

博文相关接口

功能分析

功能	函数名	Request方法	路径
发布 (增)	pub	POST	/pub
看文章 (查)	get	GET	/(\d+)
列表(分页)	getall	GET	/

创建博文应用

```
$ python manage.py startapp post
```

注意：一定要把应用post加入到settings.py中，否则不能迁移

模型

```
from django.db import models
from user.models import User

class Post(models.Model):
    class Meta:
        db_table = 'post'
    id = models.AutoField(primary_key=True)
    title = models.CharField(max_length=256, null=False)
    postdate = models.DateTimeField(null=False)
    # 从post查作者，从post查内容
    author = models.ForeignKey(User) # 指定外键，migrate会生成author_id字段
    # self.content可以访问Content实例，其内容是self.content.content

    def __repr__(self):
        return '<Post {} {} {} {}>'.format(
            self.id, self.title, self.author_id, self.content)

    __str__ = __repr__

class Content(models.Model):
    class Meta:
        db_table = 'content'
    # 没有主键，会自动创建一个自增主键
    post = models.OneToOneField(Post) # 一对一，这边会有一个外键post_id引用post.id
    content = models.TextField(null=False)

    def __repr__(self):
```

```
        return '<Content {} {}>'.format(self.post.id, self.content[:20])

    __str__ = __repr__
```

路由

全局settings.py

```
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', index),
    url(r'^index/$', index),
    url(r'^user/', include('user.urls')),
    url(r'^post/', include('post.urls')),
]
```

post应用urls.py

```
from django.conf.urls import url
from .views import pub, get, getall

urlpatterns = [
    url(r'^pub$', pub),
    url(r'^(\d+)$', get), # 给get传入一个参数str类型
    url(r'^$', getall),
]
```

pub接口实现

用户从浏览器端提交json数据，包含title， content。
提交博文需要认证用户，从请求的header中验证jwt。

```
request: POST-> @authenticate pub -> return post_id
```

```
from django.http import HttpResponse, HttpRequest, JsonResponse, HttpResponseBadRequest,
HttpResponseNotFound
from user.views import authenticate
from user.models import User
import simplejson
import datetime
from .models import Post, Content

@authenticate
def pub(request:HttpRequest):
    post = Post() # 新增
    content = Content() # 新增
    try:
        payload = simplejson.loads(request.body)
        post.title = payload['title']
```

```

post.author = User(id=request.user.id) # 注入的
#post.author = request.user
post.postdate = datetime.datetime.now(
    datetime.timezone(datetime.timedelta(hours=8)))

post.save()

content.content = payload['content']
content.post = post
content.save()

return JsonResponse({'post_id':post.id})
except Exception as e:
    print(e)
    return HttpResponseBadRequest()

```

显式事务处理

Django中每一次save()调用就会自动提交，那么上例中第一次事务提交后如果第二次提交前出现异常，则post.save()不会回滚。

解决办法可以使用事务的原子方法，参考<https://docs.djangoproject.com/zh-hans/2.1/topics/db/transactions/#controlling-transactions-explicitly>

```

from django.db import transaction

@transaction.atomic # 装饰器用法
def viewfunc(request):
    # This code executes inside a transaction.
    do_stuff()

```

```

from django.db import transaction

def viewfunc(request):
    # This code executes in autocommit mode (Django's default).
    do_stuff()

    with transaction.atomic(): # 上下文用法
        # This code executes inside a transaction.
        do_more_stuff()

```

上面2种用法都可以，我们这一次采用第二种方法。

```

@authenticate
def pub(request:HttpRequest):
    post = Post() # 新增
    content = Content() # 新增
    try:
        payload = simplejson.loads(request.body)

```

```

post.title = payload['title']
post.author = User(id=request.user.id) # 注入的
#post.author = request.user
post.postdate = datetime.datetime.now(
    datetime.timezone(datetime.timedelta(hours=8)))

with transaction.atomic(): # 原子操作
    post.save() # save方法必须在前, save后才有post.id

    content.content = payload['content']
    content.post = post
    content.save()

    return JsonResponse({'post_id':post.id})
except Exception as e:
    print(e)
    return HttpResponseBadRequest()

```

get接口实现

根据post_id查询博文并返回。

这里需要认证吗？

如果博文只能作者看，就需要认证。我们这里的公开给所有人看，所以不需要认证，同样，下面的list接口也是不需要认证的。

```
request: GET-> get Post by id -> return post+content
```

```

def get(request:HttpRequest, id): # 分组捕获传入
    try:
        id = int(id)
        post = Post.objects.get(pk=id)
        print(post, '~~~~~')
        if post:
            return JsonResponse({
                'post':{
                    'post_id':post.id,
                    'title':post.title,
                    'author':post.author.name,
                    'author_id':post.author_id, # post.author.id
                    'postdate':post.postdate.timestamp(),
                    'content':post.content.content
                }
            })
        # get方法保证必须只有一条记录, 否则抛异常
    except Exception as e:
        print(e)
        return HttpResponseNotFound()

```

getall接口实现

发起get请求，通过查询字符串 `http://url/post/?page=2` 查询第二页数据

```
request: GET-> get all (page=1) -> return post list
```

```
def getall(request:HttpRequest):
    try: # 页码
        page = int(request.GET.get('page', 1))
        page = page if page > 0 else 1
    except:
        page = 1

    try: # 页码行数
        # 注意，这个数据不要轻易让浏览器端改变，如果允许改变，一定要控制范围
        size = int(request.GET.get('size', 20))
        size = size if size > 0 and size < 101 else 20
    except:
        size = 20

    try:
        # 按照id倒排
        start = (page - 1) * size
        posts = Post.objects.order_by('-id')[start:start+size]
        #posts = Post.objects.order_by('-pk')[start:start+size]
        print(posts.query)
        return JsonResponse({
            'posts':[
                {
                    'post_id': post.id,
                    'title': post.title
                } for post in posts
            ]
        })
    except Exception as e:
        print(e)
        return HttpResponseBadRequest()
```

完善分页

分页信息，一般有：当前页/总页数、行限制数、记录总数。

当前页：page

行限制数：size，每页最多多少行

总页数：pages = math.ceil(count/size)

记录总数：count，从select * from table来

```
def getall(request: HttpRequest):
    try: # 页码
        page = int(request.GET.get('page', 1))
        page = page if page > 0 else 1
```

```

except:
    page = 1

try: # 页码行数
    # 注意, 这个数据不要轻易让浏览器端改变, 如果允许改变, 一定要控制范围
    size = int(request.GET.get('size', 20))
    size = size if size > 0 and size < 101 else 20
except:
    size = 20

try:
    # 按照id倒排
    start = (page - 1) * size
    posts = Post.objects.order_by('-id')
    print(posts.query)
    count = posts.count()

    posts = posts[start:start + size]
    print(posts.query)

    return JsonResponse({
        'posts': [
            {
                'post_id': post.id,
                'title': post.title
            } for post in posts
        ], 'pagination': {
            'page': page,
            'size': size,
            'count': count,
            'pages': math.ceil(count / size)
        }
    })

except Exception as e:
    print(e)
    return HttpResponseBadRequest()

```

也可以使用Django提供的Paginator类来完成。

Paginator文档 <https://docs.djangoproject.com/en/1.11/topics/pagination/>。

但是, 还是自己处理更加简单明了些。

改写校验函数

```

def validate(d:dict, name:str, type_func, default, validate_func):
    try: # 页码
        result = type_func(d.get(name, default))
        result = validate_func(result, default)
    except:
        result = default
    return result

```

```
def getall(request: HttpRequest):
    # 页码
    page = validate(request.GET, 'page', int, 1, lambda x,y: x if x>0 else y)
    # 注意, 这个数据不要轻易让浏览器端改变, 如果允许改变, 一定要控制范围
    size = validate(request.GET, 'size', int, 20, lambda x,y: x if x>0 and x<101 else y)

    try:
        # 按照id倒排
        start = (page - 1) * size
        posts = Post.objects.order_by('-id')
        print(posts.query)
        count = posts.count()

        posts = posts[start:start + size]
        print(posts.query)

        return JsonResponse({
            'posts': [
                {
                    'post_id': post.id,
                    'title': post.title
                } for post in posts
            ], 'pagination': {
                'page': page,
                'size': size,
                'count': count,
                'pages': math.ceil(count / size)
            }
        })
    except Exception as e:
        print(e)
        return HttpResponseBadRequest()
```