# 内容概要

- 文章的点赞点踩(重点)
- 文章的评论(重点)
  - 先只做根评论
  - 之后再做子评论
  - 小bug完善

## 文章详情页

```
# url设计
/username/article/1

# 先验证url是否会被其他url顶替

# 文章详情页和个人站点基本一致 所以用模版继承

# 侧边栏的渲染需要传输数据才能渲染 并且该侧边栏在很多页面都需要使用
 1.哪个地方用就拷贝需要的代码(不推荐 有点繁琐)

  2.将侧边栏制作成inclusion_tag
  """
 步骤
    1.在应用下创建一个名字必须叫templatetags文件夹
    2.在该文件夹内创建一个任意名称的py文件
    3.在该py文件内先固定写两行代码
        from django import template
        register = template.Library()
        # 自定义过滤器
        # 自定义标签
        # 自定义inclusion_tag

  """
  # 自定义inclusion_tag
@register.inclusion_tag('left_menu.html')
def left_menu(username):
    # 构造侧边栏需要的数据
    user_obj = models.UserInfo.objects.filter(username=username).first()
    blog = user_obj.blog
    # 1 查询当前用户所有的分类及分类下的文章数
    category_list =
models.Category.objects.filter(blog=blog).annotate(count_num=Count('article__pk')).values_list(
        'name', 'count_num', 'pk')
    # print(category_list)  # <QuerySet [('jason的分类一', 2), ('jason的分类二', 1), ('jason的分
类三', 1)]>

    # 2 查询当前用户所有的标签及标签下的文章数
    tag_list =
models.Tag.objects.filter(blog=blog).annotate(count_num=Count('article__pk')).values_list('name
',
```

```
            'count_num',

            'pk')
    # print(tag_list)  # <QuerySet [('tank的标签一', 1), ('tank的标签二', 1), ('tank的标签三',
2)]>

    # 3 按照年月统计所有的文章
    date_list =
models.Article.objects.filter(blog=blog).annotate(month=TruncMonth('create_time')).values(
        'month').annotate(count_num=Count('pk')).values_list('month', 'count_num')
    # print(date_list)
    return locals()
```

# 文章点赞点踩

```
"""
浏览器上你看到的花里胡哨的页面，内部都是HTML(前端)代码

那现在我们的文章内容应该写什么？？？    >>> html代码

如何拷贝文章
    copy outerhtml

1.拷贝文章
2.拷贝点赞点踩
    1.拷贝前端点赞点踩图标 只拷了html
    2.css也要拷贝
 由于有图片防盗链的问题 所以将图片直接下载到本地

课下思考:
 前端如何区分用户是点了赞还是点了踩
    1.给标签各自绑定一个事件
 两个标签对应的代码其实基本一样，仅仅是是否点赞点踩这一个参数不一样而已
    2.二合一
 给两个标签绑定一个事件
        //    给所有的action类绑定事件
    $('.action').click(function () {
            alert($(this).hasClass('diggit'))
        })

由于点赞点踩内部有一定的业务逻辑，所以后端单独开设视图函数处理

"""
# 个人建议:写代码先把所有正确的逻辑写完再去考虑错误的逻辑 不要试图两者兼得
import json
from django.db.models import F
def up_or_down(request):
    """
    1.校验用户是否登陆
    2.判断当前文章是否是当前用户自己写的(自己不能点自己的文章)
    3.当前用户是否已经给当前文章点过了
    4.操作数据库了
    :param request:
    :return:
    """
    if request.is_ajax():
```

```
        back_dic = {'code':1000,'msg':''}
        # 1 先判断当前用户是否登陆
        if request.user.is_authenticated():
            article_id = request.POST.get('article_id')
            is_up = request.POST.get('is_up')
            # print(is_up,type(is_up))  # true <class 'str'>
            is_up = json.loads(is_up)  # 记得转换
            # print(is_up, type(is_up))  # True <class 'bool'>
            # 2 判断当前文章是否是当前用户自己写的 根据文章id查询文章对象 根据文章对象查作者 根
request.user比对
            article_obj = models.Article.objects.filter(pk=article_id).first()
            if not article_obj.blog.userinfo == request.user:
                # 3 校验当前用户是否已经点了 哪个地方记录了用户到底点没点
                is_click =
models.UpAndDown.objects.filter(user=request.user,article=article_obj)
                if not is_click:
                    # 4 操作数据库 记录数据 要同步操作普通字段
                    # 判断当前用户点了赞还是踩 从而决定给哪个字段加一
                    if is_up:
                        # 给点赞数加一
                        models.Article.objects.filter(pk=article_id).update(up_num =
F('up_num') + 1)
                        back_dic['msg'] = '点赞成功'
                    else:
                        # 给点踩数加一

models.Article.objects.filter(pk=article_id).update(down_num=F('down_num') + 1)
                        back_dic['msg'] = '点踩成功'
                    # 操作点赞点踩表

models.UpAndDown.objects.create(user=request.user,article=article_obj,is_up=is_up)
                else:
                    back_dic['code'] = 1001
                    back_dic['msg'] = '你已经点过了,不能再点了'  # 这里你可以做的更加的详细 提示用
户到底点了赞还是点了踩
            else:
                back_dic['code'] = 1002
                back_dic['msg'] = '你个臭不要脸的!'
        else:
            back_dic['code'] = 1003
            back_dic['msg'] = '请先<a href="/login/">登陆</a>'
        return JsonResponse(back_dic)

<script>
        //   给所有的action类绑定事件
        $('.action').click(function () {
            {#alert($(this).hasClass('diggit'))#}
            let isUp = $(this).hasClass('diggit');
            let $div = $(this);
            // 朝后端发送ajax请求
            $.ajax({
                url:'/up_or_down/',
                type:'post',
                data:{
                    'article_id':'{{ article_obj.pk }}',
                    'is_up':isUp,
                    'csrfmiddlewaretoken':'{{ csrf_token }}'
                },
                success:function (args) {
                        if(args.code == 1000){
```

```
                            $('#digg_tips').text(args.msg)
                            // 将前端的数字加一
                            // 先获取到之前的数字
                            let oldNum = $div.children().text();  // 文本 是字符类型
                            // 易错点
                            $div.children().text(Number(oldNum) + 1)  // 字符串拼接了 1+1
= 11  11 + 1 = 111
                        }else{
                            $('#digg_tips').html(args.msg)
                        }
                    }
                })
            })
    </script>
```

# 文章评论

```
"""
我们先写根评论
再写子评论

点击评论按钮需要将评论框里面的内容清空

根评论有两步渲染方式
    1.DOM临时渲染
    2.页面刷新render渲染

子评论
 点击回复按钮发生了几件事
        1.评论框自动聚焦
        2.将回复按钮所在的那一行评论人的姓名
            @username
        3.评论框内部自动换行

根评论子评论都是点击一个按钮朝后端提交数据的
    parent_id
根评论子评论区别在哪?
    parent_id
"""
```