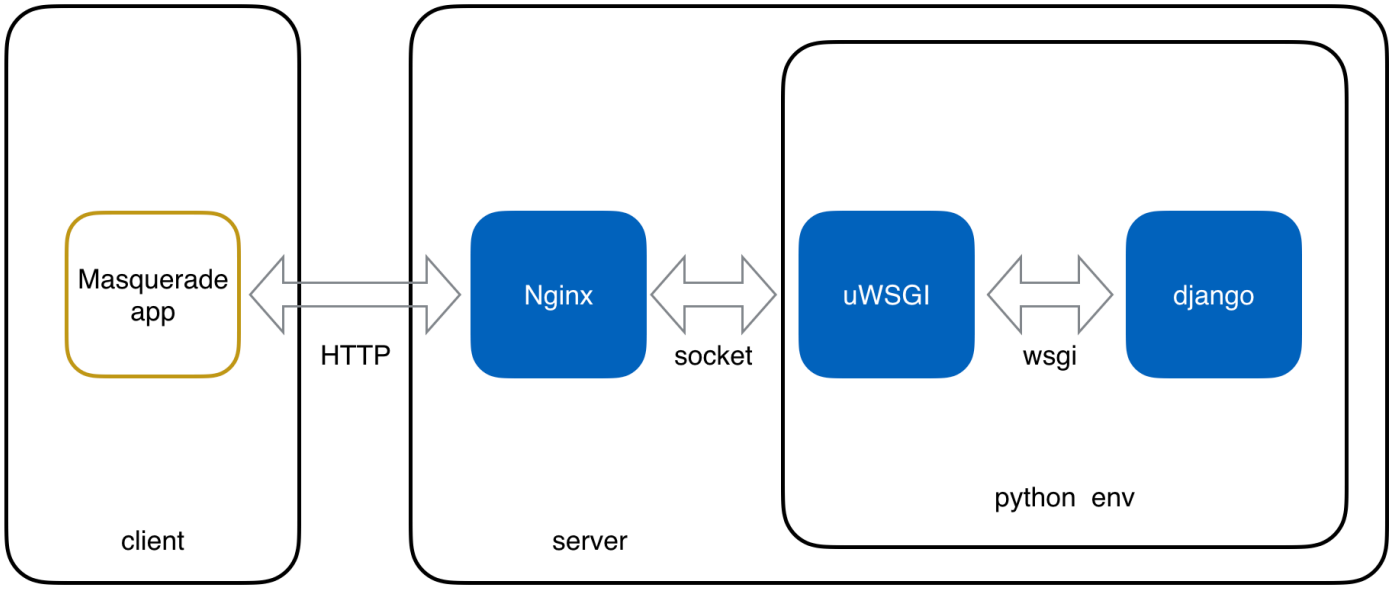
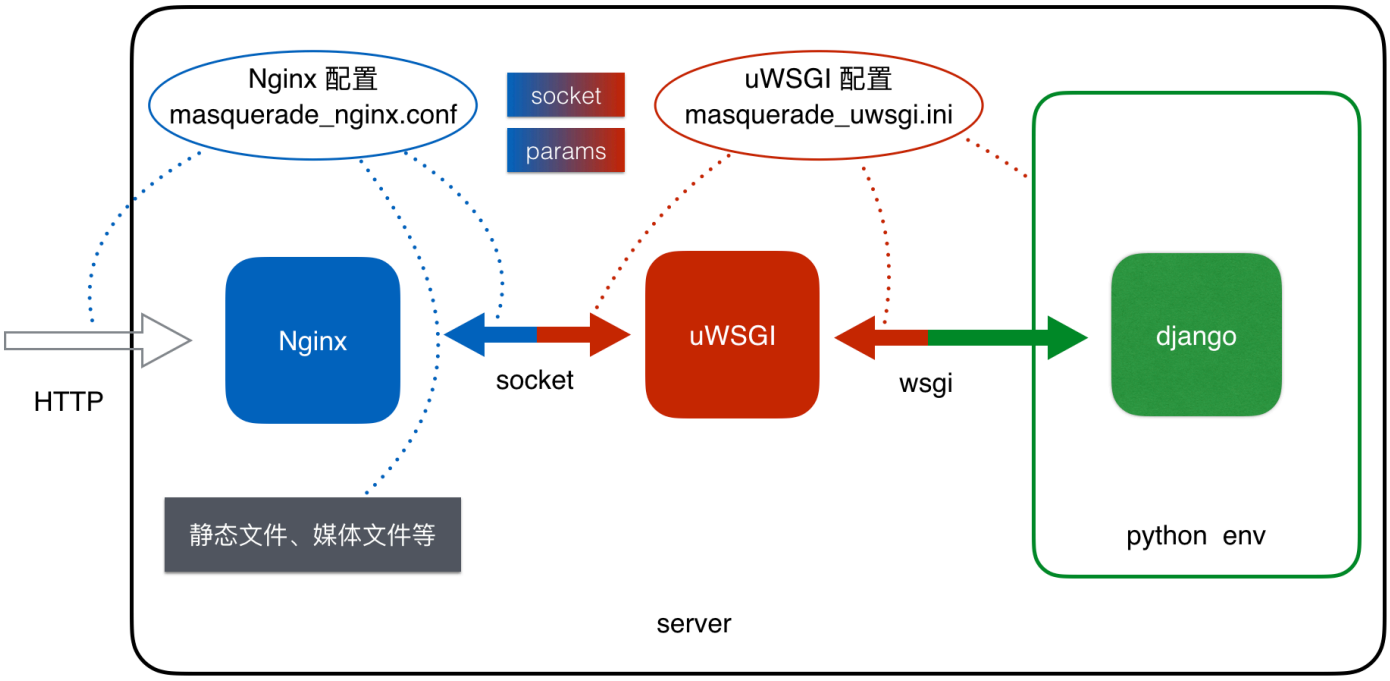
Masquerade v0.0.1

工作原理



当客户端上的 `Masquerade` 运行获取数据时，实际上是客户端和服务端的 `web` 服务产生了一个 `HTTP` 协议的通信，客户端就会看到返回的数据。 `django` 是运行在python环境中的，而且我们把它封装在了一个 `python` 的虚拟环境中，为达到和其它模块不产生冲突。 `uWSGI` 模块使用 `wsgi` 与 `django` 进行连接通信，使用 `socket` 套接字和 `Nginx` 进行通信。

运行过程



当 `HTTP` 协议来了，先到达 `Nginx` ， `Nginx` 通过它的配置使用 `socket` 连接到 `uWSGI` ， `uWSGI` 通过配置连接到了 `python` 的运行环境， `uWSGI` 要和 `Nginx` 进行通信需要给一个套接字位置和参数表。当然，严格来说应该是 `python` 的运行环境和真实环境都需要安装 `uWSGI` ，在此省略。

项目地址

请向 `Masquerade` 后端同学索取

部署

测试环境项目结构（已省略不必要的文件）

```
.
├── env # 虚拟环境
|   |
├── masquerade              # Masquerade project
|   ├── blog # 文章 app
|   |   ├── models.py       # 文章 模型
|   |   ├── urls.py         # 文章 路由
|   |   └── views.py        # 文章 路由对应的响应方法
|   ├── comment             # 评论 app
|   |   ├── urls.py         # 评论 路由
|   |   └── views.py        # 评论 路由对应的响应方法
|   ├── common              # 通用文件
|   |   ├── decorator.py    # 装饰器
|   |   ├── masLogger.py    # log
|   |   ├── middleware.py   # 中间件
|   |   ├── token_utils.py  # token校验
|   |   └── utils.py        # 通用方法
|   ├── like_statistics     # 点赞 app
|   |   ├── models.py       # 点赞 模型
|   |   ├── urls.py         # 点赞 路由
|   |   └── views.py        # 点赞 路由对应的响应方法
|   ├── log                 # 日志
|   |   ├── info.log        # INFO、WARNING级别日志
|   |   └── log.log         # ERROE级别日志
|   ├── manage.py           # 工程 管理文件
|   ├── masquerade          # Masquerade 工程
|   |   ├── settings.py     # Masquerade 设置文件
|   |   ├── urls.py         # Masquerade 总路由
|   |   └── wsgi.py         # python服务器网管接口
|   ├── media               # 媒体文件（主要是图片）
|   |   └── avatar          # 用户头像文件夹
|   ├── read_statistics     # 阅读数 app
|   |   ├── models.py       # 阅读数 模型
|   |   └── views.py        # 阅读数 处理方法
|   ├── user                # 用户 app
|   |   ├── models.py       # 用户 模型
|   |   ├── urls.py         # 用户 路由
|   |   └── views.py        # 用户 路由对应的响应方法
|   └── user_avatar         # 用户头像 app
|       ├── models.py       # 用户头像 模型
|       ├── urls.py         # 用户头像 路由
|       └── views.py        # 用户头像 路由对应的响应方法
└── requirement.txt         # 虚拟环境依赖包
```

基本环境

系统：`Ubuntu 16.04 LTS`

硬件：无

软件：

Package	Version
python	3.6.5
Django	2.0.7
mysqlclient	1.3.13
Pillow	5.2.0
pip	18.0
pytz	2018.5
setuptools	40.0.0
wheel	0.31.1

相关命令

因 `Ubuntu 16.04 LTS` 自带 `python` 版本为 `2.7` 和 `3.5` ，不太符合预先要求，需要安装python3.6。

```
    # 安装 python 3.6
    apt-get update
    apt-get install software-properties-common
    add-apt-repository ppa:jonathonf/python-3.6
    apt-get update
    apt-get install python3.6
```

```
    # 创建软链接
    cd /usr/bin
    rm python
    ln -s python3.6 python
    rm python3
    ln -s python3.6 python3
```

```
# 安装 pip3.6
apt-get install python3-pip
pip3 install --upgrade pip
```

```
# 安装虚拟环境
pip3 install virtualenv
```

```
# 创建虚拟环境
virtualenv env
source env/bin/activate
deactivate
```

```
# 安装git
apt-get install git
```

```
# clone代码
git clone  Masquerade项目git地址
```

```
# 安装mysql
wget https://dev.mysql.com/get/mysql-apt-config_0.8.10-1_all.deb
dpkg -i mysql-apt-config_0.8.10-1_all.deb
apt-get update
apt-get install mysql-server
```

```
# 安装mysqlclient
wget https://dev.mysql.com/get/libmysqlclient-dev_8.0.11-1ubuntu16.04_amd64
dpkg -i libmysqlclient-dev_8.0.11-1ubuntu16.04_amd64.deb
# 若提示找不到 libmysqlclient21, `apt-get install libmysqlclient21`
apt-get update
apt-get install libmysqlclient-dev
apt-get install python3.6-dev
apt-get install openssl
apt-get install libssl-dev

# 若安装过程中出现`OSError: mysql_config not found`。可以做个软链接
$ ln -s /usr/local/mysql/bin/mysql_config /usr/local/bin/mysql_config
```

```
    # 创建数据库（用 root 账户进入到 `mysql shell`中）
    $ create database mas_db default charset=utf8mb4 default collate utf8mb4_un
    # 创建用户
    $ create user 'masquerade'@'localhost' identified by 'masquerade_2018';
    # 对用户赋值权限
    $ grant all privileges on mas_db.* to 'masquerade’@'localhost';
    # 刷新
    $ flush privileges;
```

```
    # 创建 cache 表
    $ python manage.py createcachetable
```

```
    # 获取虚拟环境依赖包
    python manage.py requirement.txt
```

```
    # 同步数据库表结构信息
    $ python manage.py makemigrations
    $ python managa.py migrate
```

```
    # 启动测试
    python manage.py runserver 0.0.0.0:80
    # 若在非本地环境进行测试，记得先在工程 `setting.py` 文件中的 `ALLOW_HOSTS` 添加上对
```

线上环境

注：要求先部署完**测试环境**，且工程文件夹位于当前用户目录下。

增加软件： `Nginx` 服务器、 `uWSGI` 应用协议服务模块

```
......
|
├── masquerade_nginx.conf
|   |
├── masquerade_uwsgi.ini
|   |
├── uwsgi_params
|   |
└── requirement.txt         # 虚拟环境依赖包
```

相关命令

```
# 更新 apt 源
$ apt update
```

```
# 安装 nginx
$ apt install nginx
# 安装成功后，nginx 默认启动且占用80端口，浏览器直接访问服务器或域名，出现 nginx 启动成
# 测试 nginx 和 python 运行环境是否有冲突
$ python manage.py runserver 0.0.0.0:8000
```

```
# 进入到虚拟环境中
$ pip install uwsgi
# 使用 uwsgi 虚拟一个 web 测试服务器。masquerade project 文件中有个一个wsgi.py文件
$ uwsgi --http :8000 --module masquerade.wsgi
```

```
    # 新建 nginx 的配置文件
    # masquerade_nginx.conf

    # the upstream component nginx needs to connect to
    upstream django {
        # server unix:///path/to/your/mysite/mysite.sock; # for a file socket
        server 127.0.0.1:8001; # for a web port socket (we'll use this first)
    }

    # configuration of the server
    server {
        # 修改成想要监听的端口
        listen      80;
        # 修改服务器名称
        server_name .pjhubs.com;
        charset     utf-8;

        # max upload size
        client_max_body_size 75M;   # adjust to taste

        # Django media
        location /media  {
            # 修改媒体文件路径
            alias /home/ubuntu/socialNetwork-backend/masquerade/media;  # your
        }

        # Finally, send all non-media requests to the Django server.
        location / {
            uwsgi_pass  django;
            # 修改参数表位置
            include     /home/ubuntu/socialNetwork-backend/uwsgi_params; # the
        }
}
```

```
# uwsgi_param 文件内容，都是一些 nginx 和 uWSGI 互相映射的参数，几乎没变过，不需要改
uwsgi_param  QUERY_STRING       $query_string;
uwsgi_param  REQUEST_METHOD     $request_method;
uwsgi_param  CONTENT_TYPE       $content_type;
uwsgi_param  CONTENT_LENGTH     $content_length;

uwsgi_param  REQUEST_URI        $request_uri;
uwsgi_param  PATH_INFO          $document_uri;
uwsgi_param  DOCUMENT_ROOT      $document_root;
uwsgi_param  SERVER_PROTOCOL    $server_protocol;
uwsgi_param  REQUEST_SCHEME     $scheme;
uwsgi_param  HTTPS              $https if_not_empty;

uwsgi_param  REMOTE_ADDR        $remote_addr;
uwsgi_param  REMOTE_PORT        $remote_port;
uwsgi_param  SERVER_PORT        $server_port;
uwsgi_param  SERVER_NAME        $server_name;
```

```
# masquerade_uwsgi.ini 文件
[uwsgi]
# the base directory (full path)
chdir           = /home/ubuntu/socialNetwork-backend/masquerade
# Django's wsgi file
module          = masquerade.wsgi:application
# the virtualenv (full path)
home            = /home/ubuntu/socialNetwork-backend/env
# process-related settings
# master
master          = true
# maximum number of worker processes
processes       = 10
# the socket (use the full path to be safe
socket          = :8001
# ... with appropriate permissions - may be needed
# chmod-socket    = 664
# clear environment on exit
vacuum          = true
```

```
    # 把 nginx 默认的配置文件删掉，改为刚才设置的 .conf 文件
    $ cd /etc/nginx/sites-enabled/
    $ rm default
    # 软连接过去到该目录中
    $ ln -s /home/ubuntu/socialNetwork-backend/masquerade_nginx.conf mas_nginx.
    # 因为修改了配置文件，重启 nginx
    $ /etc/init.d/nginx restart
    # 回去
    $ cd
    # 测试一下
    $ uwsgi --ini uwsgi.ini
```

nginx 能够每次重启服务器时都自动重启，但是 `uWSGI` 不行，没找到一个合适的方法，只能先把启动 `uWSGI` 服务的命令放到服务器启动文件中。

```
    # 真实 python 环境也要 uWSGI，退出虚拟环境
    $ pip install uwsgi
```

```
    $ vi /etc/rc.local
    # 把 `/usr/local/bin/uwsgi --ini /home/ubuntu/socialNetwork-backend/masquer
```

```
    # 最后重启服务器即可
    $ reboot
```

思考

这部分内容可以不用关注，主要是在开发的过程中记录一些思考，为什么要这么设计以及可一些优化的地方，供后来接手的同学参考。

用户

用户这块最开始是直接继承了 `django` 的用户模块，这个模块有个利弊分明的地方，如果我们是做 `web` 开发，用 `django` 提供的用户模块是最方便不过了，但如果只是用 `django` 做 `API` 服务，太重，冗余字段和比较“僵硬”的约束，比如没法得知 `django` 的密码加密方式，前后端没法一致，客户端是没有所谓的 `cookie` 和 `session` 管理（web壳子除外），如果非得要硬上也不是不行，但是这要付出很多额外的时间，没法做到 `django` 天然支持的 `web` 开发之便利,还有其它一些觉得用起来比较棘手的地方。当然，这些约束仁者见仁智者见智。最终的做法是自建了一个 `MasUser` 模型，需要提前考虑后续开发要接入的其它用户权限认证方式，比如QQ、微信、微博等的第三方登录权限 `token` 。

在用户的权限认证这一块，因为没有采用 `django` 自带的用户模型，也就导致了没法用其提供的默认权限认证，遂直接全部抛弃，直接用了最粗暴的方法做了权限认证。自定义了一套 `token` 生成规则，当然，

从本质上说还是基于 `django-cache` 的二次封装，我所做的只是重新制定了一套 `token` 生成的规则，其余的一概未动。`token` 存储是基于 `mysql` 做的数据库缓存，我知道数据库缓存实在是不好，在性能问题上得不到很好的解决，但这就算是一个验证吧，看看数据库缓存能有多大的问题。如果真的压不住了或者其它情况，推荐直接上 `Redis` 。

评论

评论在 `Masquerade` 中从本质上来看其实也是文章的一个变种，只不过定义不同，因此它们在底层字段都是一样的 `contentType` ， 评论这块最主要的问题就是树结构，当时设计这种结构时没考虑好如果一个评论被回复得很多，实际上接口并未对评论回复，也就是 `child_comment` 做处理，现在的情况是有多少条评论的回复直接返回，这会带来一些数据传输量过大的问题，后续可考虑获取当前文章评论的回复最多设置上限为三条，当用户想要看更多回复时，客户端重新请求一个新的接口， `push` 到单独的一个页面中显示完整的数据（类似知乎）

阅读数

原本想做一个获取当前用户阅读过的文章，也就是阅读记录，想了一下 `Masquerade` 第一版要做的事情跟这个接口完全搭不上边，就按照简单原则出发，直接一个计数器搞完，后续如果有这个需求可接着开发。

点赞

点赞原本的设想也是想通过 `Redis` 去做，但是抛去了最初复杂的设想后，发现其实 `mysql` 做完全没有问题（其实就是懒），虽然性能上还是有问题，比如高并发时的频繁点赞、取消点赞，单机数据库肯定压不住，但是这都是以后要考虑的问题啦~不要过早的优化。具体实现细节都在代码中，本质上就是一个计数器，不过是带了一个 `bool` 的计数器。

其它

对 `request` 请求方法和参数校验做了两个装饰器， `token` 的校验因为每个接口都是必须，直接抽象出了一个中间件，过滤掉了注册和登录两个路由，其它都需要进行经过token装饰器。

未完待续