day84 restful相关

内容回顾

1. restful规范?

```
第一步：整体说restful规范是什么？

第二步：再详细说restful建议
    1. https代替http,保证数据传输时安全。
    2. 在url中一般要体现api标识，这样看到url就知道他是一个api。
        http://www.luffycity.com/api/....（建议，因为他不会存在跨域的问题）
        http://api.luffycity.com/....
        假设：
            前段：https://www.luffycity.com/home
            后端：https://www.luffycity.com/api/
    3. 在接口中要体现版本
        http://www.luffycity.com/api/v1....（建议，因为他不会存在跨域的问题）
        注意：版本还可以放在请求头中
            http://www.luffycity.com/api/
            accept: ...

    4. restful也称为面向资源编程，视网络上的一切都是资源，对资源可以进行操作，所以一般资源都用名词。
        http://www.luffycity.com/api/user/

    5. 如果要加入一些筛选条件，可以添加在url中
        http://www.luffycity.com/api/user/?page=1&type=9

    6. 根据method不同做不同操作。

    7. 返回给用户状态码
        - 200，成功
        - 300，301永久 /302临时
        - 400，403拒绝 /404找不到
        - 500，服务端代码错误

        很多公司：
                def get(self,request,*args,**kwargs):
                    result = {'code':1000,'data':None,'error':None}
                    try:
                        val = int('你好')
                    except Exception as e:
                        result['code'] = 10001
                        result['error'] = '数据转换错误'

                    return Response(result)

    8. 返回值
        GET http://www.luffycity.com/api/user/
            [
                {'id':1,'name':'alex','age':19},
                {'id':1,'name':'alex','age':19},
```

```
        ]
    POST http://www.luffycity.com/api/user/
        {'id':1,'name':'alex','age':19}

    GET http://www.luffycity.com/api/user/2/
        {'id':2,'name':'alex','age':19}

    PUT http://www.luffycity.com/api/user/2/
        {'id':2,'name':'alex','age':19}

    PATCH https//www.luffycity.com/api/user/2/
        {'id':2,'name':'alex','age':19}

    DELETE https//www.luffycity.com/api/user/2/
        空
```

9. 操作异常时，要返回错误信息

```
    {
        error: "Invalid API key"
    }
```

10. 对于下一个请求要返回一些接口：Hypermedia AP

```
    {
        'id':2,
        'name':'alex',
        'age':19,
        'depart': "http://www.luffycity.com/api/user/30/"
    }
```

2. drf框架

记忆：请求到来之后，先执行视图的dispatch方法。

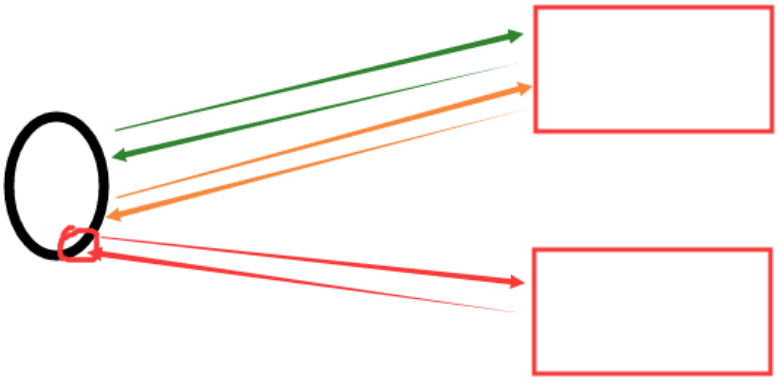
1. 视图
2. 版本处理
3. 认证
4. 权限
5. 节流（频率限制）
6. 解析器
7. 筛选器
8. 分页
9. 序列化
10. 渲染

今日概要

- 跨域
- drf访问频率的限制（了解）
- jwt

1.跨域

由于浏览器具有“同源策略”的限制。
如果在同一个域下发送ajax请求，浏览器的同源策略不会阻止。
如果在不同域下发送ajax，浏览器的同源策略会阻止。



总结

- 域相同，永远不会存在跨域。
  - crm，非前后端分离，没有跨域。
  - 路飞学城，前后端分离，没有跨域（之前有，现在没有）。
- 域不同时，才会存在跨域。
  - l拉勾网，前后端分离，存在跨域（设置响应头解决跨域）

解决跨域：CORS

本质在数据返回值设置响应头

```
from django.shortcuts import render,HttpResponse

def json(request):
    response = HttpResponse("JSONasdfasdf")
    response['Access-Control-Allow-Origin'] = "*"
    return response
```

跨域时，发送了2次请求?

在跨域时，发送的请求会分为两种：

- 简单请求，发一次请求。

  设置响应头就可以解决

  ```
  from django.shortcuts import render,HttpResponse

  def json(request):
      response = HttpResponse("JSONasdfasdf")
      response['Access-Control-Allow-Origin'] = "*"
      return response
  ```

- 复杂请求，发两次请求。
  - 预检
  - 请求

```
@csrf_exempt
def put_json(request):
    response = HttpResponse("JSON复杂请求")
    if request.method == 'OPTIONS':
        # 处理预检
        response['Access-Control-Allow-Origin'] = "*"
        response['Access-Control-Allow-Methods'] = "PUT"
        return response
    elif request.method == "PUT":
        return response
```

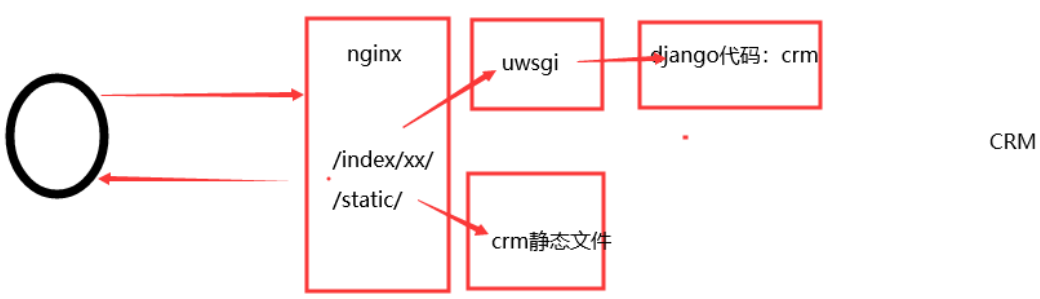
```
条件：
    1、请求方式：HEAD、GET、POST
    2、请求头信息：
        Accept
        Accept-Language
        Content-Language
        Last-Event-ID
        Content-Type 对应的值是以下三个中的任意一个
                                application/x-www-form-urlencoded
                                multipart/form-data
                                text/plain

注意：同时满足以上两个条件时，则是简单请求，否则为复杂请求
```

总结

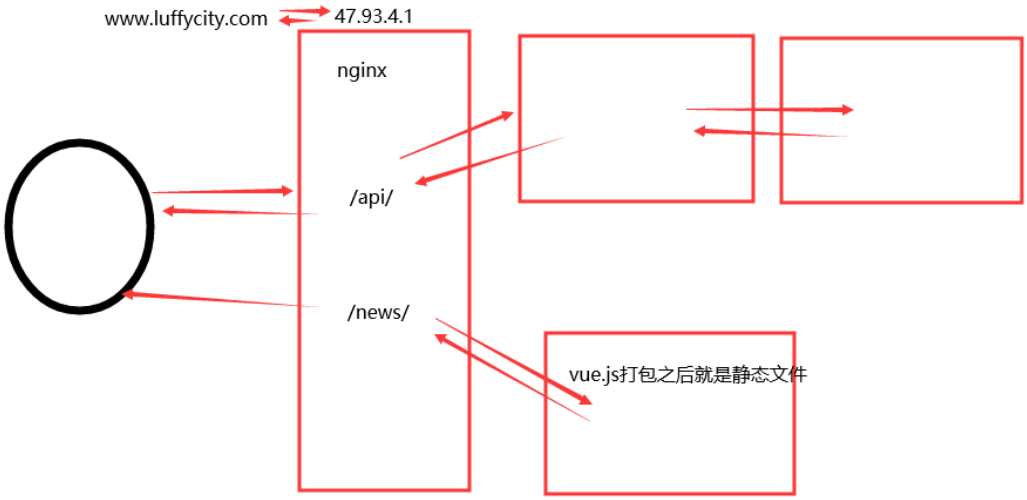
1. 由于浏览器具有“同源策略”的限制，所以在浏览器上跨域发送Ajax请求时，会被浏览器阻止。
2. 解决跨域
   - 不跨域
   - CORS（跨站资源共享，本质是设置响应头来解决）。
     - 简单请求：发送一次请求
     - 复杂请求：发送两次请求

2.项目部署

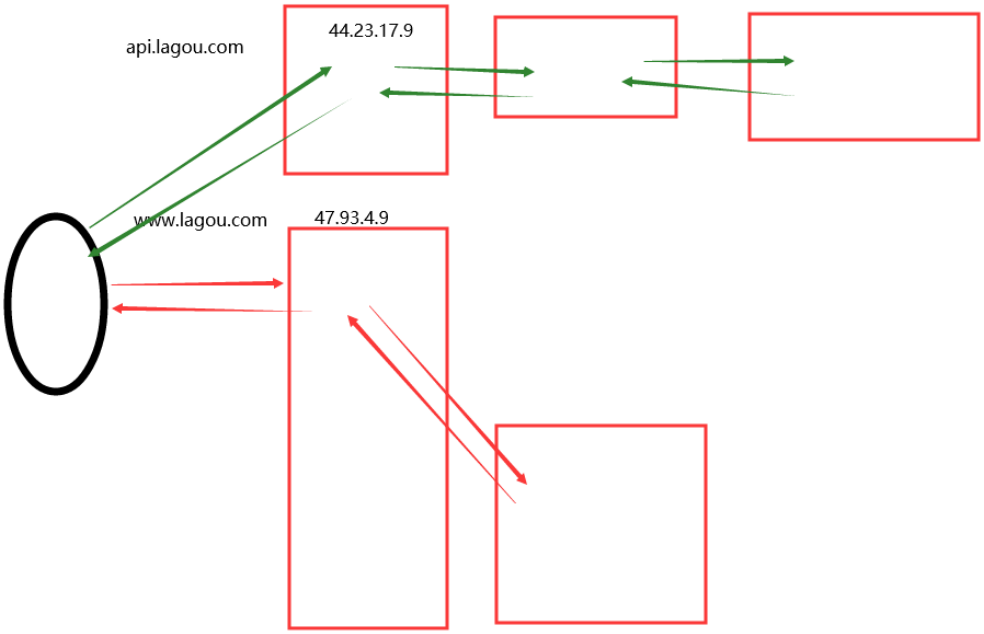
- crm部署



- 路飞部署



- 拉勾部署



3.drf的访问频率限制

- 频率限制在认证、权限之后
- 知识点

```
{
    throttle_anon_1.1.1.1:[100121340,],
    1.1.1.2:[100121251,100120450,]
}


限制：60s能访问3次
来访问时：
    1.获取当前时间 100121280
    2.100121280-60 = 100121220，小于100121220所有记录删除
    3.判断1分钟以内已经访问多少次了？ 4
    4.无法访问
```

```
停一会
来访问时：
    1.获取当前时间 100121340
    2.100121340-60 = 100121280,小于100121280所有记录删除
    3.判断1分钟以内已经访问多少次了？ 0
    4.可以访问
```

源码

```python
from rest_framework.views import APIView
from rest_framework.response import Response

from rest_framework.throttling import AnonRateThrottle,BaseThrottle

class ArticleView(APIView):
    throttle_classes = [AnonRateThrottle,]
    def get(self,request,*args,**kwargs):
        return Response('文章列表')

class ArticleDetailView(APIView):
    def get(self,request,*args,**kwargs):
        return Response('文章列表')
```

```python
class BaseThrottle:
    """
    Rate throttling of requests.
    """

    def allow_request(self, request, view):
        """
        Return `True` if the request should be allowed, `False` otherwise.
        """
        raise NotImplementedError('.allow_request() must be overridden')

    def get_ident(self, request):
        """
        Identify the machine making the request by parsing HTTP_X_FORWARDED_FOR
        if present and number of proxies is > 0. If not use all of
        HTTP_X_FORWARDED_FOR if it is available, if not use REMOTE_ADDR.
        """
        xff = request.META.get('HTTP_X_FORWARDED_FOR')
        remote_addr = request.META.get('REMOTE_ADDR')
        num_proxies = api_settings.NUM_PROXIES

        if num_proxies is not None:
            if num_proxies == 0 or xff is None:
                return remote_addr
            addrs = xff.split(',')
            client_addr = addrs[-min(num_proxies, len(addrs))]
            return client_addr.strip()

        return ''.join(xff.split()) if xff else remote_addr

    def wait(self):
        """
        Optionally, return a recommended number of seconds to wait before
```

```
        the next request.
        """
        return None


class SimpleRateThrottle(BaseThrottle):
    """
    A simple cache implementation, that only requires `.get_cache_key()`
    to be overridden.

    The rate (requests / seconds) is set by a `rate` attribute on the View
    class.  The attribute is a string of the form 'number_of_requests/period'.

    Period should be one of: ('s', 'sec', 'm', 'min', 'h', 'hour', 'd', 'day')

    Previous request information used for throttling is stored in the cache.
    """
    cache = default_cache
    timer = time.time
    cache_format = 'throttle_%(scope)s_%(ident)s'
    scope = None
    THROTTLE_RATES = api_settings.DEFAULT_THROTTLE_RATES

    def __init__(self):
        if not getattr(self, 'rate', None):
            self.rate = self.get_rate()
        self.num_requests, self.duration = self.parse_rate(self.rate)

    def get_cache_key(self, request, view):
        """
        Should return a unique cache-key which can be used for throttling.
        Must be overridden.

        May return `None` if the request should not be throttled.
        """
        raise NotImplementedError('.get_cache_key() must be overridden')

    def get_rate(self):
        """
        Determine the string representation of the allowed request rate.
        """
        if not getattr(self, 'scope', None):
            msg = ("You must set either `.scope` or `.rate` for '%s' throttle" %
                   self.__class__.__name__)
            raise ImproperlyConfigured(msg)

        try:
            return self.THROTTLE_RATES[self.scope]
        except KeyError:
            msg = "No default throttle rate set for '%s' scope" % self.scope
            raise ImproperlyConfigured(msg)

    def parse_rate(self, rate):
        """
        Given the request rate string, return a two tuple of:
        <allowed number of requests>, <period of time in seconds>
        """
        if rate is None:
```

```
            return (None, None)
        num, period = rate.split('/')
        num_requests = int(num)
        duration = {'s': 1, 'm': 60, 'h': 3600, 'd': 86400}[period[0]]
        return (num_requests, duration)

    def allow_request(self, request, view):
        """
        Implement the check to see if the request should be throttled.

        On success calls `throttle_success`.
        On failure calls `throttle_failure`.
        """
        if self.rate is None:
            return True

        # 获取请求用户的IP
        self.key = self.get_cache_key(request, view)
        if self.key is None:
            return True

        # 根据IP获取他的所有访问记录，[]
        self.history = self.cache.get(self.key, [])

        self.now = self.timer()

        # Drop any requests from the history which have now passed the
        # throttle duration
        while self.history and self.history[-1] <= self.now - self.duration:
            self.history.pop()
        if len(self.history) >= self.num_requests:
            return self.throttle_failure()
        return self.throttle_success()

    def throttle_success(self):
        """
        Inserts the current request's timestamp along with the key
        into the cache.
        """
        self.history.insert(0, self.now)
        self.cache.set(self.key, self.history, self.duration)
        return True

    def throttle_failure(self):
        """
        Called when a request to the API has failed due to throttling.
        """
        return False

    def wait(self):
        """
        Returns the recommended next request time in seconds.
        """
        if self.history:
            remaining_duration = self.duration - (self.now - self.history[-1])
        else:
            remaining_duration = self.duration
```

```
        available_requests = self.num_requests - len(self.history) + 1
        if available_requests <= 0:
            return None

        return remaining_duration / float(available_requests)


class AnonRateThrottle(SimpleRateThrottle):
    """
    Limits the rate of API calls that may be made by a anonymous users.

    The IP address of the request will be used as the unique cache key.
    """
    scope = 'anon'

    def get_cache_key(self, request, view):
        if request.user.is_authenticated:
            return None  # Only throttle unauthenticated requests.

        return self.cache_format % {
            'scope': self.scope,
            'ident': self.get_ident(request)
        }
```

总结

1. 如何实现的评率限制

```
- 匿名用户，用IP作为用户唯一标记，但如果用户换代理IP，无法做到真正的限制。
- 登录用户，用用户名或用户ID做标识。
具体实现：
    在django的缓存中 = {
        throttle_anon_1.1.1.1:[100121340,],
        1.1.1.2:[100121251,100120450,]
    }


    限制：60s能访问3次
    来访问时：
        1.获取当前时间 100121280
        2.100121280-60 = 100121220，小于100121220所有记录删除
        3.判断1分钟以内已经访问多少次了？ 4
        4.无法访问
    停一会
    来访问时：
        1.获取当前时间 100121340
        2.100121340-60 = 100121280,小于100121280所有记录删除
        3.判断1分钟以内已经访问多少次了？ 0
        4.可以访问
```

4.jwt

用于在前后端分离时，实现用户登录相关。

一般用户认证有2中方式：

- token

  用户登录成功之后，生成一个随机字符串，自己保留一分+给前端返回一份。

  以后前端再来发请求时，需要携带字符串。
  后端对字符串进行校验。

- jwt

  用户登录成功之后，生成一个随机字符串，给前端。
  - 生成随机字符串
    {typ:"jwt","alg":'HS256'} {id:1,username:'alx','exp':10}
    98qow39df0lj980945lkdjflo.saueoja8979284sdfsdf.asiuokjd978928374
    - 类型信息通过base64加密
    - 数据通过base64加密
    - 两个密文拼接在h256加密+加盐
  - 给前端返回
    98qow39df0lj980945lkdjflo.saueoja8979284sdfsdf.asiuokjd978928375

  前端获取随机字符串之后，保留起来。
  以后再来发送请求时，携带
  98qow39df0lj980945lkdjflo.saueoja8979284sdfsdf.asiuokjd978928375。

  后端接受到之后，
  1.先做时间判断
  2.字符串合法性校验。

安装

```
pip3 install djangorestframework-jwt
```

案例

- app中注册

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'api.apps.ApiConfig',
    'rest_framework',
    'rest_framework_jwt'
]
```

- 用户登录

```
import uuid
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.versioning import URLPathVersioning
from rest_framework import status

from api import models

class LoginView(APIView):
    """
    登录接口
    """
    def post(self,request,*args,**kwargs):

        # 基于jwt的认证
        # 1.去数据库获取用户信息
        from rest_framework_jwt.settings import api_settings
        jwt_payload_handler = api_settings.JWT_PAYLOAD_HANDLER
        jwt_encode_handler = api_settings.JWT_ENCODE_HANDLER

        user = models.UserInfo.objects.filter(**request.data).first()
        if not user:
            return Response({'code':1000,'error':'用户名或密码错误'})

        payload = jwt_payload_handler(user)
        token = jwt_encode_handler(payload)
        return Response({'code':1001,'data':token})
```

- 用户认证

```
from rest_framework.views import APIView
from rest_framework.response import Response

# from rest_framework.throttling import AnonRateThrottle,BaseThrottle


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

    def get(self,request,*args,**kwargs):
        # 获取用户提交的token，进行一步一步校验
        import jwt
        from rest_framework import exceptions
        from rest_framework_jwt.settings import api_settings
        jwt_decode_handler = api_settings.JWT_DECODE_HANDLER

        jwt_value = request.query_params.get('token')
        try:
            payload = jwt_decode_handler(jwt_value)
        except jwt.ExpiredSignature:
            msg = '签名已过期'
            raise exceptions.AuthenticationFailed(msg)
        except jwt.DecodeError:
            msg = '认证失败'
            raise exceptions.AuthenticationFailed(msg)
        except jwt.InvalidTokenError:
```

```
            raise exceptions.AuthenticationFailed()
        print(payload)

        return Response('文章列表')
```

呼啦圈作业

- 基于jwt认证的功能
- 文章列表+文章详细=访问频率限制（1/10次）