**数据库设计**

**用户信息表**

id

gmt\_create

gmt\_update

user\_name 也即display\_name

password md5

email

real\_name 正对中文

portrait 用户头像， 放在图片服务器上，或全路径

display\_name 显示名

first\_name 为英文保留

middle\_name 为英文保留

last\_name 为英文保留

mobile

telephone

is\_online 标识用户是否在线

status 预留字段，标识用户状态， 如 active, frozen等

is\_deleted 是否逻辑删除，以后定期清理数据

score 用户打分

degrade 用户级别： 免费、付费， 系统管理员等

接收消息类型：

Id

User\_id

Msg\_type\_id

接收消息范围：

Id

region\_type province, city, district或经纬度, customer， 如果是经纬度，则通过下面的经纬度范围来定义消息的发送范围

min\_ longitude

max\_ longitude

min\_ latitude

max\_ latitude

**后面扩展加商家信息表**

**用户的地址信息表**：可能一个人对应多个地址，为二手交换设计（送货地址），也可沿用

create table address(

id int(12) not null auto\_increment,

gmt\_create datetime,

gmt\_update datetime,

country\_id, 扩展字段，国家id

province\_id int(3),

city\_id int(6),

district\_id int(6),

zip\_code varchar(10),

address\_detail 详细地址， xx街道

PRIMARY KEY(id)

)

**合作方登陆**

create table partner\_user(

id int(12) not null auto\_increment,

gmt\_create datetime,

gmt\_update datetime,

user\_id int(12),

access\_token varchar(256),

refresh\_token varchar(256),

version varchar(6),

partner\_id,

partner\_detail json串，合作方详细信息：粉丝数，注册时间等

type int(2), 标识用户来自新浪weibo，还是腾讯微博等

PRIMARY KEY(id)

)

**省份信息**

create table provinces(

id int(2) not null auto\_increment,

name varchar(128),

PRIMARY KEY(id)

)

**城市信息**

create table cities(

id int(3) not null auto\_increment,

name varchar(128),

province\_id int(2),

zip\_code varchar(10),

PRIMARY KEY(id)

)

**区信息**

create table districts(

id int(4),

name varchar(128),

city\_id int(3),

PRIMARY KEY(id)

)

**消息类型表：**

**Msg\_type**

id int(4),

name varchar(64),

parent\_id, 父类型id

default\_expire\_time, 该类别下消息的默认过期时间，对于普通用户，如果用户不选择过期时间，默认<=该值

description varchar(256)

表示消息类型，如二手买卖消息，或路况等问答消息等

还可以加几个字段，给每个类别的消息一个默认的过期时间，和覆盖范围

保存每个类别下所有消息的数量， 还有每个用户每个类别下所有消息数量

**消息基本信息表：**

保存消息的最大、最小经纬度信息，这样可以通过商家地图选定的经纬度，加上覆盖的半径算出，覆盖正方形内

id int(16) not null auto\_increment,

gmt\_create datetime,

gmt\_update datetime,

user\_id,

user\_display\_name,

content varchar(300),

image 消息包含的图片信息，可以存放图片id，或图片地址， 如果放在图片服务器上，则全url

msg\_catogery\_id,

msg\_catogery\_name,

msg \_type 标识是消息，回复，还是转发的， 还有既是转发，又是回复

parent\_id 如果是消息，该字段为空，回复则是回复的消息Id，转发则是转发的消息Id

attach\_file 附件，预留字段，消息可以带附件

status 消息本身状态， delete\_by\_auth表示发送者删除, delete\_by\_admin表示管理员删除（可能违规）等

expire\_date 消息的过期时间

region\_type province, city, district或经纬度, customer， 如果是经纬度，则通过下面的经纬度范围来定义消息的发送范围

min\_ longitude

max\_ longitude

min\_ latitude

max\_ latitude

消息中url特殊处理，由于消息的字数限制，对于消息中的链接，需要特殊处理，即消息保存时解析消息，把url信息转换，保存到专门的表中， 再把转换后较短的链接替换真是的长链接。 当现实消息是，还是显示转化后的消息，当用户点击转换后的消息时，跳到我们的服务器，根据userId, msgId, 获取真是url, 然后跳转

**评价表：**

针对星打分：

Id

gmt\_create

gmt\_update

type

Score 1-5，表星数量, 对于普通消息，1-3代表赞，酱油或垃圾

User\_id

Msg\_id

Is\_used 打分是否已经被打分系统计算过

后面可以在用户表中加个score字段，值是由评价表的信息算出来的。 这个可以由后台定时器来跑， 定时更新用户的打分。

对于商家的， 可以根据打分级别来收费，分数越低，覆盖同等范围的区域，费用越高。

对于普通用户，如果打分太低，后面不能再发消息。

打分时结合消息被回复的次数

**用户勋章**

**收藏？ 收藏店， 或收藏有价值的消息**

**缓存设计**

**1. province, city, district信息放入缓存， 两种方案：**

a. 直接放在hashMap中，推荐放map中，因为是不变的内容

b. 通过redis缓存：

province:

|  |  |
| --- | --- |
| Key | value |
| province:$id:name | name |
| province:11:name | 浙江省 |

city:

|  |  |  |
| --- | --- | --- |
| Key | Hash | |
| city:$id:info |  |  |
| city:87: info | pid | 11 |
|  | name | 杭州市 |
|  | zipCode | 310000 |

District:

|  |  |  |
| --- | --- | --- |
| Key | Hash | |
| district:$id: info |  |  |
| District: 849: info | cityId | 87 |
|  | name | 滨江区 |

再存储省份对应的城市列表：

Key: Province:id:cities

Value: set类型的city列表

Key: city:id:districts

Value: set类型的district列表

**2. 所有的消息信息缓存，用户从缓存中取不到消息，就到数据库中找**

|  |  |  |
| --- | --- | --- |
| Key | Hash | |
| msg:$id:info |  |  |
| msg:20111123:info | content | 消息体撒 |
|  | Image | url |
|  | category | 消息类别，二手信息，还是悬赏等信息 |
|  | expire\_time | 2011-11-24:12:00:00 |
|  |  |  |

消息需要设置过期时间， 放入缓存后，根据过期时间设置

如果有需要，还可以存放每类消息对应的id的集合? 貌似目前不需要

**3. 用户信息缓存，缓存所有在线用户信息， 也是先缓存，再数据库**

|  |  |  |
| --- | --- | --- |
| Key | Hash | |
| user:$id:info |  |  |
| user:$id:info | email | email@163.com |
|  | displayName | 显示名 |
|  | Region | 地区信息，先区district:$id,取不到，再市 city:$id, 再取不到 province:$id, |
|  | longitude | 经度信息，非必须，可能拿不到 |
|  | **latitude** | 纬度信息 |

到时再看哪些是需要经常差查询的，对于较少获取的信息，直接访问数据库

4. 缓存地区下的用户列表， 用户对应的地区信息，秉承这样的原则：

A. 如果能拿到区级别的信息， 则用户的地区信息存区id

B. 如果能拿到市级别的信息， 则用户的地区信息存市id

C. 如果能拿到只能省级别的信息， 则地区信息存省id

province:

|  |  |
| --- | --- |
| Key | set |
| province:$id:users | name |
| province:11: users | 用户列表 |

city:

|  |  |
| --- | --- |
| Key | set |
| city:$id:users | name |
| city:11: users | 用户列表 |

District:

|  |  |
| --- | --- |
| Key | set |
| district:$id:users | name |
| district:11: users | 用户列表 |

5. 用户对应的消息信息，有些信息需要放到消息队列，有些可以缓存，需要与冯嘉确认，譬如每个用户有多少条消息

问题：

1. 回复的显示与存储

回复信息缓存：

|  |  |
| --- | --- |
| Key | Value |
| res:$id:content | content |
| res:34345234:content | 回复内容 |

回复关系缓存， 消息和回复在这个缓存中一样对待：

|  |  |
| --- | --- |
| Key | set |
| msg:$id:res:set | 回复列表 |
| msg:345345:res:set | 回复信息列表 |

显示一条消息时，回复下面可能还有回复，所以在这个缓存里递归取出所有的回复信息。

其实把所有的消息及其之间的关系全部放在缓存是很浪费的， 因为大多数的消息是很少人读取的，还不如直接查数据库，而且我们的内存资源也是很宝贵的。

到时要考虑下什么样的消息放入缓存。

1. 评价信息

评价最好分两套：

1. 商家的， 有5星打分
2. 非商家信息，包括消息和回复， 可以 赞（大拇指）， 酱油（酱油瓶）， 和垃圾（向下）

给商家的打分类似于淘宝的商家打分，让用户了解商家发布信息的真实度，可靠性

给其他用户的信息打分，是为了系统给用户评级，避免体统垃圾信息泛滥。 以后可以是用户打分低于某个值，账号冻结， 控制站内信息的质量。