用户功能设计与实现

提供用户注册处理
提供用户登录处理
提供路由配置

用户登录接口设计

接收用户通过POST方法提交的登录信息，提交的数据是JSON格式数据

```
{
  "password":"abc",
  "email":"wayne@magedu.com"
}
```

从user表中email找出匹配的一条记录，验证密码是否正确。
验证通过说明是合法用户登录，显示欢迎页面。
验证失败返回错误状态码，例如4xx
整个过程都采用AJAX异步过程，用户提交JSON数据，服务端获取数据后处理，返回JSON。

URL：/user/login
METHOD：POST

路由配置

```
from django.conf.urls import url
from .views import reg, login

urlpatterns = [
    url(r'^reg$', reg),
    url(r'^login$', login),
]
```

登录代码

```
    try:
        payload = simplejson.loads(request.body) # 获取登录信息数据
        print(payload)
        email = payload['email']

        user = User.objects.get(email=email) # only one
        if bcrypt.checkpw(payload['password'].encode(), user.password.encode()):
            # 验证成功
            token = gen_token(user.id)
            res =  JsonResponse({
                'user':{
                    'user_id':user.id,
```

```
                    'name':user.name,
                    'email':user.email
                }, 'token':token
            })
            res.set_cookie('jwt', token) # 成功登陆，response报文设置set-cookie
            return res
        else:
            return HttpResponseBadRequest()
    except Exception as e:
        print(e)
        return HttpResponseBadRequest()
```

认证接口

如何获取浏览器提交的token信息?

1、使用Header中的Authorization
通过这个header增加token信息。
通过header发送数据，方法可以是Post、Get

2、自定义header
在Http请求头中使用JWT字段来发送token
我们选择第二种方式

认证
基本上所有的业务都需要认证用户的信息。
在这里比较时间戳，如果过期，就直接抛未认证401，客户端收到后就该直接跳转到登录页。
如果没有提交user id，就直接重新登录。如果用户查到了，填充user对象。

request -> 时间戳比较 -> user id 比较 -> 向后执行

Django的认证

django.contrib.auth中提供了许多方法，这里主要介绍其中的三个:

1、authenticate(**credentials)
提供了用户认证，即验证用户名以及密码是否正确
user = authentica(username='someone',password='somepassword')

2、login(HttpRequest, user, backend=None)
该函数接受一个HttpRequest对象，以及一个认证了的User对象
此函数使用django的session框架给某个已认证的用户附加上session id等信息。

3、logout(request)
注销用户
该函数接受一个HttpRequest对象，无返回值。
当调用该函数时，当前请求的session信息会全部清除
该用户即使没有登录，使用该函数也不会报错

还提供了一个装饰器来判断是否登录django.contrib.auth.decorators.login_required

本项目使用了无session机制，且用户信息自己建表管理，所以，认证需要自己实现。

中间件技术Middleware

官方定义，在Django的request和response处理过程中，由框架提供的hook钩子

中间件技术在1.10后发生了改变，我们当前使用1.11版本，可以使用新的方式定义。
参看 https://docs.djangoproject.com/en/1.11/topics/http/middleware/#writing-your-own-middleware

```
#
class BlogAuthMiddleware(object):
    '''自定义认证中间件'''
    def __init__(self, get_response):
        self.get_response = get_response
        # 初始化执行一次

    def __call__(self, request):
        # 视图函数之前执行
        # 认证
        print(type(request), '++++++++++++++++++')
        print(request.GET)
        print(request.POST)
        print(request.body) # json数据
        print('-'*30)

        response = self.get_response(request)

        # 视图函数之后执行

        return response
# 要在setting的MIDDLEWARE中注册
```

但是，这样所有的请求和响应都拦截，我们还得判断是不是访问的想要拦截的view函数，所以，考虑其他方法。

中间件有很多用途，适合拦截所有请求和响应。例如浏览器端的IP是否禁用、UserAgent分析、异常响应的统一处理。

思考：使用中间件实现访问IP统计

装饰器*

在需要认证的view函数上增强认证功能，写一个装饰器函数。谁需要认证，就在这个view函数上应用这个装饰器。

```
AUTH_EXPIRE = 8 * 60 * 60 # 8小时过期

def authenticate(view):
    def wrapper(request:HttpRequest):
        # 自定义header jwt
        payload = request.META.get('HTTP_JWT') # 会被加前缀HTTP_且全大写
```

```
        if not payload: # None没有拿到，认证失败
            return HttpResponse(status=401)
        try: # 解码
            payload = jwt.decode(payload, settings.SECRET_KEY, algorithms=['HS256'])
            print(payload)
        except:
            return HttpResponse(status=401)
        # 验证过期时间
        current = datetime.datetime.now().timestamp()
        if (current - payload.get('timestamp', 0)) > AUTH_EXPIRE:
            return HttpResponse(status=401)
        print('-'*30)
        try:
            user_id = payload.get('user_id', -1)
            user = User.objects.get(pk=user_id)
            request.user = user # 如果正确，则注入user
            print('-'*30)
        except Exception as e:
            print(e)
            return HttpResponse(status=401)

        ret = view(request) # 调用视图函数
        # 特别注意view调用的时候，里面也有返回异常
        return ret
    return wrapper


@authenticate # 很自由的应用在需要认证的view函数上
def test(request:HttpRequest):
    return HttpResponse('test')
```

Jwt过期问题

pyjwt支持过期设定，在decode的时候，如果过期，则抛出异常。
需要在payload中增加claim exp。
exp要求是一个整数int的时间戳。

```
import jwt
import datetime
import threading

event = threading.Event()

key = 'magedu'

# 在jwt的payload中增加exp claim
data = jwt.encode({'name':'tom', 'age':20, 'exp': int(datetime.datetime.now().timestamp())+10},
key)
print(jwt.get_unverified_header(data)) # 不校验签名提取header
try:
    while not event.wait(1):
        print(jwt.decode(data, key)) # 过期，校验就会抛出异常
```

```
        print(datetime.datetime.now().timestamp())
except jwt.ExpiredSignatureError as e:
    print(e)
```

重写Jwt过期

```
AUTH_EXPIRE = 8 * 60 * 60 # 8小时过期

def gen_token(user_id):
    '''生成token'''
    return jwt.encode({ # 增加时间戳，判断是否重发token或重新登录
        'user_id': user_id,
        'exp': int(datetime.datetime.now().timestamp()) + AUTH_EXPIRE # 要取整
    } , settings.SECRET_KEY, 'HS256').decode() # 字符串


def authenticate(view):
    def wrapper(request:HttpRequest):
        # 自定义header jwt
        payload = request.META.get('HTTP_JWT') # 会被加前缀HTTP_且全大写
        if not payload: # None没有拿到，认证失败
            return HttpResponse(status=401)
        try: # 解码，同时验证过期时间
            payload = jwt.decode(payload, settings.SECRET_KEY, algorithms=['HS256'])
            print(payload)
        except:
            return HttpResponse(status=401)

        try:
            user_id = payload.get('user_id', -1)
            user = User.objects.get(pk=user_id)
            request.user = user # 如果正确，则注入user
            print('-'*30)
        except Exception as e:
            print(e)
            return HttpResponse(status=401)

        ret = view(request) # 调用视图函数
        return ret
    return wrapper
```