day85

内容回顾和补充

1. restful规范
2. drf组件认证的实现过程?
3. drf组件中权限的实现过程?
4. drf组件中节流的实现方式?

   ```
   - 实现原理
   - 具体流程
   ```

5. 什么是jwt？优势?

   ```
   一般在前后端分离时，用于做用户认证（登录）使用的技术。
   jwt的实现原理：
       - 用户登录成功之后，会给前端返回一段token。
       - token是由.分割的三段组成。
           - 第一段：类型和算法信心
           - 第二段：用户信息+超时时间
           - 第三段：hs256（前两段拼接）加密 + base64url
       - 以后前端再次发来信息时
           - 超时验证
           - token合法性校验
   优势：
       - token只在前端保存，后端只负责校验。
       - 内部集成了超时时间，后端可以根据时间进行校验是否超时。
       - 由于内部存在hash256加密，所以用户不可以修改token，只要一修改就认证失败。
   ```

今日概要

- 呼啦圈作业
- paramiko模块
- 练习题：django + paramiko实现远程对某些服务器执行命令+上传文件

今日详细

1.写视图的方法

- 第一种：原始APIView

  ```
  url(r'^login/$',account.LoginView.as_view()),
  ```

```
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework_jwt.settings import api_settings
from rest_framework.throttling import AnonRateThrottle
from api import models


class LoginView(APIView):
    authentication_classes = []
    def post(self,request,*args,**kwargs):
        # 1.根据用户名和密码检测用户是否可以登录
        user =
models.UserInfo.objects.filter(username=request.data.get('username'),password=request.data.get('password')).first()
        if not user:
            return Response({'code':10001,'error':'用户名或密码错误'})

        # 2. 根据user对象生成payload（中间值的数据）
        jwt_payload_handler = api_settings.JWT_PAYLOAD_HANDLER
        payload = jwt_payload_handler(user)

        # 3. 构造前面数据，base64加密；中间数据base64加密；前两段拼接然后做hs256加密（加盐），再做base64加密。生成token
        jwt_encode_handler = api_settings.JWT_ENCODE_HANDLER
        token = jwt_encode_handler(payload)
        return Response({'code': 10000, 'data': token})
```

- 第二种：ListApiView等

```
url(r'^article/$',article.ArticleView.as_view()),
url(r'^article/(?P<pk>\d+)/$',article.ArticleDetailView.as_view()),
```

```
from rest_framework.throttling import AnonRateThrottle
from rest_framework.response import Response
from rest_framework.generics import ListAPIView,RetrieveAPIView
from api import models
from api.serializer.article import ArticleSerializer,ArticleDetailSerializer

class ArticleView(ListAPIView):
    authentication_classes = []
    # throttle_classes = [AnonRateThrottle,]

    queryset = models.Article.objects.all()
    serializer_class = ArticleSerializer

class ArticleDetailView(RetrieveAPIView):
    authentication_classes = []
    queryset = models.Article.objects.all()
    serializer_class = ArticleDetailSerializer
```

- 第三种：

```
url(r'^article/$',article.ArticleView.as_view({"get":'list','post':'create'}
)),
    url(r'^article/(?
P<pk>\d+)/$',article.ArticleView.as_view({'get':'retrieve','put':'update','p
atch':'partial_update','delete':'destroy'}))
```

```
from rest_framework.viewsets import GenericViewSet
from rest_framework.mixins import
ListModelMixin,RetrieveModelMixin,CreateModelMixin,UpdateModelMixin,DestroyM
odelMixin
from api.serializer.article import ArticleSerializer,ArticleDetailSerializer

class
ArticleView(GenericViewSet,ListModelMixin,RetrieveModelMixin,CreateModelMixi
n,UpdateModelMixin,DestroyModelMixin):
    authentication_classes = []
    throttle_classes = [AnonRateThrottle,]

    queryset = models.Article.objects.all()
    serializer_class = None

    def get_serializer_class(self):
        pk = self.kwargs.get('pk')
        if pk:
            return ArticleDetailSerializer
        return ArticleSerializer
```

drf 相关知识点梳理

1. 装饰器

```
def outer(func):
    def inner(*args,**kwargs):
        return func(*args,**kwargs)
    return inner

@outer
def index(a1):
    pass

index()
```

```python
def outer(func):
    def inner(*args,**kwargs):
        return func(*args,**kwargs)
    return inner

def index(a1):
    pass

index = outer(index)

index()
```

2. django中可以免除csrftoken认证

```python
from django.views.decorators.csrf import csrf_exempt
from django.shortcuts import HttpResponse

@csrf_exempt
def index(request):
    return HttpResponse('...')

# index = csrf_exempt(index)

urlpatterns = [
    url(r'^index/$',index),
]
```

```python
urlpatterns = [
    url(r'^login/$',account.LoginView.as_view()),
]

class APIView(View):
    @classmethod
    def as_view(cls, **initkwargs):
        view = super().as_view(**initkwargs)
        view.cls = cls
        view.initkwargs = initkwargs

        # Note: session based authentication is explicitly CSRF validated,
        # all other authentication is CSRF exempt.
        return csrf_exempt(view)
```

3. 面向对象中基于继承+异常处理来做的约束

```
class BaseVersioning:
    def determine_version(self, request, *args, **kwargs):
        raise NotImplementedError("must be implemented")

class URLPathVersioning(BaseVersioning):
    def determine_version(self, request, *args, **kwargs):
        version = kwargs.get(self.version_param, self.default_version)
        if version is None:
            version = self.default_version

        if not self.is_allowed_version(version):
            raise exceptions.NotFound(self.invalid_version_message)
        return version
```

4. 面向对象封装

```
class Foo(object):
    def __init__(self,name,age):
        self.name = name
        self.age = age

obj = Foo('汪洋',18)
```

```
class APIView(View):
    def dispatch(self, request, *args, **kwargs):

        self.args = args
        self.kwargs = kwargs
        request = self.initialize_request(request, *args, **kwargs)
        self.request = request
        ...

    def initialize_request(self, request, *args, **kwargs):
        """
        Returns the initial request object.
        """
        parser_context = self.get_parser_context(request)

        return Request(
            request,
            parsers=self.get_parsers(),
            authenticators=self.get_authenticators(), #
[MyAuthentication(),]
            negotiator=self.get_content_negotiator(),
            parser_context=parser_context
        )
```

5. 面向对象继承

```
class View(object):
    pass

class APIView(View):
    def dispatch(self):
        method = getattr(self,'get')
        method()

class GenericAPIView(APIView):
    serilizer_class = None

    def get_seriliser_class(self):
        return self.serilizer_class

class ListModelMixin(object):
    def get(self):
        ser_class = self.get_seriliser_class()
        print(ser_class)

class ListAPIView(ListModelMixin,GenericAPIView):
    pass

class UserInfoView(ListAPIView):
    pass


view = UserInfoView()
view.dispatch()
```

```
class View(object):
    pass

class APIView(View):
    def dispatch(self):
        method = getattr(self,'get')
        method()

class GenericAPIView(APIView):
    serilizer_class = None

    def get_seriliser_class(self):
        return self.serilizer_class

class ListModelMixin(object):
    def get(self):
        ser_class = self.get_seriliser_class()
        print(ser_class)

class ListAPIView(ListModelMixin,GenericAPIView):
    pass

class UserInfoView(ListAPIView):
    serilizer_class = "汪洋"


view = UserInfoView()
view.dispatch()
```

```
class View(object):
    pass

class APIView(View):
    def dispatch(self):
        method = getattr(self,'get')
        method()

class GenericAPIView(APIView):
    serilizer_class = None

    def get_seriliser_class(self):
        return self.serilizer_class

class ListModelMixin(object):
    def get(self):
        ser_class = self.get_seriliser_class()
        print(ser_class)

class ListAPIView(ListModelMixin,GenericAPIView):
    pass

class UserInfoView(ListAPIView):

    def get_seriliser_class(self):
        return "咩咩"

view = UserInfoView()
view.dispatch()
```

6. 反射

```
class View(object):
    def dispatch(self, request, *args, **kwargs):
        # Try to dispatch to the right method; if a method doesn't exist,
        # defer to the error handler. Also defer to the error handler if the
        # request method isn't on the approved list.
        if request.method.lower() in self.http_method_names:
            handler = getattr(self, request.method.lower(),
self.http_method_not_allowed)
        else:
            handler = self.http_method_not_allowed
        return handler(request, *args, **kwargs)
```

7. 发送ajax请求

```

$.ajax({
    url:'地址',
    type:'GET',
    data:{...},
    success:function(arg){
        console.log(arg);
    }
})
```

8. 浏览器具有 "同源策略的限制"，导致 `发送ajax请求` + `跨域` 存在无法获取数据。
    - 简单请求，发送一次请求。
    - 复杂请求，先options请求做预检，然后再发送真正请求

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1>常鑫的网站</h1>
    <p>
        <input type="button" value="点我" onclick="sendMsg()">
    </p>
    <p>
        <input type="button" value="点他" onclick="sendRemoteMsg()">
    </p>


    <script src="https://cdn.bootcss.com/jquery/3.4.1/jquery.min.js">
</script>
    <script>
        function sendMsg() {
            $.ajax({
                url:'/msg/',
                type:'GET',
                success:function (arg) {
                    console.log(arg);
                }
            })
        }
        function sendRemoteMsg() {
            $.ajax({
                url:'http://127.0.0.1:8002/json/',
                type:'GET',
                success:function (arg) {
                    console.log(arg);
                }
            })

        }
    </script>
</body>
</html>
```

9. 如何解决ajax+跨域?

```
CORS，跨站资源共享，本质：设置响应头。
```

10. 常见的Http请求方法

```
get
post
put
patch
delete
options
```

11. http请求中Content-type请起头

```
情况一：
    content-type:x-www-form-urlencode
    name=alex&age=19&xx=10

    request.POST和request.body中均有值。

情况二：
    content-type:application/json
    {"name":"ALex","Age":19}

    request.POST没值
    request.body有值。
```

12. django中F查询
13. django中获取空Queryset

```
models.User.object.all().none()
```

14. 基于django的fbv和cbv都能实现遵循restful规范的接口

```python


def user(request):
    if request.metho == 'GET':
        pass


class UserView(View):
    def get()...

    def post...
```

15. 基于django rest framework框架实现restful api的开发。

```
- 免除csrf认证
- 视图（APIView、ListAPIView、ListModelMinx）
- 版本
- 认证
- 权限
- 节流
- 解析器
- 筛选器
- 分页
- 序列化
- 渲染器
```

16. 简述drf中认证流程?
17. 简述drf中节流的实现原理以及过程? 匿名用户/非匿名用户 如何实现频率限制?
18. GenericAPIView视图类的作用?

```
他提供了一些规则，例如：

class GenericAPIView(APIView):
    serializer_class = None
    queryset = None
    lookup_field = 'pk'

    filter_backends = api_settings.DEFAULT_FILTER_BACKENDS
    pagination_class = api_settings.DEFAULT_PAGINATION_CLASS

    def get_queryset(self):
        return self.queryset

    def get_serializer_class(self):
        return self.serializer_class

    def filter_queryset(self, queryset):
        for backend in list(self.filter_backends):
            queryset = backend().filter_queryset(self.request, queryset,
self)
        return queryset

    @property
    def paginator(self):
        if not hasattr(self, '_paginator'):
            if self.pagination_class is None:
                self._paginator = None
            else:
                self._paginator = self.pagination_class()
        return self._paginator

他相当于提供了一些规则，建议子类中使用固定的方式获取数据，例如：
class ArticleView(GenericAPIView):
    queryset = models.User.objects.all()

    def get(self,request,*args,**kwargs):
        query = self.get_queryset()

我们可以自己继承GenericAPIView来实现具体操作，但是一般不会，因为更加麻烦。
而GenericAPIView主要是提供给drf内部的 ListAPIView、Create....
class ListModelMixin:
    def list(self, request, *args, **kwargs):
        queryset = self.filter_queryset(self.get_queryset())

        page = self.paginate_queryset(queryset)
        if page is not None:
            serializer = self.get_serializer(page, many=True)
            return self.get_paginated_response(serializer.data)

        serializer = self.get_serializer(queryset, many=True)
        return Response(serializer.data)
```

```
class ListAPIView(mixins.ListModelMixin,GenericAPIView):
    def get(self, request, *args, **kwargs):
        return self.list(request, *args, **kwargs)

class MyView(ListAPIView):
    queryset = xxxx
    ser...
```

```
总结：GenericAPIView主要为drf内部帮助我们提供增删改查的类LIstAPIView、CreateAPIView、UpdateAPIView、提供了执行流程和功能，我们在使用drf内置类做CURD时，就可以通过自定义 静态字段（类变量）或重写方法（get_queryset、get_serializer_class）来进行更高级的定制。
```

19. jwt以及其优势。
20. 序列化时many=True和many=False的区别?
21. 应用DRF中的功能进行项目开发

```
*****
    解析器:request.query_parmas/request.data
    视图
    序列化
    渲染器：Response

****
    request对象封装
    版本处理
    分页处理
***
    认证
    权限
    节流
```

- 基于APIView实现呼啦圈
- 继承ListAPIView+ GenericViewSet,ListModelMixin实现呼啦圈

2.paramiko

用于帮助开发者通过代码远程连接服务器，并对服务器进行操作。

```
pip3 install paramiko
```

- 远程执行命令【用户名和密码】

```
import paramiko

# 创建SSH对象
ssh = paramiko.SSHClient()

# 允许连接不在know_hosts文件中的主机
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
```

```
# 连接服务器
ssh.connect(hostname='192.168.16.85', port=22, username='root',
password='123456')

# 执行命令
stdin, stdout, stderr = ssh.exec_command('df')
# 获取命令结果
result = stdout.read()
# 关闭连接
ssh.close()

print(result.decode('utf-8'))
```

- 远程执行命令【公钥和私钥】(公钥必须提前上传到服务器)

```
import paramiko

private_key =
paramiko.RSAKey.from_private_key_file(r'C:/Users/Administrator/.ssh/id_rsa')

# 创建SSH对象
ssh = paramiko.SSHClient()
# 允许连接不在know_hosts文件中的主机
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
# 连接服务器
ssh.connect(hostname='192.168.16.85', port=22, username='root',
pkey=private_key)

# 执行命令
stdin, stdout, stderr = ssh.exec_command('df')
# 获取命令结果
result = stdout.read()

# 关闭连接
ssh.close()

print(result)
```

- 远程上传和下载文件【用户名和密码】

```
import paramiko

transport = paramiko.Transport(('192.168.16.85', 22))
transport.connect(username='root', password='123456')
sftp = paramiko.SFTPClient.from_transport(transport)


# 将location.py 上传至服务器 /tmp/test.py
# sftp.put('wy.txt', '/data/wy.txt')
sftp.get('/data/wy.txt', 'xx.txt')

transport.close()
```

- 远程上传和下载文件【公钥和私钥】

```python
import paramiko

private_key =
paramiko.RSAKey.from_private_key_file(r'C:/Users/Administrator/.ssh/id_rsa')

transport = paramiko.Transport(('192.168.16.85', 22))
transport.connect(username='root', pkey=private_key)

sftp = paramiko.SFTPClient.from_transport(transport)
# 将location.py 上传至服务器 /tmp/test.py
# sftp.put('/tmp/location.py', '/tmp/test.py')

# 将remove_path 下载到本地 local_path
# sftp.get('remove_path', 'local_path')

transport.close()
```

补充：通过私钥字符串也可以连接远程服务器。

```python
key = """-----BEGIN RSA PRIVATE KEY-----
MIIG5AIBAAKCAYEAu0fkMInsVRnIBSiZcVYhKuccWCh6hapYgB1eSOWZLz3+xFGy
G5p2z8HgiHzfT838gAm+5OajuyAuE4+fHI77LXSg+pLbr1FhPVKAP+nbsnLgvHty
ykZmt74CKKvZ08wdM7eUWJbkdpRNWmkwHBi99LeO0zYbHdXQ+m0P9EiWfdacJdAV
RDVCghQo1/IpfSUECpfQK1Hc0126vI8nhtrvT3V9qF420U1fwW9GJrODl71WRqvJ
BgSsKsjV16f0RKARESNmtA2vEdvMeutttZoO4FbvZ+iLKpcRM4LGm2+odryr8ijv
dCPCLVvoDExOPuqP1dgt5MWcCWf6ZNhMwAs/yvRHAKetvo5gtz8YvzwlikopCLM7
bS6C6woyppMHfIPjoGJ6JuKpeaWtAgugOw/oVvj1rRYoCv48R13NftqhkFD1KD8z
km9CjDC8hv+2DmIedtjvVwUz2QF4PN/RC/i1jo3+3rbP1DLu9emTHiortBBrpQ5o
K+y4Rzv+6NusD6DHAgMBAAECggGBAJ4hTaNOUaZpZmI0rZrsxoSbL2ugghNqid9i
7MFQW89v4TWSZXi5K6iwYw3bohKYMqNJl01fENBnk4AgvJA4ig0PdP0eEzAs3pYQ
mwlcRIygQvHiqkHwv7pVTS1aLUqQBfgtAazre2xEPCwitOSEX5/JfWcJQEwoxZMt
k1MIF0mZc67Zy5sT/Vwn+XScnDt2jbsEBFkPfg1aDto3ZYCQS5Aj/D21j0OauUdy
1SDIYkw1Kivx0IKsX1Kg0S6OOcnX/B6YrJvisrlQDeZnWlTsTyKSVTekIybJjUHE
ZgLIIbifSbTW1Bv1iCkDAJBd4Cj4txjXPIgea9ylZ39wSDSV5Pxu0t/M3YbdA26j
quVFCKqskNOC+cdYrdtVSij2Ypwov67HYsXC/w32oKO7tiRqy51LAs/WXMwQeS5a
8oWDZLiYIntY4TCYTVOvFlLRtXb+1SbwWKjJdjKvdChv4eo/Ov5JEXD2FVbVC/5E
Qo3jyjIrt1lrwXUdpJa0/iz4UV33wQKBwQDprCPZVCI7yK/BWTmUvCcupotNk6CC
+QIKDcvVxz63YFD5nXto4uG7ywXR6pEwOwmycO0CBuouvlPdSioQ3RYi6k0EO3Ch
9dybC5RZ3MENBHROHvU3mp01EWPUYnXAwNpvknujJqfXMxyURZvvox7hOnu/s3m4
C3eCBrMMg+uqNZDbLqAymw3pMGhHVWjy5oO8eLuLeJv6er+XoSSPNb21Da7StdQS
fBPQ1H0/+RXnhFJOzANc4mRZcXMCNGVZX6MCgcEAzSz3evuCRQ47AaSOrDd89jAw
PgpT+PG4gWw1jFZqHTbQ8MUl3YnElOVoaWRdIdDeslg9THg1cs5Yc9RrbIibyQjV
F9k/DlXGoOF//Mgtmr7JkLP3syRl+EedRbu2Gk67XDrV7XIvhdlsEuSnEK9xOiB6
ngewM0e4TccqlLsb6u7RNMU9IjMu/iMcBXKsZ9Cr/DENmGQlTaRVt7G6UcAYGNgQ
toMoCQWjR/HihlZHssLBj9U8uPyD38HKGy2OoXyNAoHBAKQzv9lHYusJ4l+G+IyJ
DyucAsXX2HJQ0tsHyNYHtg2cVCqkPIV+8UtKpmNVZwMyaWUIL7Q98bA5NKuLIzZI
dfbBGK/BqStWntgg8fWXx90C5UvEO2MAdjpFZxZmvgJeQuEmWVVTo5v4obubkrF5
ughhVXZng0AOZsNrO8Suqxsnmww6nn4RMVxNFOoTnbUawTXezUN71HfWa+38Ybl0
9UNWQyR0e3slz7LurrkWqwrOlBwlBrPtrsCflUbWVOXR6wKBwDFq+Dy14V2SnOG7
aeXPA5kkaCo5QJqAVglOL+OaWLqqnk6vnXwrl56pVqmz0762WT0phbIqbe02CBX1
/t3IVYVpTDIPUGG6hTqDJzmSWXGhLFlfD3Ulei3/ycCnAqh5eCUxwp8LVqjtgltW
mWqqZyIx+nafsW/YgWqyYu4p1wKR/O+x5hSbsWDiwfgJ876ZgyMeCYE/9cAqqb6x
3webtfId8ICVPIpXwkks2Hu0wlYrFIX5PUPtBjJZsb0ODtuUbQKBwF5BfytRZOZ/
6ktTfHj1OJ93hJNF9iRGpRfbHNylriVRb+hjXR3LBk8tyMAqR4rZDzfBNfPip5en
4TBMg8UATf43dVm7nv4PM2e24CRCWXMXYl7G3lFsQF/g7JNUoyr6bZQBf3pQcBw4
```

```
IJ38IcKV+L475tP4rfDrqyJz7mcJ+90a+ai5cSr9XoZqviAqNdhvBq5LjGOLkcdN
bSONAVVoGqjqIY/tOd2NMTEF6kVoYfJ7ZJtjxk/R3sdbdtajV3YsAg==
-----END RSA PRIVATE KEY-----"""


import paramiko
from io import StringIO

private_key = paramiko.RSAKey(file_obj=StringIO(key))

# 创建SSH对象
ssh = paramiko.SSHClient()
# 允许连接不在know_hosts文件中的主机
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
# 连接服务器
ssh.connect(hostname='192.168.16.85', port=22, username='root',
pkey=private_key)

# 执行命令
stdin, stdout, stderr = ssh.exec_command('df')
# 获取命令结果
result = stdout.read()

# 关闭连接
ssh.close()

print(result)
```

公司员工基于xshell连接服务器

- 用户名和密码
- 公钥和私钥（rsa）
    - 生成公钥和私钥

      ```
      ssh-keygen.exe -m pem

      在当前用户家目录会生成： .ssh/id_rsa.pub    .ssh/id_rsa
      ```

    - 把公钥放到服务器

      ```
      ssh-copy-id -i ~.ssh/id_rsa.pub root@192.168.16.85
      ```

    - 以后再连接服务器时，不需要在输入密码

      ```
      ssh root@192.168.16.85
      ```

作业

- 写脚本

id	hostname
1	192.168.16.85
2	192.168.16.84
3	192.168.16.83

```
1. 运行脚本
2. 要求用户输入要在远程服务器执行的命令。
    cmd = input('请输入要执行的命令：') # ifconfig

3. 通过pymysql模块去数据库中读取主机列表。
4. 在所有的主机中执行此命令，并打印出每台服务器上执行命令结果。

知识点：for循环逐一执行；线程池可以提供并发。
```

前提：公钥和私钥自己生成并上传。