model="form.cpu\_model"></el-input>  
 </el-form-item>  
 <el-form-item label="内存：" prop="memory">  
 <el-input v-model="form.memory"></el-input>  
 </el-form-item>  
 <el-form-item label="硬盘：" prop="disk">  
 <el-input v-model="form.disk"></el-input>  
 </el-form-item>  
 <el-form-item label="上架日期：" prop="put\_shelves\_date">  
 <el-date-picker v-model="form.put\_shelves\_date" type="date" value-format="YYYY-MM-DD" placeholder="请选择日期">  
 </el-date-picker>  
 </el-form-item>  
 <el-form-item label="下架日期：" prop="off\_shelves\_date">  
 <el-date-picker v-model="form.off\_shelves\_date" type="date" value-format="YYYY-MM-DD" placeholder="请选择日期">  
 </el-date-picker>  
 </el-form-item>  
 <el-form-item label="租约过期时间：" prop="expire\_datetime">  
 <el-date-picker v-model="form.expire\_datetime" type="datetime" value-format="YYYY-MM-DD HH:mm:ss" placeholder="请选择时间">  
 </el-date-picker>  
 </el-form-item>  
 <el-form-item label="备注：">  
 <el-input v-model="form.note" type="textarea"></el-input>  
 </el-form-item>  
 </el-form>  
  
 <template #footer>  
 <span class="dialog-footer">  
 <el-button @click="dialogClose">取消</el-button>  
 <el-button type="primary" @click="submit">确定</el-button>  
 </span>  
 </template>  
</el-dialog>

##### 编辑创建框

<el-form-item label="机器名称：" prop="name">  
 <el-input v-model="form.name" placeholder="例如：测试机"></el-input>  
 </el-form-item>  
 <el-form-item label="主机名称：" prop="hostname">  
 <el-input v-model="form.hostname"></el-input>  
 </el-form-item>  
 <el-form-item label="IDC机房：" prop="idc">  
 <el-input v-model="form.idc"></el-input>  
 </el-form-item>  
 <el-form-item label="主机分组：" prop="server\_group">  
 <el-input v-model="form.server\_group"></el-input>  
 </el-form-item>  
 <el-form-item label="SSH IP：" prop="ssh\_ip">  
 <el-input v-model="form.ssh\_ip"></el-input>  
 </el-form-item>  
 <el-form-item label="SSH 端口：" prop="ssh\_port">  
 <el-input v-model="form.ssh\_port" ></el-input>  
 </el-form-item>  
 <el-form-item label="SSH凭据：" prop="credential\_id">  
 <el-input v-model="form.credential" ></el-input>  
 </el-form-item>  
</el-form>

formRules: {  
 idc: [  
 {required: true, message: '请选择IDC机房', trigger: 'change'},  
 ],  
 server\_group: [  
 {required: true, message: '请选择主机分组', trigger: 'change'},  
 ],  
 name: [  
 {required: true, message: '请输入机器名称', trigger: 'blur'},  
 {min: 2, message: '主机名长度应不小于2个字符', trigger: 'blur'}  
 ],  
 hostname: [  
 {required: true, message: '请输入主机名称', trigger: 'blur'},  
 {min: 4, message: '主机名长度不小于4个字符', trigger: 'blur'}  
 ],  
 ssh\_ip: [  
 {required: true, message: '请输入SSH IP地址', trigger: 'blur'},  
 {min: 7, message: '主机名长度不小于8个字符', trigger: 'blur'}  
 ],  
 ssh\_port: [  
 {required: true, message: '请输入SSH端口', trigger: 'blur'},  
 {min: 2, message: 'SSH端口长度不小于2个数字', trigger: 'blur'},  
 // {type: 'number', message: 'SSH端口必须是数字', trigger: 'blur'}  
 ],  
 credential\_id: [  
 {required: true, message: '请选择SSH连接凭据', trigger: 'change'},  
 ],  
},

#### 表格内容显示

##### SSH状态字段加样式

在表格中默认显示文本，并且数据库中存储的数据是约定的字符，无法直观看出，例如SSH状态，verified为验证，unverified为未验证。那么就需要对数据显示进行转换，定义插槽 template #default="scope"，scope值为 el-table绑定的data，scope.row为这行数据，获取字段值再结合判断显示不同样式。

SSH状态：

<el-table-column prop="is\_verified" label="SSH状态" width="120" sortable v-if="showColumn.is\_verified">  
 <template #default="scope">  
 <el-tag type="success" v-if="scope.row.is\_verified == 'verified'">已验证</el-tag>  
 <el-tag type="warning" v-if="scope.row.is\_verified == 'unverified'">未验证</el-tag>  
 </template>  
</el-table-column>

机器类型：

<el-table-column prop="machine\_type" label="机器类型" width="110" sortable v-if="showColumn.machine\_type">  
 <template #default="scope">  
 <span v-if="scope.row.machine\_type == 'vm'">虚拟机</span>  
 <span v-else-if="scope.row.machine\_type == 'cloud\_vm'">云主机</span>  
 <span v-else>物理机</span>  
 </template>  
</el-table-column>

##### IP、硬盘（数组）字段



IP

<el-table-column prop="public\_ip" label="公网IP" width="180" sortable v-if="showColumn.public\_ip">  
 <template #default="scope">  
 <el-tag type="info" v-for="(ip, index) in scope.row.public\_ip" :key="index">{{ip}}</el-tag>  
 </template>  
</el-table-column>

硬盘

<el-table-column prop="disk" label="硬盘" width="200" sortable v-if="showColumn.disk">  
 <template #default="scope">  
 <table style="background: #ebeef5;width: 100%" v-if="scope.row.disk"> <!--表格背景设置灰色，表格内默认白色-->  
 <thead>  
 <tr>  
 <th>设备</th>  
 <th>类型</th>  
 <th>容量</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr v-for="(disk, index) in scope.row.disk" :key="index">  
 <td>{{disk.device}}</td>  
 <td>{{disk.type}}</td>  
 <td>{{disk.size}}</td>  
 </tr>  
 </tbody>  
 </table>  
 <span v-else>{{scope.row.disk}}</span>  
 </template>  
</el-table-column>

##### 机房、分组（一对多、多对多）字段



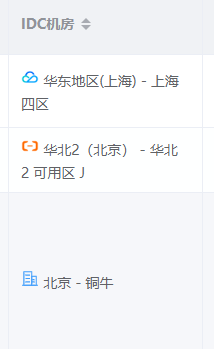
<el-table-column prop="idc" label="IDC机房" sortable v-if="showColumn.idc">  
 <template #default="scope">  
 {{scope.row.idc.city}}-{{scope.row.idc.name}}  
 </template>  
</el-table-column>

<el-table-column prop="server\_group" label="主机分组" sortable v-if="showColumn.server\_group">  
 <template #default="scope">  
 <el-tag class="ml-2" type="info" v-for="group in scope.row.server\_group" :key="group.id">{{group.name}}</el-tag>  
 </template>  
</el-table-column>

IDC：

<el-table-column prop="idc" label="IDC机房" sortable v-if="showColumn.idc">  
 <template #default="scope">  
 {{scope.row.idc.city}} - {{scope.row.idc.name}}  
 </template>  
</el-table-column>

增加图标：



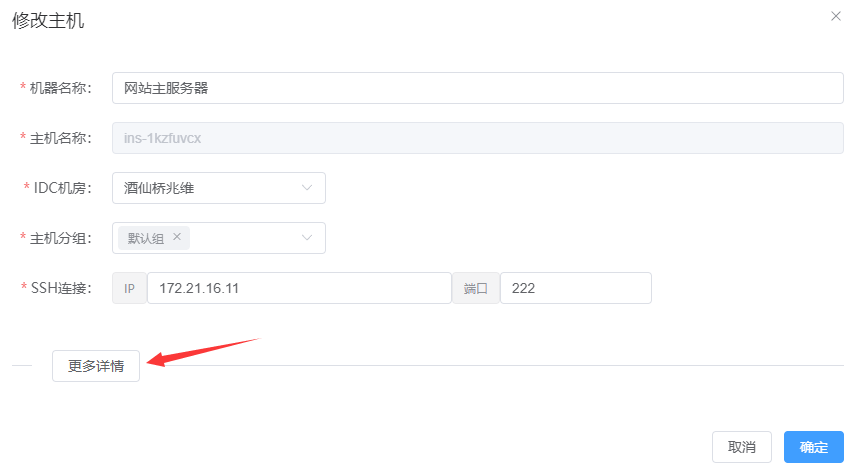
<el-table-column prop="idc" label="IDC机房" sortable v-if="showColumn.idc">  
 <template #default="scope">  
 <img src="../../assets/img/aliyun.png" style="width: 18px;height: 18px" v-if="scope.row.idc.provider == '阿里云'">  
 <img src="../../assets/img/tencend.png" style="width: 18px;height: 18px" v-else-if="scope.row.idc.provider == '腾讯云'">  
 <el-icon :size="18" color="#409EFC" v-else><office-building/></el-icon>  
 {{scope.row.idc.city}} - {{scope.row.idc.name}}  
 </template>  
</el-table-column>

分组：

<el-table-column prop="server\_group" label="主机分组" width="150" sortable v-if="showColumn.server\_group">  
 <template #default="scope">  
 <el-tag type="info" v-for="group in scope.row.server\_group" :key="group.id">{{group.name}}</el-tag>  
 </template>  
</el-table-column>

#### 操作栏：编辑

分为两部分：主机基本信息和详情，详情里是主机的配置，一般不改动，所以默认隐藏，点击时候显示。



##### 完善显示：IDC机房、主机分组和凭据

<el-form-item label="IDC机房：" prop="idc">  
 <el-select class="m-2" v-model="form.idc" @click="getIdc" placeholder="请选择">  
 <el-option v-for="row in idc" :key="row.id" :label="`${row.city}-${row.name}`" :value="row.id">  
 </el-option>  
 </el-select>  
</el-form-item>

<el-form-item label="主机分组：" prop="server\_group">  
 <el-select class="m-2" v-model="form.server\_group" @click="getServerGroup" multiple placeholder="请选择">  
 <el-option v-for="row in serverGroup" :key="row.id" :label="row.name" :value="row.id">  
 </el-option>  
 </el-select>  
</el-form-item>

<el-form-item label="SSH凭据：" prop="credential">  
 <el-select class="m-2" v-model="form.credential" @click="getCredential" placeholder="请选择">  
 <el-option v-for="row in credential" :key="row.id" :label="`${row.name}-${row.username}`" :value="row.id">  
 </el-option>  
 </el-select>  
</el-form-item>

设置变量：

idc: '',  
serverGroup: '',

credential: '',

事件方法：

getIdc() {  
 this.$http.get('/cmdb/idc/?page\_size=50')  
 .then(res => {  
 this.idc = res.data.data;  
 })  
},  
getServerGroup() {  
 this.$http.get('/cmdb/server\_group/?page\_size=50')  
 .then(res => {  
 this.serverGroup = res.data.data;  
 });  
},

getCredential() {  
 this.$http.get('/config/credential/?page\_size=50')  
 .then(res => {  
 this.credential = res.data.data;  
 });  
},

默认返回是第一页10条。主要指定条数。

由于获取的是对象，无法直接双向绑定。而select要求是数组。

// 监听窗口打开，  
watch: {  
 visible() {  
 if(this.visible) { // 关闭窗口不请求  
 this.$http.get('/cmdb/server/' + this.rowId + '/')  
 .then(res => {  
 if (res.data.code == 200) {  
 this.form = res.data.data;  
  
 // IDC机房：从对象中提取ID字段重新赋值  
 this.form.idc = res.data.data.idc.id;  
 // 主机分组：从对象中提取ID（多个）ID字段重新赋值  
 const group\_id = new Array();  
 const server\_group = res.data.data.server\_group;  
 for (let i in server\_group) {  
 group\_id.push(server\_group[i].id)  
 }  
 this.form.server\_group = group\_id;  
  
 // 重新渲染编辑对话框IDC机房和主机分组  
 this.getIdc();  
 this.getServerGroup();

this.getCredential();  
 }  
 });  
 }  
  
 }  
}

##### SSH连接



<el-form-item label="SSH连接：" required>  
 <el-col :span="1.5">  
 <el-tag size="large" type="info">IP</el-tag>  
 </el-col>  
 <el-col :span="10">  
 <el-form-item prop="ssh\_ip">  
 <el-input v-model="form.ssh\_ip"></el-input>  
 </el-form-item>  
 </el-col>  
 <el-col :span="1.5">  
 <el-tag size="large" type="info">端口</el-tag>  
 </el-col>  
 <el-col :span="5">  
 <el-form-item prop="ssh\_port">  
 <el-input v-model="form.ssh\_port"></el-input>  
 </el-form-item>  
 </el-col>  
</el-form-item>

##### 更多详情

<!--主机详情，默认不展示-->  
<br>  
<el-divider content-position="left"><el-button @click="serverDetailVisible ? serverDetailVisible = false : serverDetailVisible = true" plain>更多详情</el-button></el-divider>  
<div v-show="serverDetailVisible">  
<el-form-item label="机器类型：" prop="machine\_type">  
 <el-input v-model="form.machine\_type"></el-input>  
</el-form-item>  
<el-form-item label="系统版本：" prop="os\_version">  
 <el-input v-model="form.os\_version"></el-input>  
</el-form-item>  
<el-form-item label="公网IP：" prop="public\_ip">  
 <el-input v-model="form.public\_ip"></el-input>  
</el-form-item>  
<el-form-item label="私有IP：" prop="private\_ip">  
 <el-input v-model="form.private\_ip"></el-input>  
</el-form-item>  
<el-form-item label="CPU数量：" prop="cpu\_num">  
 <el-input v-model="form.cpu\_num"></el-input>  
</el-form-item>  
<el-form-item label="CPU型号：" prop="cpu\_model">  
 <el-input v-model="form.cpu\_model"></el-input>  
</el-form-item>  
<el-form-item label="内存：" prop="memory">  
 <el-input v-model="form.memory"></el-input>  
</el-form-item>  
<el-form-item label="硬盘：" prop="disk">  
 <el-input v-model="form.disk"></el-input>  
</el-form-item>  
<el-form-item label="上架日期：" prop="put\_shelves\_date">  
 <el-date-picker v-model="form.put\_shelves\_date" type="date" value-format="YYYY-MM-DD" placeholder="请选择日期">  
 </el-date-picker>  
</el-form-item>  
<el-form-item label="下架日期：" prop="off\_shelves\_date">  
 <el-date-picker v-model="form.off\_shelves\_date" type="date" value-format="YYYY-MM-DD" placeholder="请选择日期">  
 </el-date-picker>  
</el-form-item>  
<el-form-item label="租约过期时间：" prop="expire\_datetime">  
 <el-date-picker v-model="form.expire\_datetime" type="datetime" value-format="YYYY-MM-DD HH:mm:ss" placeholder="请选择时间">  
 </el-date-picker>  
</el-form-item>  
<el-form-item label="备注：">  
 <el-input v-model="form.note" type="textarea"></el-input>  
</el-form-item>  
</div>

##### 机器类型

<el-form-item label="机器类型：" prop="machine\_type">  
 <el-select v-model="form.machine\_type" class="m-2" placeholder="请选择">  
 <el-option label="虚拟机" value="vm"></el-option>  
 <el-option label="云主机" value="cloud\_vm"></el-option>  
 <el-option label="物理机" value="physical\_machine"></el-option>  
 </el-select>  
</el-form-item>

##### 硬盘

待优化

##### 提交

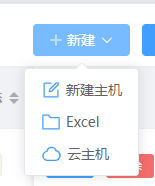
重写更新方法：

def update(self, request, \*args, \*\*kwargs):  
 partial = kwargs.pop('partial', False)  
 instance = self.get\_object()  
 serializer = self.get\_serializer(instance, data=request.data, partial=partial)  
 serializer.is\_valid(raise\_exception=True)

self.perform\_update(serializer)  
  
 hostname = request.data.get('hostname')  
 # 一对多  
 idc\_id = int(request.data.get('idc'))  
 idc\_obj = Idc.objects.get(id=idc\_id)  
 server\_obj = Server.objects.get(hostname=hostname)  
 server\_obj.idc = idc\_obj  
 server\_obj.save()  
  
 # 多对多  
 group\_id\_list = request.data.get('server\_group')  
 server = Server.objects.get(hostname=hostname)  
 server.server\_group.add(\*group\_id\_list)  
   
 res = {'code': 200, 'msg': '更新成功'}  
 return Response(res)

#### 创建

##### 增加下拉列表



在展示列上面加。

<!--新建主机-->  
<el-dropdown style="margin-right: 12px">  
 <el-button type="primary">  
 <el-icon><plus /></el-icon>&nbsp;新建<el-icon class="el-icon--right"><arrow-down /></el-icon>  
 </el-button>  
 <template #dropdown>  
 <el-dropdown-menu>  
 <el-dropdown-item @click="createDialogVisible = true"><el-icon color="#409EFC" :size="20"><Edit /></el-icon>新建主机</el-dropdown-item>  
 <el-dropdown-item><el-icon color="#409EFC" :size="20"><Folder /></el-icon>Excel</el-dropdown-item>  
 <el-dropdown-item @click="cloudDialogVisible = true"><el-icon color="#409EFC" :size="21"><MostlyCloudy /></el-icon>云主机</el-dropdown-item>  
 </el-dropdown-menu>  
 </template>  
</el-dropdown>

##### 新建主机



ServerCreate.vue

<template>  
 <el-dialog  
 :model-value="visible"  
 width="30%"  
 title="新建主机"  
 @close="dialogClose"  
 >  
 <el-form :model="form" ref="formRef" :rules="formRules" label-position=“right” label-width="100px" >  
 <el-form-item label="IDC机房：" prop="idc">  
 <el-select class="m-2" v-model="form.idc" @click="getIdc" placeholder="请选择">  
 <el-option v-for="row in idc" :key="row.id" :label="row.name" :value="row.id">  
 </el-option>  
 </el-select>  
 </el-form-item>  
 <el-form-item label="主机分组：" prop="server\_group">  
 <el-select class="m-2" multiple v-model="form.server\_group" @click="getServerGroup" placeholder="请选择">  
 <el-option v-for="row in serverGroup" :key="row.id" :label="row.name" :value="row.id">  
 </el-option>  
 </el-select>  
 </el-form-item>  
 <el-form-item label="机器名称：" prop="name">  
 <el-input v-model="form.name" placeholder="例如：测试机"></el-input>  
 </el-form-item>  
 <el-form-item label="主机名称：" prop="hostname">  
 <el-input v-model="form.hostname"></el-input>  
 <el-tag class="ml-2" type="warning"><el-icon><InfoFilled/></el-icon>必须与实际主机名一致</el-tag>  
 </el-form-item>  
 <el-form-item label="SSH连接：" required>  
 <el-col :span="1.5">  
 <el-tag size="large" type="info">IP</el-tag>  
 </el-col>  
 <el-col :span="10">  
 <el-form-item prop="ssh\_ip">  
 <el-input v-model="form.ssh\_ip"></el-input>  
 </el-form-item>  
 </el-col>  
 <el-col :span="1.5">  
 <el-tag size="large" type="info">端口</el-tag>  
 </el-col>  
 <el-col :span="5">  
 <el-form-item prop="ssh\_port">  
 <el-input v-model="form.ssh\_port"></el-input>  
 </el-form-item>  
 </el-col>  
 </el-form-item>  
 <el-form-item label="SSH凭据：" prop="credential">  
 <el-select class="m-2" v-model="form.credential" @click="getCredential" placeholder="请选择">  
 <el-option v-for="row in credential" :key="row.id" :label="`${row.name}-${row.username}`" :value="row.id">  
 </el-option>  
 </el-select>  
 <el-col :span="3" :offset="1">  
 <el-button type="primary" plain @click="this.$router.push('/config/credential')">新建</el-button>  
 </el-col>  
 </el-form-item>  
 <el-form-item label="备注：" prop="note">  
 <el-input v-model="form.note" type="textarea"></el-input>  
 </el-form-item>  
 </el-form>  
  
 <template #footer>  
 <span class="dialog-footer">  
 <el-button @click="dialogClose">取消</el-button>  
 <el-button type="primary" @click="submit">确定</el-button>  
 </span>  
 </template>  
 </el-dialog>  
  
</template>  
  
<script>  
 export default {  
 name: "IdcCreate",  
 props: {  
 visible: Boolean,  
 },  
 data() {  
 return {  
 form: {  
 'idc': '',  
 'server\_group': '',  
 'name': '',  
 'hostname': '',  
 'ssh\_ip': '',  
 'ssh\_port': null,  
 'credential': '',  
 'note': ''  
 },  
 formRules: {  
 idc: [  
 {required: true, message: '请选择IDC机房', trigger: 'blur'},  
 ],  
 server\_group: [  
 {required: true, message: '请选择主机分组', trigger: 'blur'},  
 ],  
 name: [  
 {required: true, message: '请输入机器名称', trigger: 'blur'},  
 {min: 2, message: '主机名长度应不小于2个字符', trigger: 'blur'}  
 ],  
 hostname: [  
 {required: true, message: '请输入主机名称', trigger: 'blur'},  
 {min: 4, message: '主机名长度不小于4个字符', trigger: 'blur'}  
 ],  
 ssh\_ip: [  
 {required: true, message: '请输入SSH IP地址', trigger: 'blur'},  
 {min: 7, message: '主机名长度不小于8个字符', trigger: 'blur'}  
 ],  
 ssh\_port: [  
 {required: true, message: '请输入SSH端口', trigger: 'blur'},  
 {min: 2, message: 'SSH端口长度不小于2个数字', trigger: 'blur'},  
 // {type: 'number', message: 'SSH端口必须是数字', trigger: 'blur'}  
 ],  
 credential: [  
 {required: true, message: '请选择SSH连接凭据', trigger: 'change'},  
 ],  
 },  
 idc: '',  
 serverGroup: '',  
 credential: '',  
 }  
 },  
 methods: {  
 submit() {  
 this.$refs.formRef.validate((valid) => {  
 if (valid) {  
 this.$http.post('/cmdb/create\_host/', this.form)  
 .then(res => {  
 if (res.data.code == 200){  
 this.$message.success('创建成功');  
 this.$parent.getData(); // 调用父组件方法，更新数据  
 this.dialogClose() // 关闭窗口  
 }  
 })  
 } else {  
 this.$message.error('格式错误！')  
 }  
 })  
 },  
 // 点击关闭，子组件通知父组件更新属性  
 dialogClose() {  
 this.$emit('update:visible', false) // 父组件必须使用v-model  
 },  
 getIdc() {  
 this.$http.get('/cmdb/idc/?page\_size=50')  
 .then(res => {  
 this.idc = res.data.data;  
 })  
 },  
 getServerGroup() {  
 this.$http.get('/cmdb/server\_group/?page\_size=50')  
 .then(res => {  
 this.serverGroup = res.data.data;  
 });  
 },  
 getCredential() {  
 this.$http.get('/config/credential/?page\_size=50')  
 .then(res => {  
 this.credential = res.data.data;  
 });  
 },  
 }  
 }  
</script>

##### Excel导入



ServerCreateExcel.vue

<template>  
 <el-dialog  
 :model-value="visible"  
 width="30%"  
 title="Excel导入"  
 @close="dialogClose"  
 >  
 <el-form :model="form" ref="formRef" :rules="formRules" label-position=“right” label-width="100px" >  
 <el-form-item label="模板下载：">

<el-link :href="excelTemplateDown" type="primary">主机导入模板</el-link>  
 </el-form-item>  
 <el-form-item label="IDC机房：" prop="idc">  
 <el-select class="m-2" v-model="form.idc" @click="getIdc" placeholder="请选择">  
 <el-option v-for="row in idc" :key="row.id" :label="row.name" :value="row.id">  
 </el-option>  
 </el-select>  
 </el-form-item>  
 <el-form-item label="主机分组：" prop="server\_group">  
 <el-select class="m-2" multiple v-model="form.server\_group" @click="getServerGroup" placeholder="请选择">  
 <el-option v-for="row in serverGroup" :key="row.id" :label="row.name" :value="row.id">  
 </el-option>  
 </el-select>  
 </el-form-item>  
 <el-form-item label="导入数据：">  
 <el-upload  
 :limit="1"  
 :file-list="fileList"  
 :auto-upload="false"  
 >  
 <template #trigger>  
 <el-button type="primary">选择文件</el-button>  
 </template>  
 <template #tip>  
 <div class="el-upload\_\_tip text-red">  
 导入完成后可通过验证功能批量验证  
 </div>  
 </template>  
 </el-upload>  
 </el-form-item>  
 </el-form>  
  
 <template #footer>  
 <span class="dialog-footer">  
 <el-button @click="dialogClose">取消</el-button>  
 <el-button type="primary" @click="submit">确定</el-button>  
 </span>  
 </template>  
 </el-dialog>  
  
</template>  
  
<script>  
 export default {  
 name: "IdcCreate",  
 props: {  
 visible: Boolean,  
 },  
 data() {  
 return {  
 form: [],  
 formRules: {  
 idc: [  
 {required: true, message: '请选择IDC机房', trigger: 'blur'},  
 ],  
 server\_group: [  
 {required: true, message: '请选择主机分组', trigger: 'blur'},  
 ]  
 },  
 idc: '',  
 serverGroup: '',  
 excelTemplateDown: 'http://127.0.0.1:8001/api/cmdb/excel\_create\_host/',  
 fileList: [],  
 }  
 },  
 methods: {  
 submit() {  
 this.$refs.formRef.validate((valid) => {  
 if (valid) {  
 let fd = new FormData();  
 fd.append("file", this.fileList[0].raw);  
 fd.append('idc', this.form.idc);  
 fd.append('server\_group', this.form.server\_group);  
 this.$http.post('/cmdb/excel\_create\_host/', fd)  
 .then(res => {  
 if (res.data.code == 200){  
 this.$message.success('创建成功');  
 this.$parent.getData(); // 调用父组件方法，更新数据  
 this.dialogClose() // 关闭窗口  
 }  
 })  
 } else {  
 this.$message.error('格式错误！')  
 }  
 })  
 },  
 // 点击关闭，子组件通知父组件更新属性  
 dialogClose() {  
 this.$emit('update:visible', false) // 父组件必须使用v-model  
 },  
 getIdc() {  
 console.log(this.$http.instance);  
 this.$http.get('/cmdb/idc/?page\_size=50')  
 .then(res => {  
 this.idc = res.data.data;  
 })  
 },  
 getServerGroup() {  
 this.$http.get('/cmdb/server\_group/?page\_size=50')  
 .then(res => {  
 this.serverGroup = res.data.data;  
 });  
 },  
 }  
 }  
</script>

##### 新建：云主机





1、定义3个步骤

2、定义每步form表单

3、定义3个盒子对象包括每步表单项

4、3个盒子v-show设置active等于对应步数显示

5、定义取消、上一步、下一步、确定按钮

<template>  
 <el-dialog  
 :model-value="visible"  
 width="30%"  
 title="导入云主机"  
 @close="dialogClose"  
 >  
 <el-steps :space="200" :active="active" align-center style="margin-bottom: 10%">  
 <el-step title="公有云"></el-step>  
 <el-step title="访问凭据"></el-step>  
 <el-step title="导入确认"></el-step>  
 </el-steps>  
  
 <el-form :model="form" ref="formRef" :rules="formRules" label-position="right" label-width="155px">  
 <!--第一步-->  
 <div v-show="active == 1">  
 <el-form-item prop="cloud" label-width="180px">  
 <el-radio-group v-model="form.cloud">  
 <el-radio label="aliyun"><img src="../../assets/img/aliyun.png"></el-radio>  
 <el-radio label="tencent"><img src="../../assets/img/tencend.png"></el-radio>  
 </el-radio-group>  
 </el-form-item>  
 </div>  
 <!--第二步-->  
 <div v-show="active == 2">  
 <el-form-item label="AccessKey ID：" prop="secret\_id">  
 <el-input v-model="form.secret\_id" clearable></el-input>  
 </el-form-item>  
 <el-form-item label="AccessKey Secret：" prop="secret\_key">  
 <el-input v-model="form.secret\_key" clearable></el-input>  
 <el-link v-if="form.cloud == 'aliyun'" href="https://ram.console.aliyun.com/manage/ak" target="\_blank" type="primary">如何获取AccessKey？</el-link>  
 <el-link v-if="form.cloud == 'tencent'" href="https://console.cloud.tencent.com/cam/capi" target="\_blank" type="primary">如何获取AccessKey？</el-link>  
 </el-form-item>  
 </div>  
 <!--第三步-->  
 <div v-show="active == 3">  
 <el-form-item label="选择区域：" prop="region">  
 <el-select class="m-2" v-model="form.region" @click="getCloudRegion" placeholder="请选择" style="width: 100%">  
 <el-option v-for="row in cloudRegion" :key="row.region\_id" :label="row.region\_name" :value="row.region\_id">  
 </el-option>  
 </el-select>  
 </el-form-item>  
 <el-form-item label="选择分组：" prop="server\_group">  
 <el-select class="m-2" v-model="form.server\_group" @click="getServerGroup" placeholder="请选择" style="width: 100%">  
 <el-option v-for="row in serverGroup" :key="row.id" :label="row.name" :value="row.id">  
 </el-option>  
 </el-select>  
 </el-form-item>  
 <el-form-item label="SSH连接：" required>  
 <el-col :span="1.5">  
 <el-tag size="large" type="info">IP</el-tag>  
 </el-col>  
 <el-col :span="7" :offset="1">  
 <el-radio-group v-model="form.ssh\_ip">  
 <el-radio label="public" >公网</el-radio>  
 <el-radio label="private" >内网</el-radio>  
 </el-radio-group>  
 </el-col>  
 <el-col :span="1.5">  
 <el-tag size="large" type="info">端口</el-tag>  
 </el-col>  
 <el-col :span="5">  
 <el-form-item prop="ssh\_port">  
 <el-input v-model="form.ssh\_port"></el-input>  
 </el-form-item>  
 </el-col>  
 </el-form-item>  
 </div>  
 </el-form>  
  
 <template #footer>  
 <span class="dialog-footer">  
 <el-button @click="dialogClose" v-if="active == 1">取消</el-button>  
 <el-button type="primary" @click="dialogToggle('pre')" v-if="active > 1">上一步</el-button>  
 <el-button type="primary" @click="dialogToggle('next')" v-if="active < 3">下一步</el-button>  
 <el-button type="primary" @click="submit" v-if="active == 3">确定</el-button>  
 </span>  
 </template>  
 </el-dialog>  
  
</template>  
  
<script>  
 export default {  
 name: "IdcCreate",  
 props: {  
 visible: Boolean,  
 },  
 data() {  
 return {  
 form: {  
 'cloud': '',  
 'secret\_id': '',  
 'secret\_key': '',  
 'region': '',  
 'server\_group': '',  
 'ssh\_ip': 'private', // public或private  
 'ssh\_port': 22  
 },  
 formRules: {  
 cloud: [  
 {required: true, message: '请选择', trigger: 'blur'}  
 ],  
 secret\_id: [  
 {required: true, message: '请输入密钥ID', trigger: 'blur'},  
 {min: 20, message: '密钥ID长度应不小于20个字符', trigger: 'blur'}  
 ],  
 secret\_key: [  
 {required: true, message: '请输入密钥Key', trigger: 'blur'},  
 {min: 20, message: '密钥Key长度应不小于20个字符', trigger: 'blur'}  
 ],  
 region: [  
 {required: true, message: '请选择IDC机房', trigger: 'change'},  
 ],  
 server\_group: [  
 {required: true, message: '请选择主机分组', trigger: 'change'},  
 ],  
 ssh\_port: [  
 {required: true, message: '请输入端口', trigger: 'blur'},  
 ],  
 },  
 cloudRegion: '',  
 active: 1,  
 idc: '',  
 serverGroup: '',  
 }  
 },  
 methods: {  
 submit() {  
 this.$refs.formRef.validate((valid) => {  
 if (valid) {  
 if (this.form.cloud == 'aliyun') {  
 this.$http.post('/cmdb/aliyun\_cloud/', this.form)  
 .then(res => {  
 if (res.data.code == 200){  
 this.$message.success('导入云主机成功');  
 this.$parent.getData();  
 this.dialogClose() // 关闭窗口  
 }  
 })  
 } else if (this.form.cloud == 'tencent') {  
 this.$http.post('/cmdb/tencent\_cloud/', this.form)  
 .then(res => {  
 if (res.data.code == 200){  
 this.$message.success('导入云主机成功');  
 this.$parent.getData();  
 this.dialogClose() // 关闭窗口  
 }  
 })  
 }  
 this.cloudDialogVisible = false;  
 } else {  
 this.$message.error('格式错误！')  
 }  
 });  
 },  
 // 点击关闭，子组件通知父组件更新属性  
 dialogClose() {  
 this.$emit('update:visible', false) // 父组件必须使用v-model  
 },  
 // 云主机导入第一步和第二步对话框切换  
 dialogToggle(action) {  
 if (action == 'pre') {  
 if (this.active-- < 2) {  
 this.active = 1;  
 }  
 } else if (action == 'next') {  
 if (this.active++ > 3) {  
 this.active = 1;  
 }  
 }  
 },  
 // 获取区域  
 getCloudRegion() {  
 if (this.form.cloud == 'aliyun') {  
 // 携带密钥请求  
 this.$http.get('/cmdb/aliyun\_cloud/', {params: this.form})  
 .then(res => {  
 this.cloudRegion = res.data.data;  
 });  
 } else if (this.form.cloud == 'tencent') {  
 this.$http.get('/cmdb/tencent\_cloud/', {params: this.form})  
 .then(res => {  
 this.cloudRegion = res.data.data;  
 });  
 } else {  
 this.$message.error('请选择公有云！');  
 }  
 },  
 getIdc() {  
 this.$http.get('/cmdb/idc/?page\_size=50')  
 .then(res => {  
 this.idc = res.data.data;  
 })  
 },  
 getServerGroup() {  
 this.$http.get('/cmdb/server\_group/?page\_size=50')  
 .then(res => {  
 this.serverGroup = res.data.data;  
 });  
 },  
 }  
 }  
</script>

#### 查询



<div style="margin-bottom: 20px;display: flex;align-items: center;justify-content: space-between">  
 <div>  
 <el-row>  
 <el-col :span="7">  
 <el-input  
 v-model="urlParams.search"  
 placeholder="搜索名称，主机名，IP"  
 @keyup.enter="onSearch"  
 clearable  
 @clear="onSearch"  
 class="search"  
 />  
 </el-col>  
  
 <!--选择IDC-->  
 <el-col :span="7">  
 <el-select v-model="urlParams.idc" class="m-2" @click="getIdc" clearable placeholder="IDC机房">  
 <el-option v-for="row in idc" :key="row.id" :label="row.name" :value="row.id">  
 </el-option>  
 </el-select>  
 </el-col>  
 <!--选择分组-->  
 <el-col :span="7">  
 <el-select v-model="urlParams.server\_group" class="m-2" @click="getServerGroup" clearable placeholder="主机分组">  
 <el-option v-for="row in serverGroup" :key="row.id" :label="row.name" :value="row.id">  
 </el-option>  
 </el-select>  
 </el-col>  
  
 <el-col :span="2">  
 <el-button type="primary" @click="onSearch"><el-icon><search /></el-icon>&nbsp;搜索</el-button>  
 </el-col>  
 </el-row>  
 </div>

</div>

urlParams: { // URL查询参数，传递服务端，放这里方便修改值  
 page\_num: 1,  
 page\_size: this.pageSize,  
 search: '',  
 idc: '',  
 server\_group: ''  
},

getIdc() {  
 this.$http.get('/cmdb/idc/?page\_size=50')  
 .then(res => {  
 this.idc = res.data.data;  
 })  
},  
getServerGroup() {  
 this.$http.get('/cmdb/server\_group/?page\_size=50')  
 .then(res => {  
 this.serverGroup = res.data.data;  
 });  
},

在getData() 带上查询参数。

getData() {  
 if(!this.urlParams.idc) {  
 delete this.urlParams.idc  
 }  
 if(!this.urlParams.server\_group) {  
 delete this.urlParams.server\_group  
 }  
 this.$http.get('/cmdb/server/', {params: this.urlParams})  
 .then(res => {  
 this.tableData = res.data.data;  
 this.total = res.data.count;  
 })  
},

后端：

search\_fields = ('name', 'hostname', 'public\_ip', 'private\_ip')  
filter\_fields = ('idc','server\_group')

#### 同步按钮

<!--操作栏：点击同步选择凭据对话框-->  
<el-dialog  
 v-model="credentialVisible"  
 width="15%"  
 title="请选择SSH连接凭据"  
 >  
 <el-col :span="2">  
 <el-icon :size="23"><lock/></el-icon>  
 </el-col>  
 <el-col :span="22" :offset="1">  
 <el-select class="m-2" v-model="credentialId" @click="getCredential" placeholder="请选择">  
 <el-option v-for="row in credential" :key="row.id" :label="`${row.name}-${row.username}`" :value="row.id">  
 </el-option>  
 </el-select>  
 </el-col>  
 <template #footer>  
 <span class="dialog-footer">  
 <el-button @click="credentialVisible = false">取消</el-button>  
 <el-button type="primary" @click="sync">确定</el-button>  
 </span>  
 </template>  
</el-dialog>

credential: '',   
credentialId: '', // 选择凭据ID  
credentialVisible: false // 选择凭据对话框显示与隐藏

// 选择凭据后确认  
sync() {  
 this.$http.get('/cmdb/host\_collect/', {params: {  
 'hostname': this.currentRow.hostname,  
 'credential\_id': this.credentialId}  
 })  
 .then(res => {  
 if (res.data.code == 200) {  
 this.$message.success('同步成功');  
 this.getData();  
 }  
 })  
},  
// 操作栏：同步  
handleSync(index, row) {  
 // 如果主机有凭据直接同步  
 this.currentRow = row; // 临时存储，供弹出框确认使用  
 if(row.credential) {  
 this.$http.get('/cmdb/host\_collect/', {params: {'hostname': row.hostname}})  
 .then(res => {  
 if (res.data.code == 200) {  
 this.$message.success('同步成功');  
 this.getData();  
 }  
 })  
 } else {  
 // 弹出选择凭据  
 this.credentialVisible = true;  
 }  
},

后端：

class HostCollectView(APIView):  
 def get(self, request):  
 hostname = request.query\_params.get('hostname')  
 server = Server.objects.get(hostname=hostname)  
 ssh\_ip = server.ssh\_ip  
 ssh\_port = server.ssh\_port  
  
 # 未绑定凭据并且没有选择凭据  
 credential\_id = request.query\_params.get('credential\_id')  
 if not server.credential and not credential\_id:  
 result = {'code': 500, 'msg': '未发现凭据，请选择！'}  
 return Response(result)  
 elif server.credential:  
 credential\_id = int(server.credential.id)  
 elif credential\_id:  
 credential\_id = int(request.query\_params.get('credential\_id'))  
  
 credential = Credential.objects.get(id=credential\_id)  
  
 username = credential.username  
 if credential.auth\_mode == 1:  
 password = credential.password  
 ssh = SSH(ssh\_ip, ssh\_port, username, password=password)  
 else:  
 private\_key = credential.private\_key  
 ssh = SSH(ssh\_ip, ssh\_port, username, key=private\_key)  
  
 # 先SSH基本测试  
 test = ssh.test()  
 if test['code'] == 200:  
 client\_agent\_name = "host\_collect.py"  
 local\_file = os.path.join(settings.BASE\_DIR, 'cmdb', 'files', client\_agent\_name)  
 remote\_file = os.path.join(settings.CLIENT\_COLLECT\_DIR, client\_agent\_name) # 这个工作路径在setting里配置  
 ssh.scp(local\_file, remote\_file=remote\_file)  
 ssh.command('chmod +x %s' % remote\_file)  
 result = ssh.command('python %s' % remote\_file)  
  
 if result['code'] == 200: # 采集脚本执行成功  
 # 再进一步判断客户端采集脚本提交结果  
 data = json.loads(result['data'])  
 Server.objects.filter(hostname=hostname).update(\*\*data)  
  
 # 更新凭据ID  
 server.credential = credential  
 server.is\_verified = 'verified'  
 server.save()  
  
 result = {'code': 200, 'msg': '主机配置同步成功'}  
  
 else:  
 result = {'code': 500, 'msg': '主机配置同步失败！错误：%s' %result['msg']}  
 else:  
 result = {'code': 500, 'msg': 'SSH连接异常！错误：%s' %test['msg']}  
 return Response(result)

### 4、凭据管理

<!--展示列弹出框-->  
<el-popover placement="left" :width="100" v-model:visible="columnVisible">  
 <template #reference>  
 <el-button type="primary" @click="columnVisible = true"><el-icon><setting /></el-icon>&nbsp;展示列</el-button>  
 </template>  
 <el-checkbox v-model="showColumn.name" disabled>机房名称</el-checkbox>  
 <el-checkbox v-model="showColumn.auth\_mode">认证方式</el-checkbox>  
 <el-checkbox v-model="showColumn.username">用户名</el-checkbox>  
 <el-checkbox v-model="showColumn.password">密码</el-checkbox>  
 <el-checkbox v-model="showColumn.private\_key">私钥</el-checkbox>  
 <el-checkbox v-model="showColumn.note">备注</el-checkbox>  
 <el-checkbox v-model="showColumn.create\_time">创建时间</el-checkbox>  
 <el-checkbox v-model="showColumn.update\_time">更新时间</el-checkbox>  
 <div style="text-align: right; margin: 0">  
 <el-button size="small" type="text" @click="columnVisible = false">取消</el-button>  
 <el-button size="small" type="primary" @click="saveColumn">确认</el-button>  
 </div>  
</el-popover>

数据表格显示字段：

<el-table-column prop="name" label="凭据名称" fixed="left" sortable v-if="showColumn.name"/>  
<el-table-column prop="auth\_mode" label="认证方式" sortable v-if="showColumn.auth\_mode">  
 <template #default="scope">  
 <el-tag type="primary" v-if="scope.row.auth\_mode == 1">密码</el-tag>  
 <el-tag type="primary" v-if="scope.row.auth\_mode == 2">秘钥</el-tag>  
 </template>  
</el-table-column>  
<el-table-column prop="username" label="用户名" sortable v-if="showColumn.username"/>  
<el-table-column prop="password" label="密码" sortable v-if="showColumn.password"/>  
<el-table-column prop="private\_key" label="私钥" sortable :show-overflow-tooltip="true" v-if="showColumn.private\_key"/>  
<el-table-column prop="note" label="备注" v-if="showColumn.note"/>  
<el-table-column prop="create\_time" label="创建时间" sortable v-if="showColumn.create\_time"/>  
<el-table-column prop="update\_time" label="更新时间" sortable v-if="showColumn.update\_time"/>

CredentialEdit.vue

<template>  
 <!--操作栏：编辑对话框-->  
 <el-dialog  
 :model-value="visible"  
 width="30%"  
 title="修改凭据"  
 @close="dialogClose"  
 >  
 <el-form :model="form" ref="formRef" :rules="formRules" label-position="right" label-width="100px">  
 <el-form-item label="凭据名称：" prop="name">  
 <el-input v-model="form.name"></el-input>  
 </el-form-item>  
 <el-form-item label="用户名：" prop="username">  
 <el-input v-model="form.username"></el-input>  
 </el-form-item>  
 <el-form-item label="密码：" prop="password" v-if="form.auth\_mode==1">  
 <el-input v-model="form.password" type="password" show-password></el-input>  
 </el-form-item>  
 <el-form-item label="私钥：" prop="private\_key" v-if="form.auth\_mode==2">  
 <el-input v-model="form.private\_key" type="textarea"></el-input>  
 </el-form-item>  
 <el-form-item label="备注：">  
 <el-input v-model="form.note" type="textarea"></el-input>  
 </el-form-item>  
 </el-form>  
  
 <template #footer>  
 <span class="dialog-footer">  
 <el-button @click="dialogClose">取消</el-button>  
 <el-button type="primary" @click="submit">确定</el-button>  
 </span>  
 </template>  
 </el-dialog>  
</template>  
  
<script>  
 export default {  
 name: "IdcEdit",  
 props: {  
 visible: Boolean,  
 rowId: '', // 当前行ID  
 },  
 data() {  
 return {  
 form: [],  
 formRules: {  
 name: [  
 {required: true, message: '请输入凭据名称', trigger: 'blur'},  
 {min: 2, message: '凭据名称长度应不小于2个字符', trigger: 'blur'}  
 ],  
 username: [  
 {required: true, message: '请输入用户名', trigger: 'blur'},  
 {min: 2, message: '用户名长度应不小于2个字符', trigger: 'blur'}  
 ],  
 password: [  
 {required: true, message: '请输入密码', trigger: 'blur'},  
 {min: 6, message: '密码长度应不小于6个字符', trigger: 'blur'}  
 ],  
 private\_key: [  
 {required: true, message: '请输入私钥', trigger: 'blur'},  
 {min: 20, message: '私钥长度应不小于20个字符', trigger: 'blur'}  
 ]  
 },  
 }  
 },  
 methods: {  
 submit() {  
 this.$refs.formRef.validate((valid) => {  
 if (valid) {  
 this.$http.put('/config/credential/' + this.rowId + '/', this.form)  
 .then(res => {  
 if (res.data.code == 200){  
 this.$message.success('修改成功');  
 this.$parent.getData(); // 调用父组件方法，更新数据  
 this.dialogClose() // 关闭窗口  
 }  
 })  
 } else {  
 this.$message.error('格式错误！')  
 }  
 })  
 },  
 // 点击关闭，子组件通知父组件更新属性  
 dialogClose() {  
 this.$emit('update:visible', false) // 父组件必须使用v-model  
 }  
 },  
 // 监听窗口打开，  
 watch: {  
 visible() {  
 if(this.visible) { // 关闭窗口不请求  
 this.$http.get('/config/credential/' + this.rowId + '/')  
 .then(res => {  
 if (res.data.code == 200) {  
 this.form = res.data.data;  
 }  
 });  
 }  
 }  
 }  
 }  
</script>

CredentialCreate.vue

<template>  
 <el-dialog  
 :model-value="visible"  
 width="30%"  
 title="创建凭据"  
 @close="dialogClose"  
 >  
 <el-form :model="form" ref="formRef" :rules="formRules" label-position="right" label-width="100px" >  
 <el-form-item label="凭据名称：" prop="name">  
 <el-input v-model="form.name"></el-input>  
 </el-form-item>  
 <el-form-item label="用户名：" prop="username">  
 <el-input v-model="form.username"></el-input>  
 </el-form-item>  
 <el-form-item label="认证方式：">  
 <el-radio-group v-model="form.auth\_mode">  
 <el-radio border label="1" >密码</el-radio>  
 <el-radio border label="2" >秘钥</el-radio>  
 </el-radio-group>  
 </el-form-item>  
 <el-form-item label="密码：" prop="password" v-if="form.auth\_mode=='1'">  
 <el-input v-model="form.password" type="password" show-password></el-input>  
 </el-form-item>  
 <el-form-item label="私钥：" prop="private\_key" v-if="form.auth\_mode=='2'">  
 <el-input v-model="form.private\_key" type="textarea"></el-input>  
 </el-form-item>  
 <el-form-item label="备注：">  
 <el-input v-model="form.note" type="textarea"></el-input>  
 </el-form-item>  
 </el-form>  
  
 <template #footer>  
 <span class="dialog-footer">  
 <el-button @click="dialogClose">取消</el-button>  
 <el-button type="primary" @click="submit">确定</el-button>  
 </span>  
 </template>  
 </el-dialog>  
  
</template>  
  
<script>  
 export default {  
 name: "IdcCreate",  
 props: {  
 visible: Boolean,  
 },  
 data() {  
 return {  
 form: {  
 'name': '',  
 'username': '',  
 'auth\_mode': '1',  
 'password': '',  
 'private\_key': '',  
 'note': ''  
 },  
 formRules: {  
 name: [  
 {required: true, message: '请输入凭据名称', trigger: 'blur'},  
 {min: 2, message: '凭据名称长度应不小于2个字符', trigger: 'blur'}  
 ],  
 username: [  
 {required: true, message: '请输入用户名', trigger: 'blur'},  
 {min: 2, message: '用户名长度应不小于2个字符', trigger: 'blur'}  
 ],  
 password: [  
 {required: true, message: '请输入密码', trigger: 'blur'},  
 {min: 6, message: '密码长度应不小于6个字符', trigger: 'blur'}  
 ],  
 private\_key: [  
 {required: true, message: '请输入私钥', trigger: 'blur'},  
 {min: 20, message: '私钥长度应不小于20个字符', trigger: 'blur'}  
 ]  
 },  
 }  
 },  
 methods: {  
 submit() {  
 this.$refs.formRef.validate((valid) => {  
 if (valid) {  
 this.$http.post('/config/credential/', this.form)  
 .then(res => {  
 if (res.data.code == 200){  
 this.$message.success('创建成功');  
 this.$parent.getData(); // 调用父组件方法，更新数据  
 this.dialogClose() // 关闭窗口  
 }  
 })  
 } else {  
 this.$message.error('格式错误！')  
 }  
 })  
 },  
 // 点击关闭，子组件通知父组件更新属性  
 dialogClose() {  
 this.$emit('update:visible', false) // 父组件必须使用v-model  
 }  
 }  
 }  
</script>  
  
<style scoped>  
  
</style>

视图：

from rest\_framework.viewsets import ModelViewSet  
from system\_config.serializers import CredentialSerializer  
from system\_config.models import Credential  
from rest\_framework.response import Response  
  
from django\_filters.rest\_framework import DjangoFilterBackend  
from rest\_framework import filters  
  
class CredentialViewSet(ModelViewSet):  
  
 queryset = Credential.objects.all()  
 serializer\_class = CredentialSerializer  
  
 filter\_backends = [filters.SearchFilter, filters.OrderingFilter, DjangoFilterBackend] # 指定过滤器  
 search\_fields = ('name', 'note')  
 filter\_fields = ('auth\_mode',)  
  
 def update(self, request, \*args, \*\*kwargs):  
 partial = kwargs.pop('partial', False)  
 instance = self.get\_object()  
 serializer = self.get\_serializer(instance, data=request.data, partial=partial)  
 serializer.is\_valid(raise\_exception=True)  
 self.perform\_update(serializer)  
 res = {'code': 200, 'msg': '更新成功'}  
 return Response(res)  
  
 def create(self, request, \*args, \*\*kwargs):  
 serializer = self.get\_serializer(data=request.data)  
 serializer.is\_valid(raise\_exception=True)  
 self.perform\_create(serializer)  
 res = {'code': 200, 'msg': '创建成功'}  
 return Response(res)