参考：<https://www.zhihu.com/question/19645686>

**微博和知乎中的 feed 流是如何实现的？**

在微博中每个人关注的人会成百上千，在知乎上关注了人还关注话题关注特定主题，这些feed的输出，如何能有效降低系统查询负载，特别是知乎上feed流的结果还需要特定排序。不知道这些都是如何优化的，能否指点一二？

[**fengsp**](https://www.zhihu.com/people/fengsp)

Python 程序员

230 人赞同了该回答

简单来说，Feeds这块主要包括两块内容，就是生成feeds和更新feeds。生成feeds是什么意思呢，比如我们已经关注的人做了特定操作，我们需要把这些活动加入你的feeds，让你接收到（me：观察者模式）。更新feeds包括的内容比较多，一种就是你关注点做了更新，比如你新关注了一个人，需要把他的活动加入已有feeds，与此类似，取消关注也一样；另一种就是你的关注点做了一些更新操作，比如你关注的一个人取消关注了一个问题。

我们先来谈feeds生成，用户A做了某操作，比如关注了一个问题，这时候我们首先找到用户A的所有关注者，然后给需要推送的关注者推送此操作，大家可以把每个人的feeds简单想象为一个有序列表，推送很简单，就是在每个人的列表末尾添加这个操作。怎么样，是不是很简单，:)

然后我们谈feeds更新，第一种情况和第二种情况其实都是一样的，我们需要针对这些活动更新来更新feeds，比如我们新关注了一个人，这时候我们需要取出此人的活动历史，然后按照时间顺序把这些历史塞到你的feeds中。此操作的复杂度会比较高，需要使用合适的数据结构来达到最佳性能，目前是O(log(N))。

当然，真实情况并没有这么简单，还有很多其他逻辑和注意点，比如我们的feeds需要对多人做同样的操作做合并（大家可以自行把以上的feeds有序列表变为有序集合），同样的内容的创建时间是要按最新操作的时间来计算的，一个人连续做很多操作需要对操作做合并等。所有大家看到的feeds相应的存储考虑到性能都会使用内存，除此之外所有的操作都需要做持久化存储，否则我们也就没法更新feeds了:)

下面我们谈谈这里面的技术挑战和相关技术，此部分与知乎目前的技术决策和使用技术无关，我给大家分享几个国外团队的工程决策和他们的优化手段。

先来谈谈strava，他们使用了Kafka分布式发布订阅消息系统，这个系统用来事件(event)发布，还使用了Storm分布式实时计算平台，这个计算集群会订阅Kafka的事件，然后完成相应的处理，在这块他们做了一个优化，就是一个事件不会推给所有关注者，只会推给活跃的用户（如何判定一个用户为活跃的，这就看实际情况和数据自己优化了）。

然后再来谈谈Instagram，他们产品的读写比例达到了100：1，事实上大部分互联网产品都是这样，所以这也是推技术更合适的原因，推一次开销可能大一点，不过推（也就是写）发生次数大大少于读，因为有些大牛关注者非常多，达到几百上千万，为了进行可靠地推，这个过程都会异步后台执行。同样，我们需要一个任务调度器和消息队列，任务调度他们选用了Celery，下面需要选择一个消息队列，Redis依赖订阅者轮询，不自带复制备份，而且强依赖内存是个硬伤，不合适；Beanstalk其他方面不错，可还是不支持复制(Replication)，弃掉；最后他们选用了RabbitMQ，快，高效，支持复制，而且和Celery高度兼容。

接着谈谈Pinterest，重心在于创建一个智能化的feed，也就是feed会包括一些热点和推荐，而且会根据特定算法来排序。当事件发生后，会经过一系列流程最后才进入用户的内容feed，首先经过智能feed worker来处理，这些worker会接收事件而且根据特定用户给事件打分，然后这些事件会被插入到一个排好序的feed池中，不同类型的事件会被插入各自的池中，目前他们使用HBase的基于key的排序实现了此优先队列，接着智能feed内容生成器接手了，它会从几个池中根据策略取出feeds甚至剔除一些feeds，最后面向用户的智能feed服务会整合旧的feed和新生成的feed，展现给用户看到的Home Feeds。

最后简单谈谈Facebook，用户量大了之后对工程团队要求会更高，每个facebook用户都会有一个属于自己的独一无二的动态生成的首页。很多团队都会用用户ID为Key来把feeds存入Key-Value存储系统，比如Redis，问题是通过网络连接做远程过程调用非常慢，无法满足实时性的要求，所以facebook团队也开始使用了嵌入式数据库，他们也开源了自己在用的[RocksDB](https://link.zhihu.com/?target=http%3A//rocksdb.org/)。

这些内容都在他们的技术博客里面有提到，链接在这里：  
[Strava Engineering](https://link.zhihu.com/?target=http%3A//engineering.strava.com/)  
[How Instagram Feeds Work: Celery and RabbitMQ](https://link.zhihu.com/?target=http%3A//blogs.vmware.com/vfabric/2013/04/how-instagram-feeds-work-celery-and-rabbitmq.html)  
[Making Pinterest](https://link.zhihu.com/?target=http%3A//engineering.pinterest.com/post/96012794429/building-a-smarter-home-feed)  
[https://www.facebook.com/notes/10151822347683920/](https://link.zhihu.com/?target=https%3A//www.facebook.com/notes/10151822347683920/)

就像电影特效是为剧情服务一样，技术是为产品服务的。针对不同的业务场景，适合的技术也是不一样的。随着产品的调整和业务规模的扩大，相应的技术都会做进化和调整。针对不同的难题，需要提出不同的技术方案，feeds的生成也是这样，如果有必要，我们也会对这些方案做调整，目的都是一样，那就是又对又快又稳定。

如果有什么错误，希望大神指出。如果有更好的方案或者建议，欢迎交流。

[编辑于 2015-12-31](https://www.zhihu.com/question/19645686/answer/30086587)

230​14 条评论

​分享

​收藏​感谢收起

**知乎用户**

124 人赞同了该回答

刚好最近想写一个Feed机制的文章，就来回答一下吧。

楼上各位其实把大体的情况都已经说的很明白了，我来分享一下我们目前线上一个feed实现机制，已经在生产环境运行了大半年。理论来说，百万级别用户没有什么问题。

为了节省大家流量，全程无图（其实是我懒得画 - - ），希望大家能把省下的流量钱来给我发红包，鼓掌 。

首先，抛去数据库这一块，数据库我想大家肯定知道怎么设计，但是查询肯定是个麻烦事， 所以我使用了redis进行一个冗余设计，开始介绍之前，需要了解什么是推拉模式，楼上说的两篇文章 s[新浪微博架构和FEED架构分析--人人架构\_paper0023\_新浪博客](https://link.zhihu.com/?target=http%3A//blog.sina.com.cn/s/blog_53b95aec0100ujim.html) ，还有推拉模式以及时间分区拉模式的分析 [微博feed系统的推(push)模式和拉(pull)模式和时间分区拉模式架构探讨](https://link.zhihu.com/?target=http%3A//www.cnblogs.com/sunli/archive/2010/08/24/twitter_feeds_push_pull.html) 其实已经足够了解了，请各位看官如果未对推拉模式了解，那么请先看文章，再来看我的回答，我的回答只是阐述具体实现，谢谢。

那么我现在说一下redis这块怎么来完成推拉模式，以及内存尽可能节省，速度尽可能提高吧（当然，只是我理解的节省内存跟速度提高，如果看官们有其他的意见，我就两点要求，一、轻喷，二、说完再喷）。  
**1、实现**  
首先，先解决发布跟接收的问题，目前有大体以下几种方式：  
1、推模式  
什么是推模式？推模式就是，用户A关注了用户B，用户B每发送一个动态，后台遍历用户B的粉丝，往他们粉丝的feed里面推送一条动态。   
2、拉模式  
与推模式相反，拉模式则是，用户每次刷新feed第一页，都去遍历关注的人，把最新的动态拉取回来。

但是，不管推模式还是拉模式都存在若关注数量或者粉丝数量过多，导致遍历时间太长的问题，怎么去解决 ？这里就出现了第三种模式，推拉模式。  
3、推拉模式  
这是一种折中的解决方案，就是在线推，离线拉。粉丝几百上千万， 跟你发布动态同时在线的肯定也就只有那么顶天几百几千几万，何况这类大V很少，只推给在线的粉丝，离线的粉丝上线后，再去拉取动态即可！但是，不管是什么模式，每个用户都会维护一个类似发件箱跟收件箱的东西，保存自己发过的动态以及Feed动态（具体实现看下面），来完成推与拉。

而这里讲的，肯定就是推拉模式，用户A关注了用户B ， 用户B发布动态则将动态推进用户A的feed，这里使用redis的zset实现，sort为time（记得以毫秒为时间戳，秒级在数据量达到一定程度后，会有读取不到的问题，比如以时间戳为分页页码），value为具体的动态 ID（为什么是动态ID， 其实很简单， 就是因为动态的内容可以进行缓存，在redis里面全部走ID，修改动态内容也需要修改一处，动态内容可以保存在hash结构里）， 每个用户维护一个zset保存我发布的动态，一个zset保存我的feed动态，过期时间3~7天看情况而定。为什么要设计过期时间后面会细说。  
OK，全局维护一个在线用户列表，怎么设计这个就自己琢磨了，为了防止用户挂后台导致与服务端为离线状态，所以最好是1~3小时未操作或者离线时间不大于3小时的，都当做在线处理，反正这个看情况定。  
那么，当用户发了一条动态后，后台会有以下这些操作：  
在线推： 异步遍历在线的粉丝，将动态ID，添加到粉丝的Feed中。  
离线拉： 离线用户打开APP后，我们是会请求一个公共的入口接口，主做统计以及其他初始化操作，在这里，我们也开了一个异步线程，对用户进行Feed更新操作，防止用户进入APP后等待拉取时间过长，毕竟关注成千上万的人肯定有（其实万单位以下遍历都很快）。拉取过程其实就是把自己最后一条Feed的时间戳取出，去遍历关注的人的feed，将大于该时间的ID全部拉取回来。用户进入APP后，刷新即可看到最新操作。  
另：如果有Feed新消息数提示的需求，可以在推拉的同时进行增加， 刷新feed时清空即可。

**其实到这里，发布接收的问题已经解决了，那么有一个问题，用户feed里面过长，占用内存怎么办？**  
我是这么处理的，一个用户的feed第一次拉取的时候，feed长度为500条，在我们APP里，相当于50页，而后的数据，都走数据库。大页码翻页其实就是个伪需求而且耗性能的东西，用户除了第一次用这个APP，才会翻到底，第一次使用， 能有几个动态 ？而对于二次使用以上的用户，一般来讲， 翻了几页就已经到达上一次看过的地方了，所以500条数据，在关注量一般的情况下，内容已经足够消费，甚至达到疲劳，可能有关注量很大的用户他的Feed每天可能有很多很多动态，但是，不用说，肯定是做广告的，关注一堆人等着回粉，这种人更不会去消费内容，50页的内容，翻起来都累。当然，并不是说放弃了这些人，feed找不到走数据库嘛~~~~爱走不走，想走就给我翻50页再说~

**还有一个问题，每个用户都维护自己的动态跟Feed队列，当用户上百万时，内存的占有量肯定不小，要怎么释放内存才合适 ？**  
这里就回到上面那个问题了，为什么要给feed的key设计过期时间？为什么是设计3~7天过期时间？  
原因有以下：  
一、一个用户3~7天不打开APP，可能已经对APP失去兴趣了，打开几率很小，或者已经被卸载了，没有存在的意义了。  
二、3~7天未登陆APP，关注的人发的动态也不少了，Feed未拉取回来的数据肯定也不少，那么这时候去遍历其实拉取量很大，那么还不如直接全部重新拉一边或者拉取用户最后登陆时间后产出的数据。

到这里，其实已经差不多了，大部分业务逻辑已经足够满足，并且速度也理想，目前我们线上这种模式走了半年，feed一般都是10~80ms响应完毕。

好了，大概就是这样了。  
最后说一句：  
<?php   
echo 'php 是世界上最好的语言！！！';  
?>

[发布于 2016-02-06](https://www.zhihu.com/question/19645686/answer/85075806)

124​22 条评论

​分享

​收藏​感谢收起

**知乎用户**

19 人赞同了该回答

先不考虑数据的物理分布，仅讨论下业务的设计。

有三种基本思路：  
思路1，产生的所有动态，都在同一条索引上，上面承载所有的更新，以及读取。每个用户都有自己的filter规则，这个规则包括「屏蔽了谁，关注了谁，屏蔽了哪个问题」。  
用户在读取feeds，便是使用自己的filter，按照时间顺序，从头遍历这个索引，直到得到足够的条目。

优点：业务逻辑上是最清晰，性能稳定。  
缺点：技术难度最大。

思路2，读扩散。  
每个人有自己产生的feeds队列，打开自己的首页时，按照自己的关注列表和屏蔽规则，去读取其他用户的feeds队列，然后汇总、排序。

优点：实现最简单，关注列表 敏感性最高。  
缺点：性能最差，而且差的很稳定。

思路3，写扩散。  
每个人有自己的产生的feeds队列和 待读取的feeds队列。每个用户产生动态时，压入到关注者的 待读取feeds队列，压入前，需要根据屏蔽规则来判断下是否要压入。

优点：每个用户页面打开的速度是最快的，性能最高。  
缺点：关注列表变化时，敏感度略低，但读取队列的时候，再根据规则过滤一遍，也没啥太大问题。

**个人推荐：思路1 。**

根据个人的简单猜测，**知乎的首页，貌似是 思路3+思路2 。**  
猜测的理由：  
1-当我取关时，刷新，页面还会有这个用户的动态，所以 思路3的可能性很大。  
2-当我新关注一个人，刷新，页面上会有这个用户的动态，可能针对最近产生的关注行为做了特殊处理，少量的做了读扩散。  
3-B关注了A。如果A账号产生一条动态，比如 赞同，然后再取消赞同。此时A账号的 全部动态 中，是没有刚才的赞同记录的，但是在 B 的首页上，还是会有 A 的赞同行为。

-----------------------  
update at 2014-12-08  
我今天偶然的发现，我关注的某些人的回答，是没有出现在我的首页的。  
我反复对比过，估计 大概是4天前的4、5条动态，没有出现在我的首页。  
根据这个现象，知乎的首页应该是 写扩散更多些。  
而且，在写扩散的时候，貌似写失败了，就拉倒了。

[编辑于 2016-02-02](https://www.zhihu.com/question/19645686/answer/30092960)

19​11 条评论

​分享

​收藏​感谢收起

[**gglinux**](https://www.zhihu.com/people/gglinux)

[http://gglinux.com/](https://link.zhihu.com/?target=http%3A//gglinux.com/)

7 人赞同了该回答

本文会先介绍几种不同的Feed设计，让大家对Feed实现有初步的了解。其次会对我们采用的Feed方案作出详细的解答。

**推方式**

推方式，是发生在用户触发行为（发布新的动态，关注某个人，点赞）的时候。在触发时，用户的自身行为会记录到对应的行为表中，其次用户的行为也会记录到自己的粉丝对应动态表中。

1. 用户A发布新的帖子（动态），帖子记录到帖子表（主表）中。
2. 发帖行为塞到队列（Redis List）中。触发异步操作，消费者会先读取用户的粉丝列表（uid分表），依次写入到用户的动态表（uid分表）中。
3. 前端读取用户动态Feed，使用过滤条件，读取用户的动态表（关联查询帖子表）。

使用推方式，对需求变更是易适应的。为什么这么说呢？因为用户每一次的行为，我们都有存储相应的数据（数据模型）。即使变更，只需更改逻辑层代码。另外性能较好，后台数据已经准备好了，无需复杂的SQL查询。当然这样做，也存在很多弊端。1. 如果在用户A发完动态后，其粉丝B取消关注了A。在这个时间差内，内容已经推送给粉丝B了。2. 数据量存储成本较大，假如一个用户的粉丝数是100万，在发帖后会写入100万条数据。

**拉方式**

拉方式，是发生在粉丝拉取Feed时。粉丝拉取自己的动态，首先会检索自己的关注用户（uid分表）。得到关注的uid之后，再根据uid去查询关注用户发布的帖子。

拉的模式相对是比较简单易实现的，另外对用户关系变更（新增，删除用户）是敏感的。其次也不存在数据存储压力。但在查询的时候，对帖子表本身压力是很大的。尤其是用户本身关注的人很多的话，会有很严重的性能问题。

**拉方式优化-伪实时拉取**

用户在登录APP时，会发送用户活跃态到服务端。活跃信号塞到队列中，消费者依次读取活跃态uid，得到用户的关注者列表。得到关注者列表后，会去帖子表，查询关注人的发布的帖子。写到用户自己的Feed中。

这种方式和对拉方式而言，能有效避免接口性能问题，相当于通过定时任务提前把用户的动态Feed跑出来。和推方式比较，推是比较盲目的，这种方式只需针对活跃用户即可，能避免存储浪费。缺点在于实时性不好，用户登录APP后马上进入自己的Feed页，此时如果后台用户动态还没跑完，接口读取的就是历史数据了。当然这种方式不适合知乎，微博这种类型的APP的。

**拉方式优化-分区拉取**

分区拉取，是为了避免频繁查询单一帖子表所采用的一种优化手段。通过对帖子按照时间片分表，每次查询都能均摊到不同的表中，以此减轻主表的压力。

**推方式优化-定时推**

定时推，是以常驻进程的方式读取用户的发帖行为，再批量写入到粉丝的动态表中。这种方式和推方式差不多，只不过可以对多个发帖行为做聚合。

**推方式优化-特定用户推**

特定用户推，是推方式的一种优化方法。用户发送帖子时，只对活跃的粉丝用户写入。当然活跃用户的判定策略，是需要商定的。

**综合**

以上几种方案，都有自己的利弊和适用场景。选择最适合自己的就是最好的。最后，我们采用了伪实时拉取这种方式。 因为我们的需求是对点赞用户的聚合展示，类似于知乎点赞这种。本身点赞行为会存储在两种类型的表（以帖子ID的点赞分表，以用户ID的点赞分表）中，如果单纯以拉方式话，会比较难处理的，而且有性能问题。其次，点赞这个行为的重要程度还不足以我们批量去存储的。

原文地址：[Feed设计与实现 - 知乎专栏](https://zhuanlan.zhihu.com/p/25614036)

[编辑于 2017-03-07](https://www.zhihu.com/question/19645686/answer/150259928)

7​添加评论

​分享

​收藏​感谢收起

[**胡一**](https://www.zhihu.com/people/8080)

阿里云自研分布式NoSQL数据库，求研发

1 人赞同了该回答

可以看这篇文章：《[如何打造千万级Feed流系统](https://link.zhihu.com/?target=https%3A//yq.aliyun.com/articles/224132)》

[编辑于 2017-10-16](https://www.zhihu.com/question/19645686/answer/195929546)

1​添加评论

​分享

​收藏​感谢

[**邱东方**](https://www.zhihu.com/people/qiu-dong-fang)

关注产品经理、电子商务、网络营销、创业融资、财经投资等

2 人赞同了该回答

这里看到一篇新浪微博做的技术分享，值得一看。虽然没有太多的技术实现细节，不过大的框架已经说明的差不多了。还是好几年前的分享，现在应该架构变化又大了很多，不过一个小网站参考的话，这些已经足够了。 [新浪微博架构和FEED架构分析--人人架构\_paper0023\_新浪博客](https://link.zhihu.com/?target=http%3A//blog.sina.com.cn/s/blog_53b95aec0100ujim.html)

另外一篇，一名网友对推拉模式以及时间分区拉模式的分析 [微博feed系统的推(push)模式和拉(pull)模式和时间分区拉模式架构探讨](https://link.zhihu.com/?target=http%3A//www.cnblogs.com/sunli/archive/2010/08/24/twitter_feeds_push_pull.html)

[发布于 2015-03-17](https://www.zhihu.com/question/19645686/answer/42234405)

2​添加评论

​分享

​收藏​感谢

[**杉枫**](https://www.zhihu.com/people/liu-bao-feng)

闲思录

feed流这是一个不断优化过程，最开始你没有偏好时候，即冷启动过程给你推荐些热门内容，通过你的点击浏览会不断给你画像，根据你的画像偏好给你推喜欢内容，也同时会推荐偏好相似以及新内容，避免范围太窄看不到新领域，这些内容会通过机器学习深度学习模型通过你的历史行为实时行为进行打分，即点击量预估，根据预估对内容进行排序展现。详细可以看下文章<https://zhuanlan.zhihu.com/p/29835658>，希望对大家有帮助。

[发布于 2017-10-27](https://www.zhihu.com/question/19645686/answer/250736841)

0​添加评论

​分享

​收藏​感谢

**匿名用户**

想象一下，每个人的feed就是一个list, 一般使用成熟的kv store来存储，可以是redis也可以casandra或者hbase或者mysql（当然要sharded）。用户前端产生的所有事件被统一发送到一个event bus，比如kafka或者各种message queue里面，然后应用层的server对这些事件进行监听，根据业务逻辑将新的item插入到下游用户的feed里面，虽说事实很复杂，但是落实下来就是一个prepend的操作而已。

基本都是这个思路，你可以把它叫做push模式，不过似乎采用pull的不多，所以push也就是业界主流了。其实没什么特别玄乎的。