项目文档日志

一些成员在开发的时候有记录日志的习惯，就把一些日志整理了下来

推荐算法

推荐分为基于内容的推荐和协同过滤推荐，前者与用户行为无关，单纯关注物品本身的属性，对于一部番剧而言，内容标签包括类型tag（比如冒险、校园、悬疑等）以及导演、声优、脚本、监督、音乐、原画，在bangumi.tv中还有网站用户通过自行添加和投票产生的标签

比如上图，就是bangumi.tv中的某个详情页面，其中可以挖掘出相关的信息，基于内容的推荐可以通过获取每部番剧的feature vecter，通过对比各个番剧的特征向量生成相似度矩阵，再按相似度从大到小返回最后的推荐结果。

具体的实现方式，可以使用Python的Scikit learn模块，sklearn中的文本特征提取模块有CountVectorizer()函数，CountVectorizer会将文本中的词语转换为词频矩阵，它通过fit_transform函数计算各个词语出现的次数，可以很方便的将文本中的特征相似度算出来。我们把各部番剧的特征向量进行词频比较，就可以算出对于番剧A，其余番剧与它的相似度，我们最后需要的也就是一个相似度矩阵。

它的一些参数如下

参数表	作用
input	一般使用默认即可，可以设置为"filename’或’file’
encodeing	使用默认的utf-8即可，分析器将会以utf-8解码raw document

参数表	作用
decode_error	默认为strict， 遇到不能解码的字符将报UnicodeDecodeError错误， 设为ignore将会忽略解码错误，还可以设为replace， 作用尚不明确
strip_accents	默认为None，可设为ascii或unicode， 将使用ascii或unicode编码在预处理步骤去除raw document中的重音符号
analyzer	一般使用默认，可设置为string类型，如'word'，'char'，'char_wb'，还可设置为callable类型， 比如函数是一个callable类型

```python
# 连接数据库读取数据
def load_sql():
    conn = sql.connect(host="localhost", user = "root", password = "qwert789",
    sql_query = 'SELECT * FROM bangumi'
    df = pd.read_sql(sql_query, con=conn, index_col='bangumi_id')
    df['feature'] = ''
    count = CountVectorizer()
    # 增加一列特征列，通过create_feature函数
    create_feature(df)
    print(df['feature'])
    count_matrix = count.fit_transform(df['feature'])
    # 计算余弦相似度
    cosine_sim = cosine_similarity(count_matrix, count_matrix)
    print(cosine_sim)
    indices = pd.Series(df.index)

    id = 899
    recommended_bangumis = []
    idx = indices[indices == id].index[0]
    score_series = pd.Series(cosine_sim[idx]).sort_values(ascending = False)
    top_10_indexes = list(score_series.iloc[1:11].index)
    top_10_values = list(score_series.iloc[1:11].values)
    for i in top_10_indexes:
        recommended_bangumis.append(list(df.index)[i])

    recommended_result = {}
    for (i, bangumi) in enumerate(recommended_bangumis):
        recommended_result[str(int(bangumi))] = top_10_values[i]
    print(recommended_result)
    return recommended_bangumis
    conn.close()


def create_feature(df):
    for i in range(len(df['name'].array)):
        tags = df['tags'].array[i].split(',')
        cv_list = df['cv_list'].array[i].split(',')
        if len(cv_list) > 6:
            cv_list = cv_list[:6]
        try:
            staff_dict = df['staff_list'].array[i].replace('\t', '').replace("
            staff_dict = staff_dict.replace("{\'", "{\"").replace("\'}", "\"}"
            staff_dict = json.loads(staff_dict)
        except:
            print(df['staff_list'].array[i])
        feature = tags + cv_list
        staff_feature = ["原作", "导演", "音乐", "原画"]
        for s in staff_feature:
            staff = staff_dict.get(s, '').split(',')[0]
            if len(staff) > 0:
```

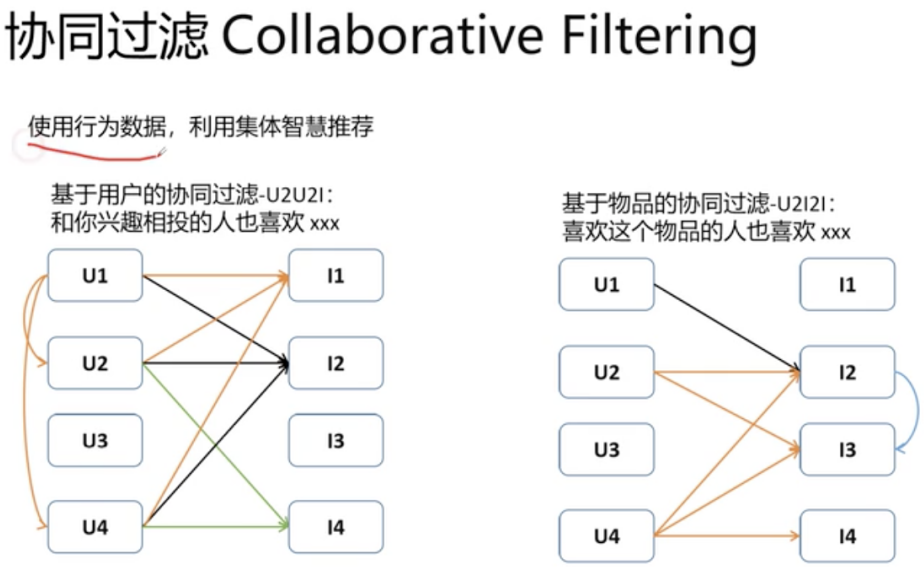
```
                feature.append(staff)

        feature = ','.join(set(feature))
        #print(feature)
        df.iloc[i, 9] = feature


load_sql()
```

仅仅使用基于内容的推荐（content-based），其实已经可以取得不错的效果，但是总觉得难以得到惊喜的结果，所以我们在之前开题报告的基础上，增加了协同过滤推荐（collaborative filtering，CF）方式，两种推荐方式的相似度数值会进行叠加

分为基于用户的协同过滤算法UserCF和基于物品的协同过滤算法ItemCF

UserCF一般针对用户数量比物品多的情况，它主要通过分析用户的行为记录计算物品之间的相似度，比如该算法会因为你购买过《数据挖掘导论》而给你推荐《机器学习》，因为很多用户在买了前者后会购买后者。

UserCF的推荐结果着重于反映和用户兴趣相似的小群体的热点，而ItemCF的推荐结果着重于维系用户的历史兴趣。换句话说，UserCF的推荐更社会化，反映了用户所在的小型兴趣群体中物品的热门程度，而ItemCF的推荐更加个性化，反映了用户自己的兴趣传承。



因为我们没法一开始弄出这么大的一个用户群体，所以也借助其他网站数年积累而来的用户数据，针对bangumi.tv的用户收藏进行爬取，主要针对收藏量在十部以上的资深用户数据进行爬取。

爬取用户的数据，写入一个csv文件中，具体代码在上面的爬虫部分提到了

接下来对这个用户数据进行处理：

读取

```
import pandas as pd
data_all = pd.read_csv('user.csv', header=None)
data_all.columns = ['user_id', 'bangumi']
print(data_all.head())
print(len(data_all.bangumi.unique()))
print(len(data_all.bangumi))
```

```
user_id  bangumi
0        1   268067
1        1   269235
2        1   231261
3        1   263808
4        1   221736
2343
8177
```

建立dataframe：按照收藏人数排序，筛选出被收藏的人多的稍微热门一点的番剧进行推荐

```
data_rec=data_all.groupby('bangumi').count()['user_id']
data_rec=data_rec.sort_values(ascending=False)
data_rec=pd.DataFrame(data_rec)
data_rec
```

```
user_id
bangumi
485     36
286     34
1424    33
311     32
276     32
...     ...
56108   1
56117   1
56118   1
56842   1
34227   1
2343 rows × 1 columns
```

```
data_rec.columns=['count']
data_rec = data_rec[data_rec['count']>5]
data_new = data_all.merge(data1,on='bangumi',how='inner')
data_new = data_new.sort_values(by='user_id', ascending=True)
data_new = data_new[['user_id', 'bangumi']]
```

```
user_id bangumi
0       1       268067
20      1       231261
6       1       269235
26      1       312
124     2       218712
...     ...     ...
4174    20785   1044
3313    20785   805
808     20785   51
2885    20785   292
3942    20785   240038
```

增加一列评分列，实际上我们没有评分，所以都置为1，然后写入data_new.csv，从而完成了用户数据的初步处理

```
i=0
like=[]
while i < 4573:
    y=1
    like.append(y)
    i=i+1
data_new.insert(1,'like',like)
data_new
data_new.to_csv('data_new.csv',index=False, header= False)
```

```
user_id like    bangumi
0       1       1       268067
20      1       1       231261
6       1       1       269235
26      1       1       312
124     2       1       218712
...     ...     ...     ...
4174    20785   1       1044
3313    20785   1       805
808     20785   1       51
2885    20785   1       292
3942    20785   1       240038
4573 rows × 3 columns
```

data_new.csv:

```
1,1,268067
1,1,231261
1,1,269235
1,1,312
2,1,218712
2,1,279457
2,1,267732
2,1,245665
2,1,235128
2,1,248175
2,1,266372
......
20728,1,2225
20733,1,235128
20733,1,240386
20733,1,266372
20765,1,12703
20765,1,10942
20785,1,827
20785,1,979
20785,1,485
20785,1,276
20785,1,1487
20785,1,1044
20785,1,805
20785,1,51
20785,1,292
20785,1,240038
```

重新读取data_new.csv文件，构建用户-物品矩阵

```
pd.options.display.width =200
pd.set_option('precision', 4)

import numpy as np
import matplotlib.pyplot as plt
import math
%matplotlib inline
train = dict()  # 用户-物品矩阵
for line in open('data_new.csv'):
    user, score, item = line.strip().split(",")
    train.setdefault(user, {})
    train[user][item] = int(float(score))
for k,v in train.items() :
    print("Key: " + k)
    print("Value: " + str(v)+'\n')
```

train的形式，key是用户id，value是一个json形式的用户收藏番剧id列表

```
Key: 1
Value: {'268067': 1, '231261': 1, '269235': 1, '312': 1}

Key: 2
Value: {'218712': 1, '279457': 1, '267732': 1, '245665': 1, '235128': 1, '248175': 1, '26637

Key: 6
Value: {'1856': 1, '823': 1, '133860': 1, '16235': 1, '1453': 1, '18624': 1, '2463': 1, '186

Key: 11
Value: {'536': 1, '605': 1, '299': 1, '337': 1, '2790': 1, '1424': 1, '1707': 1, '300': 1, '
......
```

```python
# 通过番剧被不同用户收藏的次数建立物品-物品矩阵
C = dict()  # 物品-物品的共现矩阵
N = dict()  # 番剧被多少个不同用户喜爱
for user, items in train.items():
    for i in items.keys():
        N.setdefault(i, 0)
        N[i] += 1
        C.setdefault(i, {})
        for j in items.keys():
            if i == j: continue
            C[i].setdefault(j, 0)
            C[i][j] += 1
            # 计算相似度矩阵
for k,v in N.items() :
    print("Key: " + k)
    print("Value: " + str(v)+'\n')
for k,v in C.items() :
    print("Key: " + k)
    print("Value: " + str(v)+'\n')
```

番剧-番剧共现次数矩阵N的形式：

```
Key: 268067
Value: 6

Key: 231261
Value: 6

Key: 269235
Value: 14

Key: 312
Value: 22

Key: 218712
Value: 10

Key: 279457
Value: 13

Key: 267732
Value: 7
...
```

番剧-番剧共现详情矩阵C的形式：
key是番剧id，value是番剧id列表json，值代表共现的次数

```
Key: 268067
Value: {'231261': 1, '269235': 3, '312': 1, '266794': 1, '216372': 1, '260680': 1, '238962':

Key: 231261
Value: {'268067': 1, '269235': 2, '312': 1, '263750': 2, '243916': 1, '211027': 2, '218708':

Key: 269235
Value: {'268067': 3, '231261': 2, '312': 1, '266794': 1, '273843': 5, '199228': 3, '266157':
......
```

计算余弦相似度

使用上面的公式(4)

$$\cos(\theta) = \frac{\sum_{i=1}^{n}(x_i \times y_i)}{\sqrt{\sum_{i=1}^{n}(x_i)^2} \times \sqrt{\sum_{i=1}^{n}(y_i)^2}}$$

计算两个句子向量

句子A：(1，1，2，1，1，1，0，0，0)

和句子B：(1，1，1，0，1，1，1，1，1)的向量余弦值来确定两个句子的相似度。

计算过程如下：

$$\cos(\theta) = \frac{1\times1+1\times1+2\times1+1\times0+1\times1+1\times1+0\times1+0\times1+0\times1}{\sqrt{1^2+1^2+2^2+1^2+1^2+1^2+0^2+0^2+0^2}\times\sqrt{1^2+1^2+1^2+0^2+1^2+1^2+1^2+1^2+1^2}}$$

$$= \frac{6}{\sqrt{7}\times\sqrt{8}}$$

$$= 0.81$$

```
# 计算余弦相似度
W = dict()
hot_bangumi = []
for i, related_items in C.items():
    W.setdefault(i, {})
    for j, cij in related_items.items():
        W[i][j] = cij / (math.sqrt(N[i] * N[j]))
for k,v in W.items():
    print("Key: " + k)
    print("Value: " + str(v)+'\n')
    hot_bangumi.append(k)
print(hot_bangumi)
```

W：番剧、番剧之间的相似度矩阵

```
Key: 268067
Value: {'231261': 0.16666666666666666, '269235': 0.3273268353539886, '312': 0.08703882797784

Key: 231261
Value: {'268067': 0.16666666666666666, '269235': 0.2182178902359924, '312': 0.08703882797784

Key: 269235
Value: {'268067': 0.3273268353539886, '231261': 0.2182178902359924, '312': 0.056980288229818

Key: 312
Value: {'268067': 0.08703882797784893, '231261': 0.08703882797784893, '269235': 0.0569802882
```

hot_bangumi返回了一个值得被推荐的列表，大约几百部番剧

```
['268067', '231261', '269235', '312', '218712', '279457', '267732', '245665', '235128', '248
```

最后的推荐

```
rank = dict()
action_item = {'766': 1, '767': 1, '812': 1, '56093': 1, '233': 1, '3423': 1,
for item, score in action_item.items():
    for j, wj in sorted(W[item].items(), key=lambda x: x[1], reverse=True)[0:3
        if j in action_item.keys():
            continue
        rank.setdefault(j, 0)
        rank[j] += score * wj

final = dict(sorted(rank.items(), key=lambda x: x[1], reverse=True)[0:10])

print('用户收藏的番剧列表 ： ' + str(action_item) +'\n')
print('最后推荐给用户的番剧列表及可信度 ： '+ str(final))
```

```
用户收藏的番剧列表 ： {'766': 1, '767': 1, '812': 1, '56093': 1, '233': 1, '3423': 1, '2225': 1}

最后推荐给用户的番剧列表及可信度 ： {'1333': 0.8423671308740659, '72941': 0.24618298195866545, '721
```

用户部分的api(基于Golang&MongoDB)

1. 查找所有用户

api URL: api/user/all
Method: get
Response example:

```
[
    {
        "_id": "5e6c4783684ae5798c3478e8",
        "username": "user1",
        "password": "xxxxxx",
        "avatar": "https://i.loli.net/2020/03/10/KhPeDH3twLNYuSb.png",
        "email": "123456@gmail.com",
        "role": 0,
        "bangumi_list": "307,310,312"
    }
]
```

2. 通过id或者用户名查找某个用户

api URL: api/user/<user_id>
或者api/user/username/<user_name>
Method: get
Response example:

```
{
    "_id": "5e6c4783684ae5798c3478e8",
    "username": "user1",
    "password": "xxxxxx",
    "avatar": "https://i.loli.net/2020/03/10/KhPeDH3twLNYuSb.png",
    "email": "123456@gmail.com",
    "role": 0,
    "bangumi_list": "307,310,312"
}
```


3. 通过用户名更新某个用户的信息

api URL: api/user/username/<user_name>
Method: put(更新信息，可用于用户任意信息的改变，传入完整的form data)
传入的FormData:（奇怪的原因，表单得用bangumiList，不能用bangumi_list)


```
{
    "_id": "5e6c4783684ae5798c3478e8",
    "username": "user1",
    "password": "xxxxxx",
    "avatar": "https://i.loli.net/2020/03/10/KhPeDH3twLNYuSb.png",
    "email": "123456@gmail.com",
    "role": 0,
    "bangumiList": "307,310,312,314"
}
```


Response example:


```
{
    "_id": "5e6c4783684ae5798c3478e8",
    "username": "user1",
    "password": "xxxxxx",
    "avatar": "https://i.loli.net/2020/03/10/KhPeDH3twLNYuSb.png",
    "email": "123456@gmail.com",
    "role": 0,
    "bangumi_list": "307,310,312,314",
}
```


4. 新增用户(就是注册)

api URL: api/user
Method: POST（传入一个表单，至少有用户名和密码，还可以有头像和邮箱)
传入的FormData:

```
{
  "username": "user3",
  "password": "xxxxxx",
  "avatar": "https://i.loli.net/2020/03/10/KhPeDH3twLNYuSb.png",
  "email": "user3@gmail.com",
}
```


Response Example: (Role是啥不用管)


```
{
    "_id": "5e6c4c03543c6e449a699717",
    "username": "user3",
    "password": "xxxxxx",
    "avatar": "https://i.loli.net/2020/03/10/KhPeDH3twLNYuSb.png",
    "email": "user3@gamil.com",
    "role": 2,
    "bangumi_list": ""
}
```


5. 登陆

API URL: api/user/login
Method: POST (传入一个表单，有两项：用户名和密码)
正确情况
传入的FormData:


```
{
  "username": "user3",
  "password": "654321",
}
```


返回:


```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhZG1pbiI6dHJ1ZSwiZXhwIjo
}
```


返回token (这里不用管token了，把userData存到localStorage里就行了)

错误情况:
传入的FormData:

```
{
  "username": "user3",
  "password": "654322",
}
```

返回Http.StatusUnauthorized，也就是401错误
数据部分则为错误信息：

```
{
    密码不正确!
}
```

6．某个用户收藏某部番剧

即番剧id加入该用户的收藏list

（理论上用put的那个api就可以在前端完成列表的新增和删除，但是为了减少前端工作量，后端另外开增加和删除的api）

api URL: api/user/username//addBangumi/<bangimi_id>（这里id其实可以是多个，一般一个就可以了）
比如 api/user/username/user3/addBangumi/425（赶时间就这样算了，长点应该没事）
Method: get(这里get就行了，因为url里都有)
Response Example:

```
{
    "_id": "5e6c4c03543c6e449a699717",
    "username": "user3",
    "password": "xxxxxx",
    "avatar": "https://i.loli.net/2020/03/10/KhPeDH3twLNYuSb.png",
    "email": "user3@gamil.com",
    "role": 2,
    "bangumi_list": "425"
}
```

比如我们再加一个：api/user/username/user3/addBangumi/436
现在返回

```
{
    "_id": "5e6c4c03543c6e449a699717",
    "username": "user3",
    "password": "xxxxxx",
    "avatar": "https://i.loli.net/2020/03/10/KhPeDH3twLNYuSb.png",
    "email": "user3@gamil.com",
    "role": 2,
    "bangumi_list": "425,436"
}
```


7. 某个用户取消收藏某部番剧

番剧id会从该用户的收藏list中移除，每次请求只支持一个bangumi_id

API URL: api/user/username//removeBangumi/<bangimi_id>
比如:
http://localhost:1323/api/user/username/user3/removeBangumi/436
Method: get
Response Example:


```
{
    "_id": "5e6c4c03543c6e449a699717",
    "username": "user3",
    "password": "xxxxxx",
    "avatar": "https://i.loli.net/2020/03/10/KhPeDH3twLNYuSb.png",
    "email": "user3@gamil.com",
    "role": 2,
    "bangumi_list": "425"
}
```


在Ubuntu上部署Go项目

服务器基本更新

```
sudo apt-get update
sudo apt-get upgrade
```

安装与配置Go

```
sudo apt-get install golang
```

使用 `go env` 查看GOROOT的位置，一般是 `/usr/lib` 或者 `usr/local` 里，记一下

配置GOPATH等信息:

`sudo vim ~/.profile`
添加这两行

```
export GOPATH=$HOME/go
# PATH实际就是$PATH:$GOROOT/bin，视GOROOT的位置更改
export PATH=$PATH:/usr/lib/go/bin:$GOPATH/bin
```

更新terminal source： `source ~/.profile`

安装与配置MongoDB

`sudo apt install -y mongodb`

`sudo mongod` 启动mongod服务

```
# 路径视具体情况更改，可以用whereis mongo来查找
cd /usr/local/mongo/bin
./mongo
```

进入mongo，可以新建数据表（collection），mongodb的语法不赘述

运行项目

在GOPATH/src下，把项目拷贝过去，通过 `go run <path_to_main>/main.go` 尝试运行，应该会报很多包丢失的错误，很正常因为没有安装那些包

通过 `go get xxx` 一个一个安装（国外服务器没问题，而国内的会报错，需要换源）

使用 `go build <path_to_main>/main.go` 生成二进制可执行文件main，记住它的位置，假如是 `/<absulute_path>/main` （之后要用到绝对路径）

持久化设置

后端服务需要一直维持运行，在之前的情况下，ssh断开后，后端服务就会停止

新建一个文件 `sudo vim /lib/systemd/system/goweb.service`

添加以下内容

```
[Unit]
Description=goweb

[Service]
Type=simple
Restart=always
RestartSec=5s
ExecStart=/<absulute_path>/main

[Install]
WantedBy=multi-user.target
```

现在就可以使用 `sudo service goweb start` 来启动后端，和
用 `sudo service goweb status` 来查看状态

配置nginx

使用nginx进行端口分发

安装nginx： `sudo apt-get install nginx`

```
cd /etc/nginx/sites-available
```

新建一个文件，比如 `goweb`

然后 `sudo vim goweb`

添加以下内容
(localhost的端口情况取决于项目运行时的端口情况)

```
server {
server_name your_domain www.your_domain;

location / {
    proxy_pass http://localhost:1323;
  }
}
```

在site-enabled文件夹下，运行

```
sudo ln -s /etc/nginx/sites-available/goweb /etc/nginx/sites-enabled/goweb
```

重载nginx： `sudo nginx -s reload` ，可以直接用ip或者域名访问到网站，而不需要指明端口

在Ubuntu上部署Django后端

相关软件版本：

Django 2.1.3
Python 3.6.6
nginx 1.14.0
uwsgi 2.0.17.1

服务器：
Ubuntu-server 18.04

安装python3-pip、python3-setuptools、gcc、python3-dev、wheel：
（缺一不可，不然之后用pip安装uwsgi会有各种各样的报错）

```
sudo apt-get install python3-pip python-setuptools python3-dev wheel
```

放置Django项目
在传输之前，要做一些工作：

先更改一下 `setting.py` 里的 `ALLOWED_HOSTS` ，把服务器的ip加进去，有域名的话顺便把域名也加进去，要不然之后会无法加载Django项目

在本地的Python虚拟环境上使用 `pip freeze > requirements.txt` ,生成一个txt文件，里面是需要的Python库以及其版本，之后一并传给服务器

传输文件到服务器的方法非常之多：可以使用Xshell自带的文件传输，也可以使用linux命令scp或安装更直观的lrzsz，或者使用本地的FileZilla、Winscp等软件，当然万能的git也很不错。

在服务器上使用 `pip install -r requirements.txt` 来安装必要的Python packages

安装与配置uwsgi

使用pip3安装uwsgi（注意是pip安装，不是apt-get，否则之后会各种报错）

```
pip3 install uwsgi
```

找个位置新建一个py文件，比如 `uwsgi_test.py` ，然后用vim打开

```
touch uwsgi_test.py
vim uwsgi_test.py
```

写入以下内容：

```
def application(env, start_response):
    start_response('200 OK', [('Content-Type','text/html')])
    return [b"Hello Uwsgi"]
```

然后输入以下命令启动uwsgi，把这个部署到某个端口，以9090端口为例

```
uwsgi --http :9090 --wsgi-file uwsgi_test.py
```

这时会出现 spawned uWSGI worker 1 (and the only) (pid: 11812, cores: 1)
找个浏览器，访问 http://<你的服务器ip>:9090/ ，不出意外的话你会看到Hello Uwsgi的字样，说明uwsgi能正常运行。

在项目目录下新建 uwsgi.ini 文件并编辑加入以下内容：

```
[uwsgi]
# 直接访问uwsgi的端口号，绕过nginx
http = :8010
# 转发给nginx的端口号
socket = 127.0.0.1:8001
# 是否使用主线程
master = true
# 项目的绝对路径
chdir = /var/www/<PROJECT_NAME>/
# Django项目wsgi.py文件的相对路径
wsgi-file = <PROJECT_NAME>/wsgi.py
# 进程数
processes = 4
# 每个进程的线程数
threads = 2
# 监听端口
stats = 127.0.0.1:9191
# 每次退出时是否清理环境配置
vacuum = true
# 目录中一旦有文件被改动就自动重启
touch-reload = /var/www/my_site
# 存放日志
daemonize = /var/www/my_site/uWSGI.log
```

加入uwsgi.ini的目的是使让uwsgi对接Django项目的启动变得更简便

有了 uwsgi.ini 我们只需要输入 uwsgi --ini uwsgi.ini 就可以运行，浏览器输入ip地址加:8010端口，发现可以显示我们的项目了

安装和配置nginx

先 sudo apt-get install nginx 安装nginx，安装后nginx会自动启动，默认端口为80端口，浏览器输入ip地址加:80，可以看到"Welcome to nginx"的欢迎界面

把/etc/nginx/目录下的 uwsgi_params 复制到项目目录下，也可以直接项目目录下新建 uwsgi_params 文件，写入以下内容：

```
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;
```

前往/etc/nginx/目录，查看 nginx.conf （nginx基础配置），发现里面有这么两行，意思就是包含conf.d文件夹中所有以conf后缀的配置和site-enabled文件夹中的内容

```
include /etc/nginx/conf.d/*.conf;
include /etc/nginx/sites-enabled/*;
```

不更改nginx.conf基础配置，只需要修改 conf.d 目录下的conf文件即可，进入 conf.d 文件夹，修改 default.conf 文件，没有的话就新建一个

然后写入以下内容：（务必根据自己的情况做相应更改）

```
upstream django {
   server 127.0.0.1:8001;
}

server {
    # 监听端口，可改
    listen       80;
    # 修改为你的ip或者域名
    server_name  1.2.3.4;
    # 编码方式
    charset utf-8;

    # 日志记录，可选
    access_log      /var/www/<PROJECT_NAME>/nginx_access.log;
    error_log       /var/www/<PROJECT_NAME>/nginx_error.log;

    # 静态文件所在目录（自行修改）
    location /static {
        alias /var/www/my_site/blog/static;
    }
    # 媒体文件所在目录（自行修改）
    #location /media  {
    #    alias /home/www/djangotest/Hello/media; # 媒体文件所在文件夹
    #}
    location / {
        include /var/www/<PROJECT_NAME>/uwsgi_params;
        uwsgi_pass django;
    }
}
```

运行 `service nginx restart`

如果报
错 `nginx.service failed because the control process exited with error code`，那么运行一下 `nginx -t -c /etc/nginx/nginx.conf`，可以很容易的找到问题在哪。

浏览器输入ip地址，发现看到的还是"Welcome to nginx"，这个是因为
在 `nginx.conf` 中还include了一个 `sites-enabled/*`，它覆盖了我们在 `default.conf` 中的配置，可以干脆直接去 `nginx.conf` 里把 `include /etc/nginx/sites-enabled/*;` 这一行删掉，或者调换两行位置。

这时再访问我们的ip，就能看到自己在本地搭建的Django项目了，因为在配置nginx的时候写入了static的路径，所以css什么的都加载进来了。

至此nginx配置完毕，Django项目就部署完成了

前端

使用了Vue.js作为前端架构，以下是在学习Vue.js时的一些笔记记录

Vue指令指的是那些带有 v- 前缀的特殊特性，是vue提供的特殊特性

v-bind：动态绑定一些属性

```
<div id="app-2">
 <span v-bind:title="message">
   鼠标悬停几秒钟查看此处动态绑定的提示信息!
 </span>
</div>
<script>
var app2 = new Vue({
 el: '#app-2',
 data: {
   message: '页面加载于' + new Date().toLocaleString()
 }
})
</script>
```

v-bind: 可以简写为 :

可以动态绑定多种class：

```
<div v-bind:class="{param1: A, param2: B}">
<div>

data: {
  A: true,
  B: false,
}
```

这个很常用，A、B也可以是比较式，从而判断是否添加param等属性，之后在css中定义

也可以绑定多种组件：

```
<my-component v-bind:class="{ active: isActive }"></my-component>


Vue.component('my-component', {
  template: '<p class="foo bar">Hi</p>'
})
```

绑定style

```
<div v-bind:style="styleObject"></div>

...

data: {
  styleObject: {
    font: 24px,
    color: red;
  }
}
```

最后的class属性就只有param1，不过都可以动态变化

v-if：根据后面的值是否为真来保留/移除该标签

```
<div id="app-3">
 <p v-if="seen">现在你看到我了</p>
</div>
<script>
var app3 = new Vue({
  el: '#app-3',
  data: {
    seen: true
  }
})
</script>
```

后面可以接 <v-else> 标签，表示else（后面不加条件），vue2.10新增 <v-else-if> ，表示else if

用key管理可复用的元素

```
<div v-if="loginMethod === 'e-mail'">
    <label>邮箱登陆：</label>
    <input placeholder="input your email" key="email-login">
    <button @click="toggleLoginMethod">使用账号登陆</button>
    </div>
<div v-else>
    <label>账号登陆：</label>
    <input placeholder="input your username" key="username-login">
    <button @click="toggleLoginMethod">使用邮箱登陆</button>
</div>
```

如果key一样或没有key的话，切切换登陆方式后input里的内容不变

v-show：根据后面的值的真假来确定是否显示（都会渲染出）

和v-if很像，但是v-if内容如果为false，该标签将不会被渲染，而v-show只是决定该标签是否显示

v-for：可以用作遍历

```
<div id="app-4">
  <ol>
    <li v-for="todo in todos">
      {{ todo.text }}
    </li>
  </ol>
</div>
<script>
var app4 = new Vue({
  el: '#app-4',
  data: {
    todos: [
      { text: '学习 JavaScript' },
      { text: '学习 Vue' },
      { text: '整个牛项目' }
    ]
  }
})
</script>
```

v-on: 监听DOM事件

用于在触发时运行后面的一小段js代码，比如 `v-on:click="count += 1"`
更多的时候是触发某个函数（代码比较长的话），比如 `v-on:click="func(item)"`

`v-on:` 可以简写为 `@`

v-model: 实现表单输入和应用状态之间的双向绑定

```
<input v-model="content">
...
data: {
  content = '';
}
```

动态属性

2.6新增， `v-bind:[name]="..."`
可以动态得到name，如name="href"字符串，那么就是 `v-bind:href="..."`

模板语法

文本： {{ string }} 直接展示string
html代码 v-html="string" 把string做html解析
简单运算/操作 {{ message.split('').reverse().join('') }} , {{ nuber + 1 }}
不能是完整语句或流控制

组价

组件的作用就是为了复用

声明组件

```
Vue.component('tag-name', {
  props: { // 参数
    prop1: String,
    prop2: Boolean,
  },
  template:`
    // 需要复用的html代码段
  `
  data: function () {
    return {}
  },
  methods: {

  }
})
```

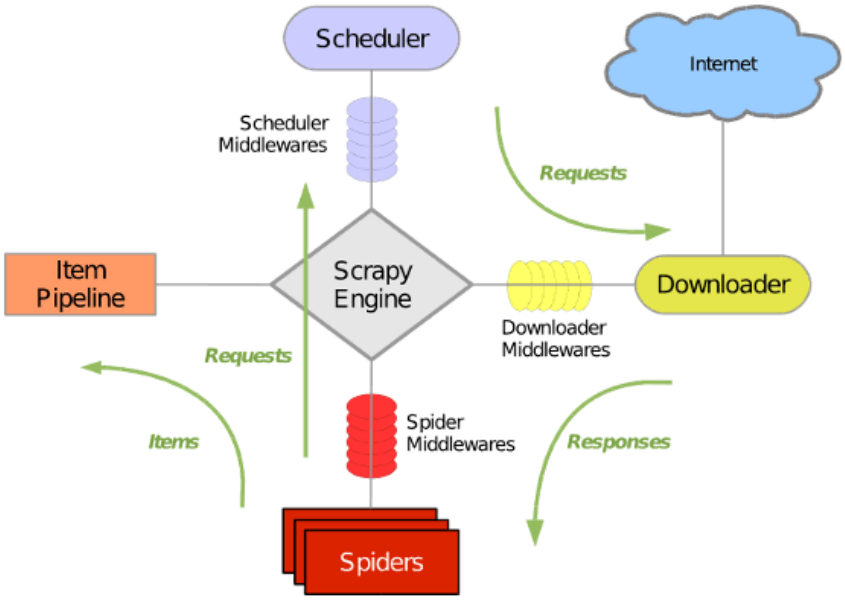
使用组件：

```
<tag-name :prop1="" :prop2=""> </tag-name>
```

爬虫模块

Scrapy是python crawler最强大和最著名的框架，在我们的项目中，获取番剧信息用到了scrapy

Scrapy包含以下七个部分：
-engine：就像大脑，Spider的调度中心
-调度程序：initail和以下url将放在调度程序中
-下载器：获取指定页面的数据
-Spider：类来查找特定的消息，而Spider负责一个任务
-项目管道：负责数据处理
-下载器中间件：engine与下载器之间的一个特定挂钩，可扩展scrapy的功能
-Spider中间器：engine和Spider之间的一个特殊钩



启动一个scrapy项目： `scrapy startproject bangumiSpider`

```
demo/
    scrapy.cfg
    demo/
        spiders/
            __init__.py
        __init__.py
        items.py
        middlewares.py
        pipelines.py
        settings.py
```

创建一个Spider

```
scrapy genspider bangumi bangumi.tv
```

随后在items.py里创建爬虫需要的字段名称

```python
import scrapy


class BangumispiderItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    pass


class BangumiItem(scrapy.Item):
    bangumi_id = scrapy.Field()  # 番剧id
    name = scrapy.Field()  # 番剧名
    cover_url = scrapy.Field()  # 海报url
    bangumi_score = scrapy.Field()  # Bangumi网站的分数
    vote_num = scrapy.Field()  # 评分人数
    episode_num = scrapy.Field()
    tags = scrapy.Field()  # 前几个标签
    desc = scrapy.Field()  # 番剧描述
    staff_list = scrapy.Field()  # 制作者（以json表示）
    cv_list = scrapy.Field()  # 声优列表
```

编写爬虫：使用xpath、re、requests等模块，具体使用方法不赘述

```python
# -*- coding: utf-8 -*-
import scrapy
from bangumiSpider.items import BangumiItem
import json
import re
import requests


class BangumiSpider(scrapy.Spider):
    name = 'bangumi'
    allowed_domains = ['bangumi.tv']
    start_urls = ['https://bangumi.tv/anime/browser?sort=rank&page=234']

    def parse(self, response):
        bangumi_list = response.xpath('//*[@id="browserItemList"]//li/@id').ex
        for bangumi in bangumi_list:
            bangumi_id = bangumi.split('_')[-1]
            detail_url = 'https://bangumi.tv/subject/' + str(bangumi_id)
            yield scrapy.Request(url=detail_url, meta={"bangumi_id": bangumi_i
        pre_url = response.xpath('//*[@id="columnSubjectBrowserA"]/div[2]/div/
        if pre_url:
            previous_url = 'https://bangumi.tv/anime/browser' + pre_url[0]
            print("######## url = " + str(previous_url))
            # 把新的页面url加入待爬取页面
            yield scrapy.Request(previous_url, callback=self.parse)

    def parse_detail(self, response):
        bangumi_item = BangumiItem()
        info_dict = {}
        info_count = 1
        while True:
            try:
                info_key = response.xpath('//*[@id="infobox"]/li[' + str(info_
                           :-2]
            except IndexError:
                break
            else:
                if response.xpath('//*[@id="infobox"]/li[' + str(info_count) +
                    info_content = ','.join(
                        response.xpath('//*[@id="infobox"]/li[' + str(info_cou
                else:
                    info_content = response.xpath('//*[@id="infobox"]/li[' + s
                # print("i=" + str(info_count) + ', span=' + str(info_key) + '
                info_count = info_count + 1
                info_dict[info_key] = str(info_content)
        # info_json = json.dumps(info_dict, indent=2, ensure_ascii=False)
        # print(info_json)
        bangumi_id = response.meta['bangumi_id']
        try:
            name = info_dict['中文名']
```

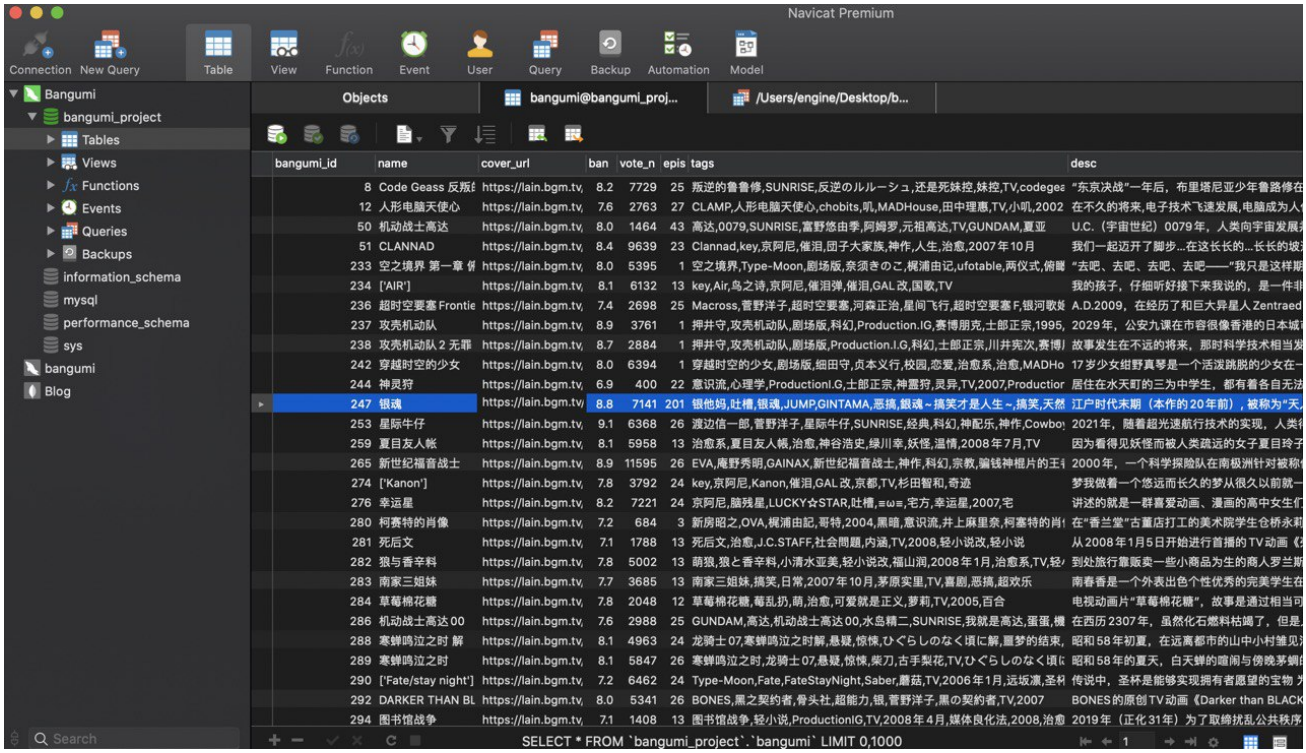
```
except KeyError:
    name = response.xpath('//*[@id="headerSubject"]/h1/a/text()').extr
try:
    episode_num = info_dict['话数']
except KeyError:
    episode_num = -1
try:
    cover_url = 'https:' + str(response.xpath('//*[@id="bangumiInfo"]/
except IndexError:
    cover_url = 'https:' + str(response.xpath('//*[@id="bangumiInfo"]/
try:
    bangumi_score = response.xpath('//*[@class="global_score"]/span[1]
except IndexError:
    bangumi_score = response.xpath('//*[@class="global_score"]/span[1]
try:
    vote_num = response.xpath('//*[@class="chart_desc"]/small/span/tex
except IndexError:
    vote_num = response.xpath('//*[@class="chart_desc"]/small/span/tex
desc = str(''.join(response.xpath('//*[@id="subject_summary"]/text()')
    '\u3000', '').replace('\u3002', '')
cv_list = []
cv_count = 1
while True:
    try:
        cv = \
        response.xpath('//*[@id="browserItemList"]/li[' + str(cv_count
            0]
    except IndexError:
        try:
            cv = response.xpath(
                '//*[@id="browserItemList"]/li[' + str(cv_count + 1) +
        except IndexError:
            break
        else:
            cv_count = cv_count + 1
    else:
        cv_count = cv_count + 1
        cv_list.append(cv)
# print(cv_list)
tag_count = 1
tag_list = []
while True:
    try:
        tag = \
        response.xpath('//*[@id="subject_detail"]/div[3]/div/a[' + str
            0]
    except IndexError:
        break
    else:
        if tag_count < 10:
            tag_count = tag_count + 1
```

```
                tag_list.append(tag)
            else:
                break
        # print(tag_list)
        bangumi_item['bangumi_id'] = str(bangumi_id)
        bangumi_item['name'] = str(name)
        bangumi_item['cover_url'] = str(cover_url)
        bangumi_item['bangumi_score'] = str(bangumi_score)
        bangumi_item['vote_num'] = str(vote_num)
        bangumi_item['episode_num'] = str(episode_num)
        bangumi_item['tags'] = list2str(tag_list)
        bangumi_item['desc'] = desc
        bangumi_item['staff_list'] = dict2str(info_dict)
        bangumi_item['cv_list'] = list2str(cv_list)
        print(bangumi_item)
        # yield bangumi_item


def list2str(l):
    result = ''
    for count, item in enumerate(l):
        result = result + item
        if count != len(l) - 1:
            result = result + ','
    return result


def dict2str(d):
    result = '{'
    for (key, value) in d.items():
        result = result + '\'' + str(key) + '\'' + ':' + '\'' + str(value) + '
    result = result[:-1] + '}'
    return result
```

其中用到了一些数据处理函数，另外需要二级页面抓取，即先在分页的列表页获取url，再去url抓取具体信息，使用yield关键字可以将获取的item结构体发送到piplines进行数据处理

在Piplines中，连接本地MySQL数据区、创建insert语句和执行，把一条条数据插入到bangumi表中

```python
import pymysql
from bangumiSpider import settings


class BangumispiderPipeline(object):
    def __init__(self):
        self.connect = pymysql.connect(
            host=settings.MYSQL_HOST,
            db=settings.MYSQL_DBNAME,
            user=settings.MYSQL_USER,
            passwd=settings.MYSQL_PASSWD,
            charset='utf8',
            use_unicode=True)
        self.cursor = self.connect.cursor()

    def process_item(self, item, spider):
        bangumi_id = pymysql.escape_string(item['bangumi_id'])
        name = pymysql.escape_string(item['name'])
        cover_url = pymysql.escape_string(item['cover_url'])
        bangumi_score = pymysql.escape_string(item['bangumi_score'])
        vote_num = pymysql.escape_string(item['vote_num'])
        episode_num = pymysql.escape_string(item['episode_num'])
        tags = pymysql.escape_string(item['tags'])
        desc = pymysql.escape_string(item['desc'])
        staff_list = pymysql.escape_string(item['staff_list'])
        cv_list = pymysql.escape_string(item['cv_list'])
        sql_command = "insert ignore into bangumi values ( '{0}', '{1}', '{2}'
        self.cursor.execute(sql_command)
        self.connect.commit()
        print("sql_command = " + sql_command)
        return item
```



在加入协同过滤模块之后，我们需要对用户数据进行爬取，这个比之前的简单

```
import requests
import re
import csv
import html
import lxml
from lxml import etree
import csv
import time


headers = {
'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_2) AppleWebKit/537
}


def request_user_page(url):
    bangumis = []
    blank = []
    try:
        resp = requests.get(url, headers=headers, verify=False)
    except:
        return []
    if resp.status_code == 200:
        resp = resp.content.decode("utf-8")
        tree = lxml.etree.HTML(resp)
        for i in range(16):
            bangumi = tree.xpath('//*[@id="browserItemList"]/li[' + str(i) + '
            if bangumi:
                bangumi = str(bangumi)[7:-2]
                bangumis.append(bangumi)
        if len(bangumis) > 12:
            return bangumis
        else:
            return blank
    else:
        return blank


def write_csv():
    with open('user.csv', 'w', newline='') as file:
        writer = csv.writer(file)
        for user_id in range(1, 30000):
            print("User:" + str(user_id))
            bangumis = request_user_page("https://bangumi.tv/anime/list/" + st
            if len(bangumis) > 0:
                for bangumi in bangumis:
                    writer.writerow([user_id, bangumi])
                    print(str(bangumi))
                time.sleep(2)
```

```
write_csv()
```

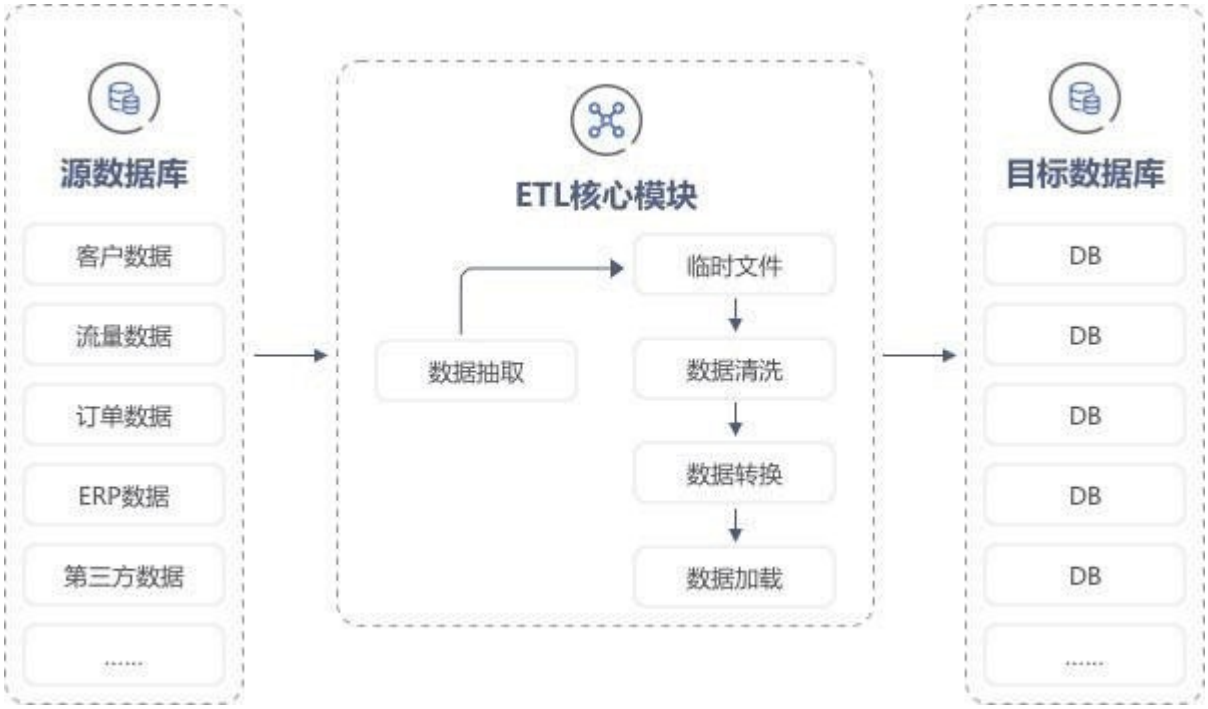
写到一个csv表格中

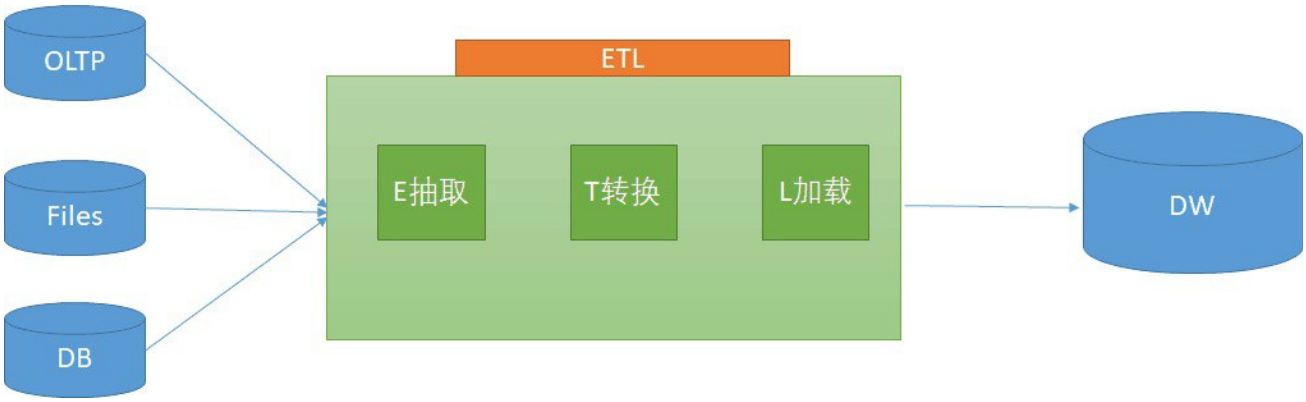
1	231261
1	263808
1	221736
1	227724
1	232067
1	217072
1	152091
1	209925
1	312
1	56846
1	117777
1	186620
1	206737
2	266372
2	279457
2	284546
2	267732
2	245665
2	235128
2	270294
2	193619
2	239646
2	248175
2	225604
2	240039
2	234349
2	218712
2	129807

暂时先爬取了八千多行的数据，存在user.csv中，用于之后的推荐模块协同过滤

数据清洗模块

大数据清洗的过程，即将数据从来源端经过抽取(extract)、转换(transform)、加载(load)至目的端的过程，即ETL。 ETL在转化的过程中，主要体现在以下几方面： 1． 空值处理：可捕获字段空值，进行加载或替换为其他含义数据。 2． 规范化数据格式：可实现字段格式约束定义，对于数据源中时间、数值、字符等数据，可自定义加载格式。 3． 拆分数据：依据业务需求对字段可进行分解。例如将番剧的staff list拆分为director与original等不同维度。 4． 数据替换：对于因业务因素，可实现无效数据、缺失数据的替换。 5． 建立ETL过程的主外键约束：对无依赖性的非法数据，可替换或导出到错误数据文件中，保证主键唯一记录的加载。 6． 字符串处理：从数据源某个字符串字段中经常可以获取特定信息，例如staff列表。而且，经常会有数值型值以字符串形式体现。对字符串的操作通常有类型转换、字符串截取等。但是由于字符类型字段的随意性也造成了脏数据的隐患，所以在处理这种规则的时候，会进行异常处理。 7． 日期转换。在数据仓库中日期值一般都会有特定的，不同于日期类型值的表示方法，例如使用8位整型20200202表示日期。而在数据源中，这种字段基本都是日期类型的，所以对于这样的规则，需要一些共通函数来处理将日期转换为8位日期值、6位月份值等。


源数据库
客户数据
流量数据
订单数据
ERP数据
第三方数据
.....
ETL核心模块
临时文件
数据抽取
数据清洗
数据转换
数据加载
目标数据库
DB
DB
DB
DB
DB
.....


OLTP
Files
DB
ETL
E抽取
T转换
L加载
DW