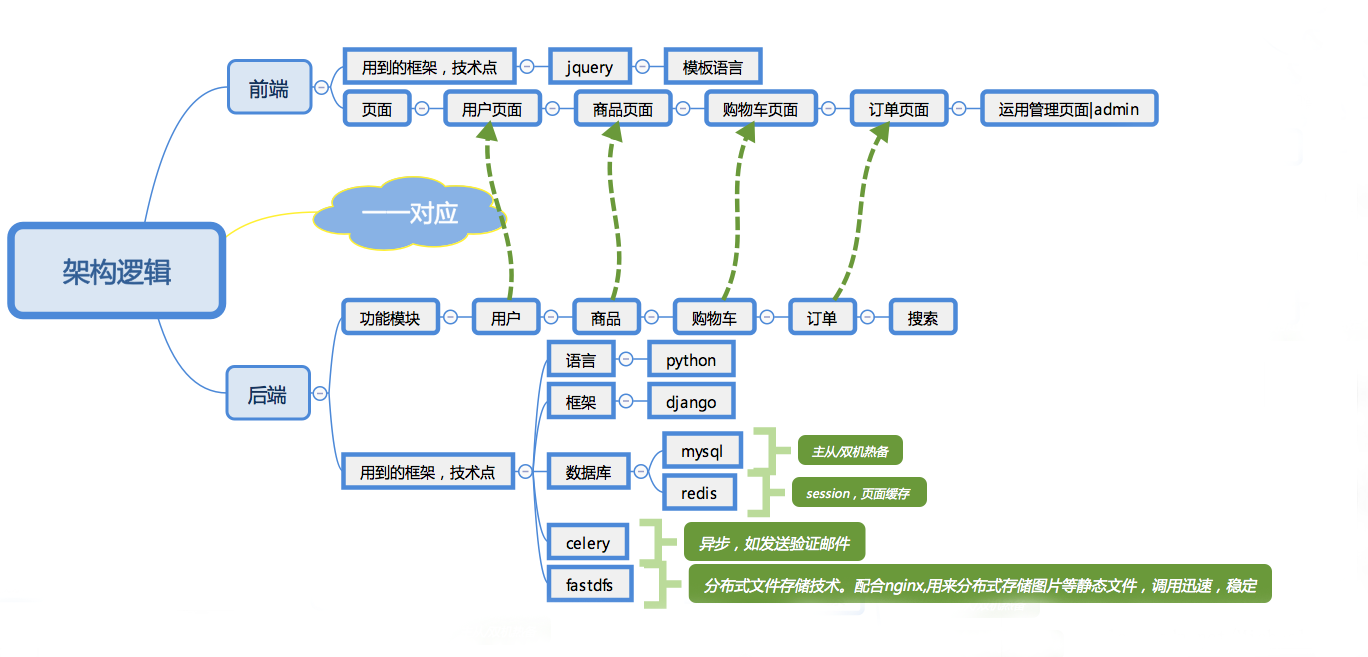
# **Dailyfresh**

基于B2C的天天生鲜商城

#### 技术栈

* 语言：Python3.\* (Django)
* 数据库: MySql、 redis
* 任务队列(异步处理): celery(django-celery)
* 分布式文件存储: FastDFS
* 搜索引擎(商品检索)： haystack(django-haystack)、whoosh、二次开发
* web服务器配置: Nginx+ uwsgi
* 开发环境： PyCharm、Linux、vim



#### 功能与性能优化

1. 用户注册发激活邮件时，可能发送邮件所需的时间较长，客户端会需要等待，用户体验不好。

改进: 把耗时的任务放到后台异步执行，此处使用celery任务队列, 其中使用redis作中间件。

1. redis存储用户历史浏览记录, 采用list数据结构: History\_用户id: [skuid1,skuid2,skuid3]
2. 使用redis存储用户购物车商品信息，采用hash数据结构: cart\_userid: {'sku\_id1': num, 'sku\_id2': num}
3. 采用分布式文件系统， 把商品图片等信息存储在FastDFS系统中, Nginx+FastDFS配合, 减少服务器的压力。
4. 页面静态化： 首页、商品列表页、商品详情页等用户共同的页面, 把页面静态化，以减少对数据库的操作。当后台数据更改时自动重新生成静态页。
5. 页面数据缓存，把页面使用的数据存放在缓存中，当再次使用这些数据时，先从缓存中获取，如获取不到，再去查询数据库，减少数据库的查询次数。
6. 订单并发

# **2019.12.24**

决定开始记录一下项目中遇到的问题，于是创建该文档。

目前项目进展：

正在做用户模块，用户的注册、登录功能已经完成，目前在进展在用户信息页面（信息页、订单页、地址页）

之前遇到问题：用户注册使用异步发送邮件的功能，因为最新celery的版本不适用于django1.8.2，导致无法开启worker，项目进度卡了一天，最后将当前celery卸载，重新安装celery==4.1.1，问题解决

# **2019.12.24**

使用用户模块的地址添加时浏览器报以下错误：

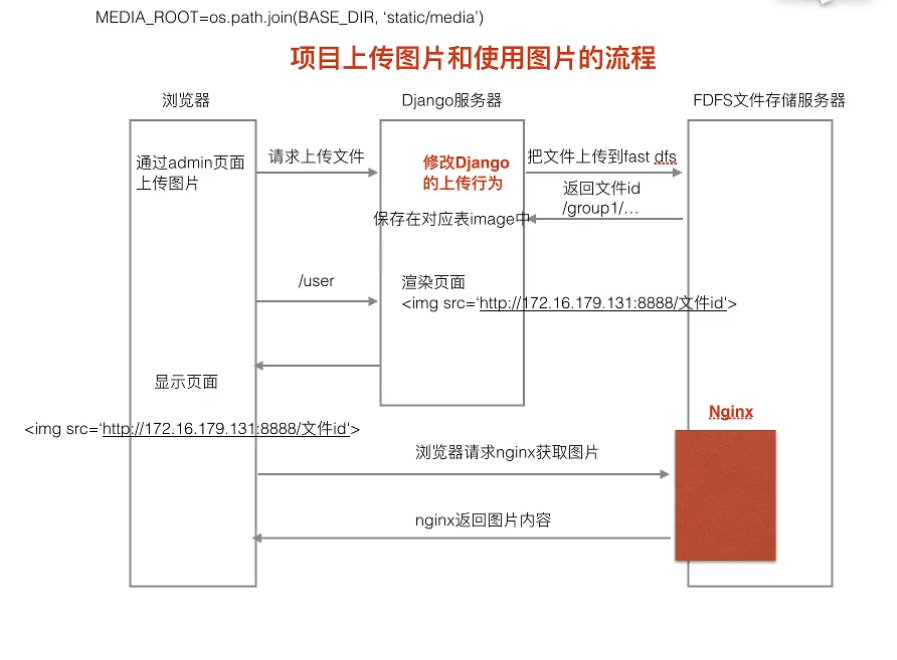
(1366, "Incorrect string value: '\xE5\xB9\xBF\xE4\xB8\x9C...' for column 'address' at row 1")

错误原因：mysql数据库默认的字符编码为latin1，不能正确显示中文。

当时创建数据库的时候忘记加charset = utf8参数，导致迁移的数据表全是latin1格式的了。

解决：删除原数据库，创建新数据库后，生成迁移文件，done。

…………由于期末考试，项目暂停……………



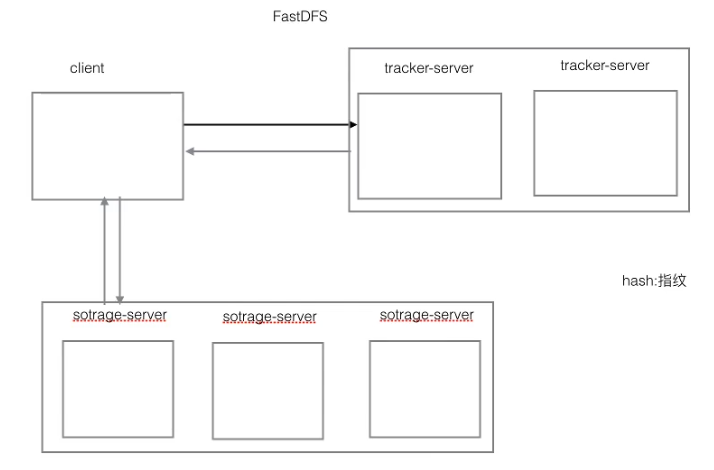
# **2020.1.12**

使用：

**分布式文件管理系统FastDFS:**

**海量存储，存储容量扩展方便**

**文件内容重复的话不用重新上传，结合nginx提高网站提供图片的效率**



**FastDFS安装与配置**

1 安装fastdfs依赖包

1. 解压缩libfastcommon-master.zip

2. 进入到libfastcommon-master的目录中

3. 执行 ./make.sh

4. 执行 sudo ./make.sh install

2 安装fastdfs

1. 解压缩fastdfs-master.zip

2. 进入到 fastdfs-master目录中

3. 执行 ./make.sh

4. 执行 sudo ./make.sh install

3 配置跟踪服务器tracker

1. sudo cp /etc/fdfs/tracker.conf.sample /etc/fdfs/tracker.conf

2. 在/home/python/目录中创建目录 fastdfs/tracker

mkdir –p /home/python/fastdfs/tracker

3. 编辑/etc/fdfs/tracker.conf配置文件 sudo vim /etc/fdfs/tracker.conf

修改 base\_path=/home/python/fastdfs/tracker

4 配置存储服务器storage

4.1. sudo cp /etc/fdfs/storage.conf.sample /etc/fdfs/storage.conf

4.2. 在/home/python/fastdfs/ 目录中创建目录 storage

mkdir –p /home/python/fastdfs/storage

4.3. 编辑/etc/fdfs/storage.conf配置文件 sudo vim /etc/fdfs/storage.conf

修改内容：

base\_path=/home/python/fastdfs/storage

store\_path0=/home/python/fastdfs/storage

tracker\_server=自己ubuntu虚拟机的ip地址:22122

5. 启动tracker 和 storage

sudo service fdfs\_trackerd start

sudo service fdfs\_storaged start

6 测试是否安装成功

6.1. sudo cp /etc/fdfs/client.conf.sample /etc/fdfs/client.conf

6.2. 编辑/etc/fdfs/client.conf配置文件 sudo vim /etc/fdfs/client.conf

修改内容：

base\_path=/home/python/fastdfs/tracker

tracker\_server=自己ubuntu虚拟机的ip地址:22122

6.3. 上传文件测试：

fdfs\_upload\_file /etc/fdfs/client.conf 要上传的图片文件

如果返回类似group1/M00/00/00/rBIK6VcaP0aARXXvAAHrUgHEviQ394.jpg的文件id则说明文件上传成功

2020.1.18 Nginx配合FastDFS使用的安装与配置

1. 解压缩nginx-1.8.1.tar.gz

2. 解压缩fastdfs-nginx-module-master.zip

3.安装nginx的依赖包

{

apt-get install build-essential

apt-get install libtool

apt-get install openssl

apt-get install libssl-dev

}

4. 进入nginx-1.8.1目录中

sudo./configure --prefix=/usr/local/nginx/ --add-module=fastdfs-nginx-module-master解压后的目录的绝对路径/src

5.编译

sudo make

sudo make install

在第5步可能会编译失败,错误如下

cc1: all warnings being treated as errors

objs/Makefile:440: recipe for target 'objs/src/core/ngx\_murmurhash.o' failed

make[1]: \*\*\* [objs/src/core/ngx\_murmurhash.o] Error 1

make[1]: 离开目录“/home/nginx/nginx-1.8.1”

Makefile:8: recipe for target 'build' failed

make: \*\*\* [build] Error 2

解决办法

将对应的makefile文件夹中（如本文中在 /nginx-1.8.1/objs/Makefile） 找到 -Werrori并删除 在重新make即可

---> /Werror <--- 在vim模式下快速定位

-Werror 视警告为错误;出现任何警告即放弃编译.

6. sudo cp fastdfs-nginx-module-master解压后的目录中src下的mod\_fastdfs.conf /etc/fdfs/mod\_fastdfs.conf

sudo vim /etc/fdfs/mod\_fastdfs.conf

修改内容：

connect\_timeout=10

tracker\_server=自己ubuntu虚拟机的ip地址:22122

url\_have\_group\_name=true

store\_path0=/home/python/fastdfs/storage

7. sudo cp 解压缩的fastdfs-master目录中的http.conf /etc/fdfs/http.conf

8. sudo cp 解压缩的fastdfs-master目录中的mime.types /etc/fdfs/mime.types

9.sudo vim /usr/local/nginx/conf/nginx.conf

在http部分中添加配置信息如下：

server {

listen 8888;

server\_name localhost;

location ~/group[0-9]/ {

ngx\_fastdfs\_module;

}

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root html;

}

}

10. 启动nginx

sudo /usr/local/nginx/sbin/nginx

# **2020.1.19 django对接fastDFS**

django二次开发对接FastDFS

如果你需要提供自定义文件存储 – 一个普遍的例子是在某个远程系统上储存文件 – 你可以通过定义一个自定义的储存类来实现

1.你的自定义储存类必须是django.core.files.storage.Storage的子类

。。。。详见django1.8.2中文文档—自定义存储系统

<https://yiyibooks.cn/xx/django_182/index.html>

2.在类中实现\_save方法创建Fdfs\_client对象使用其upload\_by\_buffer，上传文件到FastDFS系统里

# **2020.1.20 网站优化**

对网站进行优化，减少数据库查询的次数，一定程度上能防止恶意攻击（DDOS攻击）

页面静态化

1.使用celery 生成静态页面（储存在worker的项目目录里）

目的：当大量用户访问网站时，避免每一次访问都去查数据表

2.配置nginx提供静态页面

3.管理员修改首页所使用表中的数据时，重新生成index静态页面：

新建模型管理器类继承ModelAdmin类，重写它的save\_model(request, obj, form, change)和delete\_model(request, obj)方法，在管理员保存修改或删除时重新生成index静态页面。

数据缓存（redis）

把页面要用到的数据缓存起来，如果使用这些数据的时候，先从缓存中获取，如获取不到，再去查询数据库

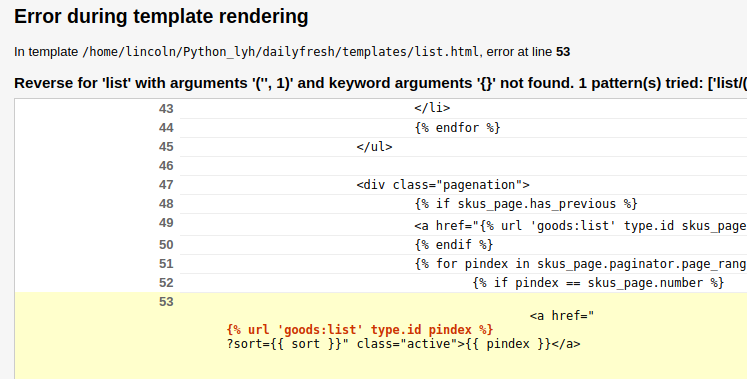
From Django.core.cache impot cache

设置缓存cache.set(key，value，timeout), 获取缓存cache.get(), 删除缓存cache.delete()

# **2020.1.22 全部商品列表**

项目进行到商品列表页，遇到如下错误：

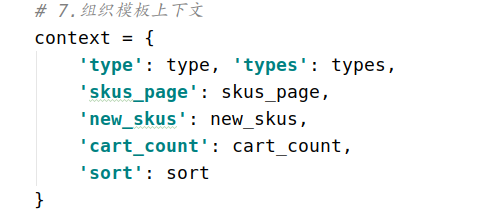
**Reverse for 'list' with arguments '('', 1)' and keyword arguments '{}' not found. 1 pattern(s) tried: ['list/(?P<type\_id>**[**\\d+)/(?P<page>\\d+)$**](file:///\\d+)/(%3fP%3cpage%3e\\d+)$)**']**



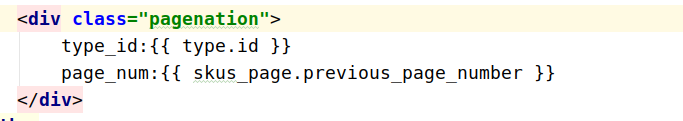
根据错误信息可以判断为url反向解析时参数缺少

由此排查渲染模板时传递给模板的数据正确性：

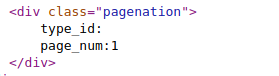
传递给模板文件的数据为以下：



选择在出错页面上直接显示传递过去的数据



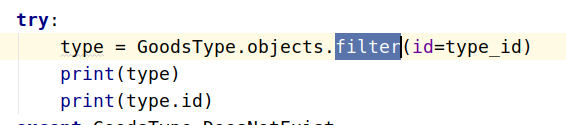
可以正常显示页面，检查网页源代码后发现type.id的数据缺失



然后继续从视图函数里排查，直接打印type.id ,发现又出现错误

**'QuerySet' object has no attribute 'id'**

**可从此错误信息中得知，type对象为一个查询集，在没有遍历其中的对象时（哪怕只有一个对象），没有id属性。**



**最终发现，我无意间使用了filter()方法查询数据库，而这里因为只需获取单个数据，不需要返回查询集，使用get() 后，问题得到解决**

# 2020.2.3 商品搜索栏

### 使用全文检索haystack框架

见另一文档

# 2020.2.4 购物车模块完成

购物车模块包括购物车商品的添加、删除，商品数目的更新，网页前端的勾选等

大量js代码，过了一下

**后台视图的核心：获取数据 校验数据 业务处理 返回应答**

**1）请求方式，采用ajax post**

**如果涉及到数据的修改（新增，更新，删除），采用post**

**如果只涉及到数据的获取，采用get**

2）传递参数：商品id 商品数量

**前后端开发时应该提前确立的几点：**

确定前端是否传递数据，传递什么数据，什么格式

确定前端访问的方式（get post）

确定返回给前端什么数据 什么格式

# 2020.2.5 订单模块的开发：订单提交

今天完了提交订单页面的显示，创建订单后台视图，以及创建订单前端的js

**订单的创建：**

用户点击提交订单时，**需要传递的参数：收获地址 支付方式 商品id**

后台涉及数据库两张表的添加记录：

订单信息表：df\_order\_info

订单商品表：df\_order\_goods

**用户每下一个订单，就需要向df\_order\_info表中加入一条记录**

**用户的订单中有几个商品，就需要向df\_order\_goods表中加入几条记录**

# 2020.2.6 解决订单高并发问题

1. 悲观锁

select \* from df\_goods\_sku where id=17 for update;



悲观锁获取数据时对数据行了锁定，其他事务要想获取锁，必须等原事务结束。

(2) 乐观锁

查询时不锁数据，提交更改时进行判断.

update df\_goods\_sku set stock=0, sales=1 where id=17 and stock=1;

应用场景：

冲突比较少的时候，使用乐观锁。

冲突比较多的时候，使用悲观锁。

# Mysql事务的思想

根据需要修改了mysql事务的隔离等级

1. 事务概念

一组mysql语句，要么执行，要么全不不执行。

1. 事务的特点

1、**原子性：**一组事务，要么成功；要么撤回。

2、**稳定性 ：**有非法数据（外键约束之类），事务撤回。

3、**隔离性：**事务独立运行。一个事务处理后的结果，影响了其他事务，那么其他事务会撤回。事务的100%隔离，需要牺牲速度。

4、**可靠性：**软、硬件崩溃后，InnoDB数据表驱动会利用日志文件重构修改。可靠性和高速度不可兼得， innodb\_flush\_log\_at\_trx\_commit 选项 决定什么时候吧事务保存到日志里。

1. 事务控制语句

BEGIN或START TRANSACTION；显式地开启一个事务；

COMMIT；也可以使用COMMIT WORK，不过二者是等价的。COMMIT会提交事务，并使已对数据库进行的所有修改称为永久性的；

ROLLBACK；有可以使用ROLLBACK WORK，不过二者是等价的。回滚会结束用户的务，并撤销正在进行的所有未提交的修改；

SAVEPOINT identifier；SAVEPOINT允许在事务中创建一个保存点，一个事务中可以有多个SAVEPOINT；

RELEASE SAVEPOINT identifier；删除一个事务的保存点，当没有指定的保存点时，执行该语句会抛出一个异常；

ROLLBACK TO identifier；把事务回滚到标记点；

1. mysql事务隔离级别

SQL标准定义了4类隔离级别，包括了一些具体规则，用来限定事务内外的哪些改变是可见的，哪些是不可见的。低级别的隔离级一般支持更高的并发处理，并拥有更低的系统开销。  
Read Uncommitted（读取未提交内容）

在该隔离级别，所有事务都可以看到其他未提交事务的执行结果。本隔离级别很少用于实际应用，因为它的性能也不比其他级别好多少。读取未提交的数据，也被称之为脏读（Dirty Read）。  
**Read Committed（读取提交内容）**

这是大多数数据库系统的默认隔离级别（**但不是MySQL默认的**）。它满足了隔离的简单定义：一个事务只能看见已经提交事务所做的改变。这种隔离级别 也支持所谓的不可重复读（Nonrepeatable Read），因为同一事务的其他实例在该实例处理其间可能会有新的commit，所以同一select可能返回不同结果。  
**Repeatable Read（可重读）**

**这是MySQL的默认事务隔离级别**，它确保同一事务的多个实例在并发读取数据时，会看到同样的数据行。不过理论上，这会导致另一个棘手的问题：幻读 （Phantom Read）。简单的说，幻读指当用户读取某一范围的数据行时，另一个事务又在该范围内插入了新行，当用户再读取该范围的数据行时，会发现有新的“幻影” 行。InnoDB和Falcon存储引擎通过多版本并发控制（MVCC，Multiversion Concurrency Control）机制解决了该问题。

Serializable（可串行化）   
       这是最高的隔离级别，它通过强制事务排序，使之不可能相互冲突，从而解决幻读问题。简言之，它是在每个读的数据行上加上共享锁。在这个级别，可能导致大量的超时现象和锁竞争。

         这四种隔离级别采取不同的锁类型来实现，若读取的是同一个数据的话，就容易发生问题。例如：

脏读(Drity Read)：某个事务已更新一份数据，另一个事务在此时读取了同一份数据，由于某些原因，前一个RollBack了操作，则后一个事务所读取的数据就会是不正确的。

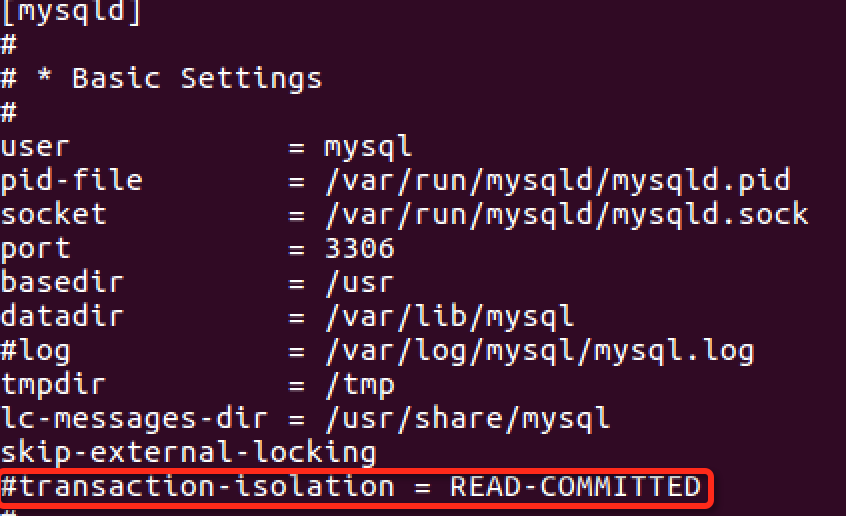
不可重复读(Non-repeatable read):在一个事务的两次查询之中数据不一致，这可能是两次查询过程中间插入了一个事务更新的原有的数据。

幻读(Phantom Read):在一个事务的两次查询中数据笔数不一致，例如有一个事务查询了几行(Row)数据，而另一个事务却在此时插入了新的几行数据，先前的事务在接下来的查询中，就会发现有几行数据是它先前所没有的。



1. 设置mysql事务的隔离级别

打开mysql配置文件: sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf, 添加如下行。



保存配置文件，重启mysql服务。

# 2020.2.8 对接支付宝支付

使用支付宝沙箱环境，

<https://openhome.alipay.com/platform/home.htm>

第三方python SDK来对接

https://github.com/fzlee/alipay

# 2020.2.9 项目部署、项目总结

**项目部署流程：**

## uwsgi

遵循wsgi协议的web服务器。

### 1.1 uwsgi的安装

pip install uwsgi

### 1.2 uwsgi的配置

项目部署时，需要把settings.py文件夹下的：

DEBUG=FALSE

ALLOWED\_HOSTS=[‘\*’]

[uwsgi]

#使用nginx连接时使用

#socket=127.0.0.1:8080

#直接做web服务器使用

http=127.0.0.1:8080

#项目目录

chdir=/Users/smart/Desktop/dj/bj17/dailyfresh

#项目中wsgi.py文件的目录，相对于项目目录

wsgi-file=dailyfresh/wsgi.py

processes=4

threads=2

master=True

pidfile=uwsgi.pid

daemonize=uwsgi.log

virtualenv=/Users/smart/.virtualenvs/dailyfresh

### 1.3 uwsgi的启动和停止

启动:uwsgi –-ini 配置文件路径 例如:uwsgi –-ini uwsgi.ini

停止:uwsgi --stop uwsgi.pid路径 例如:uwsgi –-stop uwsgi.pid

## 2 nginx

### 2.1 nginx 配置转发请求给uwsgi

location / {

include uwsgi\_params;

uwsgi\_pass uwsgi服务器的ip:port;

}

### 2.2 nginx配置处理静态文件

django settings.py中配置收集静态文件路径:

STATIC\_ROOT=收集的静态文件路径 例如:/var/www/dailyfresh/static;

django 收集静态文件的命令:

python manage.py collectstatic

执行上面的命令会把项目中所使用的静态文件收集到STATIC\_ROOT指定的目录下。

收集完静态文件之后,让nginx提供静态文件，需要在nginx配置文件中增加如下配置:

location /static {

alias /var/www/dailyfresh/static/;

}

### 2.3 nginx转发请求给另外地址

在location 对应的配置项中增加 proxy\_pass 转发的服务器地址。

如当用户访问127.0.0.1时，在nginx中配置把这个请求转发给172.16.179.131:80(nginx)服务器，让这台服务器提供静态首页。

配置如下:

location = /{

proxy\_pass http://172.16.179.131;

}

9.2.4 [nginx配置upstream实现负载均衡](http://www.baidu.com/link?url=tRw5xlP0ktNnJtBzk3YU6DNaWgD6zFGKamwpc8k4qastl1_eDyHAJH4sj0FSRsvtsd7k_wrl8PX7S_YGd7XpY_)

ngnix 配置负载均衡时，在server配置的前面增加upstream配置项。

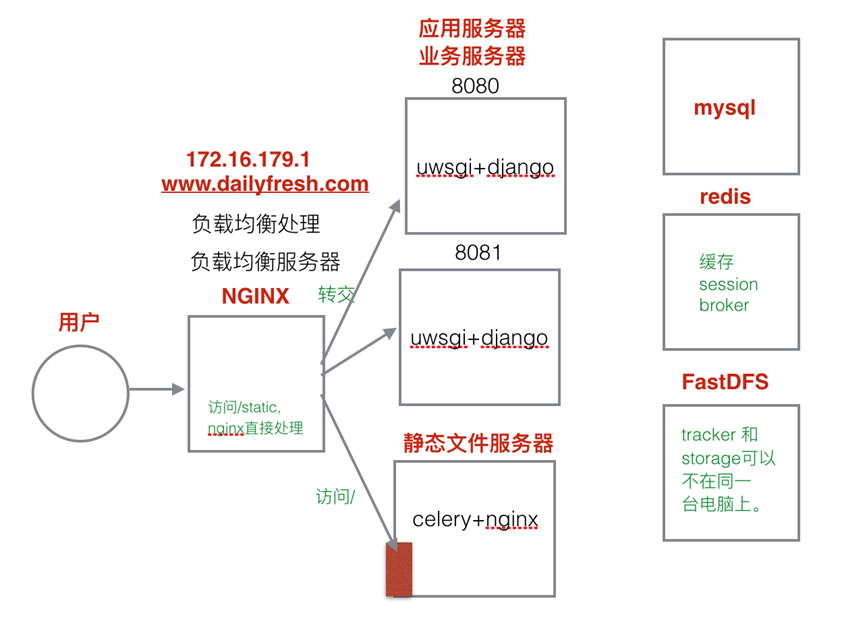
upstream dailyfresh {

server 127.0.0.1:8080;

server 127.0.0.1:8081;

}

### 3部署项目流程图



# 项目总结

1. 生鲜类产品 B2C PC电脑端网页
2. 功能模块：用户模块 商品模块（首页、 搜索、商品） 购物车模块 订单模块（下单、 支付）
3. 用户模块：**注册、登录、激活、退出**、个人中心、地址
4. 商品模块：**首页**、详情、列表、**搜索（haystack+whoosh）**
5. 购物车： **增加**、删除、修改、**查询**
6. 订单模块：确认订单页面、**提交订单（下单）、请求支付、查询支付结果、**评论
7. django默认的认证系统 **AbstractUser**
8. itsdangerous 生成签名的token （序列化工具 dumps loads）
9. 邮件 （django提供邮件支持 配置参数 send\_mail）
10. celery (重点 整体认识 异步任务)
11. **页面静态化 （缓解压力 celery nginx）**
12. **缓存**（缓解压力， 保存的位置、有效期、与数据库的一致性问题）
13. FastDFS (分布式的图片存储服务， 修改了django的默认文件存储系统)
14. 搜索（ whoosh 索引 分词）
15. **购物车**redis 哈希 **历史记录**redis list
16. ajax 前端用ajax请求后端接口
17. 事务
18. 高并发的库存问题 （**悲观锁、乐观锁**）
19. **支付的使用流程**
20. nginx （负载均衡 提供静态文件）