删除tweet：Memcached中对应的单独的tweet可以存储tweet object，而tweet list这些只存储tweet id，渲染翻页时，for循环看看相应的id是否被删除了，就不展示了，并继续向下查找直到凑够一页的tweet。

queryset是一种懒惰加载方式，当它产生过数据库请求后，之后再访问，就不会再进行数据库请求了(如果使用iterator()来访问会重新产生数据库请求)，这也意味着如果数据库有更新，这个queryset也不会体现出来。

Memcached和Redis都缓存了tweet，其实是希望在Redis中缓存tweet id list，先从Redis中读取到tweet id，然后根据tweet id去Memcached中获取到具体的tweet对象，不过目前项目中直接在Redis中缓存了tweet list，但newsfeeds 缓存在redis中的list中存储的是tweet id，渲染newsfeeds时，需要从Memcached中获取tweet对象，还可以尝试在tweet view retrieve中进行优化，从Memcached中获取单个tweet对象。

点赞数和评论数的去标准化，虽然有可能导致获取的点赞数和评论数会不一致，但可以加快速度，在社交系统中这很重要，比一致性重要，而金融系统一致性最重要。

点赞数和评论数标准的取法是从数据库表中采用SELECT COUNT(\*) FROM Table，例如TweetSerializer中包含点赞数和评论数，前端渲染tweets列表时，显示时有10个tweet，那么点赞数和评论数就分别需要10次Query，这会导致N次Queries，而且sql语句中的COUNT(\*)也比较耗时，导致总的查询非常耗时。

解决办法是点赞数和评论数进行denormalization去标准化，都存储在tweet表中，相当于冗余存储。

所谓denormalization就是之前标准化获取点赞数和评论数只能通过SELECT COUNT(\*) FROM table来获取，现在可以直接从tweet的记录中直接查询出来。好处是查询tweet记录时就可以查询到点赞数和评论数，不用额外的查询，坏处是可能会产生一致性的问题，这时需要明确一个source of truth，即绝对准确的来源，点赞数和评论数的source of truth显然是SELECT查询表。

这种不一致的问题，可以不处理，例如点赞数和评论数就不需要处理，如果处理，可以采用随机的方式，在一定概率下取SELECT查询表来获取点赞数和评论数，大致代码如下：





mysql中这种操作是原子操作，它有row lock的功能，但不能先取出来，加上1，再写会去，这样无法保证是原子操作。

点赞操作时不要尝试把cache中的tweet和tweets列表失效掉，以保证likes\_count会被更新，这样做如果用户时明星用户，那么就会频繁的失效cache中的tweet，会降低访问速度。同理评论操作也不要失效cache。

点赞数和评论数会经常改变，要单独缓存到Redis。一般来说经常容易改变的对象不应该缓存在memcached或Redis中，除非像点赞数和评论数这样cache提供了相应的+1和-1操作，而且要单独缓存，不要和其他数据一起缓存。

异步任务(async task)：客户端发出的请求，有可能是短时间内做不完的，如果让用户一直等待，用户体验不好，好的做法是告诉用户需要等待或者表示服务端已经在做了，保证能完成，用户可以干其它的。例如，大规模的广告邮件，例如发10万封，点击发送时肯定没法很快完成，这时就需要异步任务。

Message Queue：用于任务分发的中间件。

前端：手机端，PC浏览器

大前端：前端+类似django web server

后端：Message Queue。

所有第三方都是不可信的。

同步任务(sync task)：操作没结束，需要一直等待。

服务器主动向客户端推送消息，通过Web socket来实现。

搞点简单的javascript，发起一个nginx请求，成功怎么处理，失败怎么处理。

任务队列：是一种在线程或机器间分发任务的机制。任务队列的输入是一个被称为任务的工作单元。专门的工作进程时刻监视任务队列以执行新的工作。

Celery通过消息通信，通常使用一个broker在客户端和workers之间调度。启动一个任务，客户端将一条消息到队列，broker传递这条消息给一个worker。

Celery系统可以包括多个workers和brokers，提供高可用性和水平扩展。

ratelimiter目的就是限制高频访问去打到数据库DB上。