内容概要

bbs是一个前后端不分离的全栈项目，前端和后端都需要我们自己一步步的完成

- 表创建及同步
- 注册功能
  - forms组件
  - 用户头像前端实时展示
  - ajax
- 登陆功能
  - 自己实现图片验证码
  - ajax

今日内容详细

数据库表创建及同步

```
"""
由于django自带的sqlite数据库对日期不敏感，所以我们换成MySQL
"""
from django.db import models

# Create your models here.
"""
先写普通字段
之后再写外键字段
"""
from django.contrib.auth.models import AbstractUser


class UserInfo(AbstractUser):
    phone = models.BigIntegerField(verbose_name='手机号',null=True)
    # 头像
    avatar = models.FileField(upload_to='avatar/',default='avatar/default.png',verbose_name='用
户头像')
    """
 给avatar字段传文件对象 该文件会自动存储到avatar文件下 然后avatar字段只保存文件路径
avatar/default.png
    """
    create_time = models.DateField(auto_now_add=True)

    blog = models.OneToOneField(to='Blog',null=True)


class Blog(models.Model):
    site_name = models.CharField(verbose_name='站点名称',max_length=32)
    site_title = models.CharField(verbose_name='站点标题',max_length=32)
    # 简单模拟 带你认识样式内部原理的操作
```

```
    site_theme = models.CharField(verbose_name='站点样式',max_length=64)  # 存css/js的文件路径


class Category(models.Model):
    name = models.CharField(verbose_name='文章分类',max_length=32)
    blog = models.ForeignKey(to='Blog',null=True)


class Tag(models.Model):
    name = models.CharField(verbose_name='文章标签',max_length=32)
    blog = models.ForeignKey(to='Blog', null=True)


class Article(models.Model):
    title = models.CharField(verbose_name='文章标题',max_length=64)
    desc = models.CharField(verbose_name='文章简介',max_length=255)
    # 文章内容有很多 一般情况下都是使用TextField
    content = models.TextField(verbose_name='文章内容')
    create_time = models.DateField(auto_now_add=True)

    # 数据库字段设计优化
    up_num = models.BigIntegerField(verbose_name='点赞数',default=0)
    down_num = models.BigIntegerField(verbose_name='点踩数',default=0)
    comment_num = models.BigIntegerField(verbose_name='评论数',default=0)

    # 外键字段
    blog = models.ForeignKey(to='Blog', null=True)
    category = models.ForeignKey(to='Category',null=True)
    tags = models.ManyToManyField(to='Tag',
                                  through='Article2Tag',
                                  through_fields=('article','tag')
                                  )


class Article2Tag(models.Model):
    article = models.ForeignKey(to='Article')
    tag = models.ForeignKey(to='Tag')


class UpAndDown(models.Model):
    user = models.ForeignKey(to='UserInfo')
    article = models.ForeignKey(to='Article')
    is_up = models.BooleanField()  # 传布尔值 存0/1


class Comment(models.Model):
    user = models.ForeignKey(to='UserInfo')
    article = models.ForeignKey(to='Article')
    content = models.CharField(verbose_name='评论内容',max_length=255)
    comment_time = models.DateTimeField(verbose_name='评论时间',auto_now_add=True)
    # 自关联
    parent = models.ForeignKey(to='self',null=True)  # 有些评论就是根评论
```

注册功能

```
"""
我们之前是直接在views.py中书写的forms组件代码
但是为了接耦合 应该将所有的forms组件代码单独写到一个地方

如果你的项目至始至终只用到一个forms组件那么你可以直接建一个py文件书写即可
    myforms.py
但是如果你的项目需要使用多个forms组件，那么你可以创建一个文件夹在文件夹内根据
forms组件功能的不同创建不同的py文件
    myforms文件夹
        regform.py
        loginform.py
        userform.py
        orderform.py
        ...
"""
def register(request):
    form_obj = MyRegForm()
    if request.method == 'POST':
        back_dic = {"code": 1000, 'msg': ''}
        # 校验数据是否合法
        form_obj = MyRegForm(request.POST)
        # 判断数据是否合法
        if form_obj.is_valid():
            # print(form_obj.cleaned_data)  # {'username': 'jason', 'password': '123',
'confirm_password': '123', 'email': '123@qq.com'}
            clean_data = form_obj.cleaned_data  # 将校验通过的数据字典赋值给一个变量
            # 将字典里面的confirm_password键值对删除
            clean_data.pop('confirm_password')  # {'username': 'jason', 'password': '123',
'email': '123@qq.com'}
            # 用户头像
            file_obj = request.FILES.get('avatar')
            """针对用户头像一定要判断是否传值 不能直接添加到字典里面去"""
            if file_obj:
                clean_data['avatar'] = file_obj
            # 直接操作数据库保存数据
            models.UserInfo.objects.create_user(**clean_data)
            back_dic['url'] = '/login/'
        else:
            back_dic['code'] = 2000
            back_dic['msg'] = form_obj.errors
        return JsonResponse(back_dic)
    return render(request,'register.html',locals())


<script>
    $("#myfile").change(function () {
        // 文件阅读器对象
        // 1 先生成一个文件阅读器对象
        let myFileReaderObj = new FileReader();
        // 2 获取用户上传的头像文件
        let fileObj = $(this)[0].files[0];
        // 3 将文件对象交给阅读器对象读取
        myFileReaderObj.readAsDataURL(fileObj)  // 异步操作  IO操作
        // 4 利用文件阅读器将文件展示到前端页面 修改src属性
        // 等待文件阅读器加载完毕之后再执行
        myFileReaderObj.onload = function(){
             $('#myimg').attr('src',myFileReaderObj.result)
        }
```

```
    })

    $('#id_commit').click(function () {
        // 发送ajax请求 我们发送的数据中即包含普通的键值也包含文件
        let formDataObj = new FormData();
        // 1.添加普通的键值对
        {#console.log($('#myform').serializeArray())  // [{},{},{},{},{}]  只包含普通键值对#}
        $.each($('#myform').serializeArray(),function (index,obj) {
            {#console.log(index,obj)#}  // obj = {}
            formDataObj.append(obj.name,obj.value)
        });
        // 2.添加文件数据
        formDataObj.append('avatar',$('#myfile')[0].files[0]);

        // 3.发送ajax请求
        $.ajax({
            url:"",
            type:'post',
            data:formDataObj,

            // 需要指定两个关键性的参数
            contentType:false,
            processData:false,

            success:function (args) {
                if (args.code==1000){
                    // 跳转到登陆页面
                    window.location.href = args.url
                }else{
                    // 如何将对应的错误提示展示到对应的input框下面
                    // forms组件渲染的标签的id值都是 id_字段名
                    $.each(args.msg,function (index,obj) {
                         if (obj[0]=="两次密码不一致"){
                          $("#id_confirm_password").next().text("两次密码不一
致").parent().addClass("has-error")
                        }
                        {#console.log(index,obj)  //  username        ["用户名不能为空"]#}
                        let targetId = '#id_' + index;
                        $(targetId).next().text(obj[0]).parent().addClass('has-error')
                    })
                }
            }
        })
    })
    // 给所有的input框绑定获取焦点事件
    $('input').focus(function () {
        // 将input下面的span标签和input外面的div标签修改内容及属性
        $(this).next().text('').parent().removeClass('has-error')
    })
</script>

# 扩展
"""
一般情况下我们在存储用户文件的时候为了避免文件名冲突的情况
会自己给文件名加一个前缀
    uuid
 随机字符串
    ...
"""
```

登陆功能1

```
"""
img标签的src属性
    1.图片路径
    2.url
    3.图片的二进制数据

我们的计算机上面致所有能够输出各式各样的字体样式
内部其实对应的是一个个.ttf结尾的文件

http://www.zhaozi.cn/ai/2019/fontlist.php?
ph=1&classid=32&softsq=%E5%85%8D%E8%B4%B9%E5%95%86%E7%94%A8
"""



"""
图片相关的模块
    pip3 install pillow
"""
from PIL import Image,ImageDraw,ImageFont
"""
Image:生成图片
ImageDraw:能够在图片上乱涂乱画
ImageFont:控制字体样式
"""
from io import BytesIO,StringIO
"""
内存管理器模块
BytesIO:临时帮你存储数据 返回的时候数据是二进制
StringIO:临时帮你存储数据 返回的时候数据是字符串
"""
import random
def get_random():
    return random.randint(0,255),random.randint(0,255),random.randint(0,255)
def get_code(request):
    # 推导步骤1:直接获取后端现成的图片二进制数据发送给前端
    # with open(r'static/img/111.jpg','rb') as f:
    #     data = f.read()
    # return HttpResponse(data)

    # 推导步骤2:利用pillow模块动态产生图片
    # img_obj = Image.new('RGB',(430,35),'green')
    # img_obj = Image.new('RGB',(430,35),get_random())
    # # 先将图片对象保存起来
    # with open('xxx.png','wb') as f:
    #     img_obj.save(f,'png')
    # # 再将图片对象读取出来
    # with open('xxx.png','rb') as f:
    #     data = f.read()
    # return HttpResponse(data)

    # 推导步骤3:文件存储繁琐IO操作效率低 借助于内存管理器模块
    # img_obj = Image.new('RGB', (430, 35), get_random())
    # io_obj = BytesIO()  # 生成一个内存管理器对象 你可以看成是文件句柄
    # img_obj.save(io_obj,'png')
```

```
    # return HttpResponse(io_obj.getvalue())  # 从内存管理器中读取二进制的图片数据返回给前端


    # 最终步骤4:写图片验证码
    img_obj = Image.new('RGB', (430, 35), get_random())
    img_draw = ImageDraw.Draw(img_obj)  # 产生一个画笔对象
    img_font = ImageFont.truetype('static/font/222.ttf',30)  # 字体样式 大小

    # 随机验证码 五位数的随机验证码 数字 小写字母 大写字母
    code = ''
    for i in range(5):
        random_upper = chr(random.randint(65,90))
        random_lower = chr(random.randint(97,122))
        random_int = str(random.randint(0,9))
        # 从上面三个里面随机选择一个
        tmp = random.choice([random_lower,random_upper,random_int])
        # 将产生的随机字符串写入到图片上
        """
为什么一个个写而不是生成好了之后再写
因为一个个写能够控制每个字体的间隙 而生成好之后再写的话
间隙就没法控制了
        """
        img_draw.text((i*60+60,-2),tmp,get_random(),img_font)
        # 拼接随机字符串
        code += tmp
    print(code)
    # 随机验证码在登陆的视图函数里面需要用到 要比对 所以要找地方存起来并且其他视图函数也能拿到
    request.session['code'] = code
    io_obj = BytesIO()
    img_obj.save(io_obj,'png')
    return HttpResponse(io_obj.getvalue())
```

```
  <script>
    $("#id_img").click(function () {
        // 1 先获取标签之前的src
        let oldVal = $(this).attr('src');
        $(this).attr('src',oldVal += '?')
    })
</script>
```

登陆功能2

```
def login(request):
    if request.method == 'POST':
        back_dic = {'code':1000,'msg':''}
        username = request.POST.get('username')
        password = request.POST.get('password')
        code = request.POST.get('code')
        # 1 先校验验证码是否正确 自己决定是否忽略 统一转大写或者小写再比较
        if request.session.get('code').upper() == code.upper():
            # 2 校验用户名和密码是否正确
            user_obj = auth.authenticate(request,username=username,password=password)
```

```
            if user_obj:
                # 保存用户状态
                auth.login(request,user_obj)
                back_dic['url'] = '/home/'
            else:
                back_dic['code'] = 2000
                back_dic['msg'] = '用户名或密码错误'
        else:
            back_dic['code'] = 3000
            back_dic['msg'] = '验证码错误'
        return JsonResponse(back_dic)
    return render(request,'login.html')
```