参考：<http://blog.sina.com.cn/s/blog_53b95aec0100ujim.html>

新浪微博架构和FEED架构分析--人人架构

<http://tech.sina.com.cn/i/2010-11-16/17574872122.shtml>

<http://news.csdn.net/a/20100726/277273.html>

<http://www.infoq.com/cn/articles/ywh-sina-mini-blog-arch>

大家下午好，在座的大部分都是技术开发者，技术开发者往往对微博这个产品非常关心。最晚的一次，是12点多收到一个邮件说想了解一下微博底层是怎么构架的。很多技术人员对微博的构架非常感兴趣，就是一个明星他有300万粉丝，这个技术怎么来实现？今天在这里跟大家分享一下微博的底层架构，让大家对微博的底层技术有更好的了解。另外不管是做客户端、1.0、2.0、论坛、博客都要考虑架构的问题，架构实际上是有一些共性的。今天我通过讲解微博里面的一些架构，分析一下架构里面哪些共性大家可以参考。

首先给大家介绍一下微博架构发展的历程。新浪微博在短短一年时间内从零发展到五千万用户，我们的基层架构也发展了几个版本。

第一版就是是非常快的，我们可以非常快的实现我们的模块。我们看一下技术特点，微博这个产品从架构上来分析，它需要解决的是发表和订阅的问题。我们第一版采用的是推的消息模式，假如说我们一个明星用户他有10万个粉丝，那就是说用户发表一条微博的时候，我们把这个微博消息攒成10万份，这样就是很简单了，第一版的架构实际上就是这两行字。第一版的技术细节，典型的LAMP架构，是使用Myisam搜索引擎，它的优点就是速度非常快。另外一个是MPSS，就是多个端口可以布置在服务器上。为什么使用MPSS？假如说我们做一个互联网应用，这个应用里面有三个单元，我们可以有三种部署方式。我们可以把三个单元部署在三台服务器上，另外一种部署模式就是这三个单元部署在每个服务器上都有。这个解决了两个问题，一个是负载均衡，因为每一个单元都有多个结点处理，另外一个是可以防止单点故障。如果我们按照模式一来做的话，任何一个结点有故障就会影响我们系统服务，如果模式二的话，任何一个结点发生故障我们的整体都不会受到影响的。

　　我们微博第一版上线之后，用户非常喜欢这个产品，用户数增长非常迅速。我们技术上碰到几个问题。第一个问题是发表会出现延迟现象，尤其是明星用户他的粉丝多。另外系统处理明星用户发表时候的延迟，可能会影响到其他的用户，因为其他的用户同一时间发表的话，也会受到这个系统的影响。我们就考虑这个系统怎么改进。首先是推模式，这肯定是延迟的首要原因，我们要把这个问题解决掉。其次我们的用户越来越多，这个数据库表从一百万到一亿，数据规模不一样处理方式是有差别的。我们第一版单库单表的模式，当用户数量增多的时候，它不能满足就需要进行拆分。第二个是锁表的问题，我们考虑的是更改引擎。另外一个是发表过慢，我们考虑的是异步模式。

　　第二版我们进行了模块化，我们首先做了一个层，做了拆分，最右边的发表做了异步模式。第二个服务层，我们把微博基础的单元设计成服务层一个一个模块，最大是对推模式进行了改进。首先看一下投递模式的优化，首先我们要思考推模式，如果我们做一下改进把用户分成有效和无效的用户。我们一个用户比如说有一百个粉丝，我发一条微博的时候不需要推给一百个粉丝，因为可能有50个粉丝不会马上来看，这样同步推送给他们，相当于做无用功。我们把用户分成有效和无效之后，我们把他们做一下区分，比如说当天登陆过的人我们分成有效用户的话，只需要发送给当天登陆过的粉丝，这样压力马上就减轻了，另外投递的延迟也减小了。

　　我们再看数据的拆分，数据拆分有很多方式，很多互联网产品最常用的方法，比如说如可以按照用户的UID来拆分。但是微博用户的一个特点就是说大家访问的都是最近的数据，所以我们考虑微博的数据我们按照时间拆分，比如说一个月发一张表，这样就解决了我们不同时间的维度可以有不同的拆分方式。第二个考虑就是要把内容和索引分开存放。假如说一条微博发表的地址是索引数据，内容是内容数据。假如说我们分开的话，内容就简单的变成了一种key-value的方式，key-value是最容易扩展的一种数据。比如说一个用户发表了一千条微博，这一千条微博我们接口前端要分页放，比如说用户需要访问第五页，那我们需要迅速定位到这个记录。假如说我们把这个索引拆分成一个月一张表，我们记录上很难判断第五页在哪张表里，我们需要索引所有的表。如果这个地方不能拆分，那我们系统上就会有一个非常大的瓶颈。最后我们想了一个方法，就是说索引上做了一个二次索引，改变我们还是按照时间拆分，但是我们把每个月记录的偏移记下来，就是一个月这个用户发表了多少条，ID是哪里，就是按照这些数据迅速把记录找出来。

　　异步处理，发表是一个非常繁重的操作，它要入库、统计索引、进入后台，如果我们要把所有的索引都做完用户需要前端等待很长的时间，如果有一个环节失败的话，用户得到的提示是发表失败，但是入库已经成功。所以我们做了一个异步操作，就是发表成功我们就提示成功，然后我们在后台慢慢的消息队列慢慢的做完。另外新浪发表了一个很重要的产品叫做MemcacheQ，我们去年做了一个对大规模部署非常有利的指令，就是stats queue，适合大规模运维。

　　第二版我们做了这些改进之后，微博的用户和访问量并没有停止，还有很多新的问题出现。比如说系统问题，单点故障导致的雪崩，第二个是访问速度问题因为国内网络环境复杂，会有用户反映说在不同地区访问图片、js这些速度会有问题。另外一个是数据压力以及峰值，MySql复制延迟、慢查询，另外就是热门事件，比如说世界杯，可能会导致用户每秒发表的内容达到几百条。我们考虑如何改进，首先系统方面循序任意模块失败。另外静态内容，第一步我们用CDN来加速，另外数据的压力以及峰值，我们需要将数据、功能、部署尽可能的拆分，然后提前进行容量规划。

　　另一方面我们还有平台化的需求，去年11月我们就说要做开放平台，开放平台的需求是有差异的，Web系统它有用户行为才有请求，但是API系统特别是客户端的应用，只要用户一开机就会有请求，直到他关闭电脑这种请求一直会不间断的过来，另外用户行为很难预测。

　　系统规模在持续的增大，另外也有平台化的需求，我们新架构应该怎么做才能满足这些需要？我们看一下同行，比如说Google怎么样考虑这个问题的？Google首席科学家讲过一句话，就是一个大的复杂的系统，应该要分解成很多小的服务（me:map reduce思想，分而治之思想）。比如说我们在Google.com执行一个搜索查询的话，实际上这个操作会调动内部一百多个服务。因此，我们第三版的考虑就是先有服务才有接口最后才有应用（me：这也是先有根基，然后才有建筑），我们才能把这个系统做大。

　　现在我们看一下第三版，首先我们把底层的东西分成基础服务，基础服务里面比如说分布式的存储，还有分层，我们做了一些去中心化、自动化的操作。在基础服务之上有平台服务，我们把微博常用的应用做成各种小的服务。然后我们还有应用服务，这个是专门考虑平台各种应用的需求。最上面我们有API，API就是新浪微博各种第三方应用都在上面跑。

　　平台服务和应用服务是分开的，这样实现了模块隔离，即使应用服务访问量过大的话，平台服务不会首先影响。另外我们把微博的引擎进行了改进，实现了一个分层关系。用户的关注关系，我们改成一个多维度的索引结构，性能极大的提高。第四个层面就是计数器的改进，新版我们改成了基于偏移的思路，就是一个用户他原来读的一个ID比如说是10000，系统最新的ID是10002的话，我们很清楚他有两条未读。原来的版本是采用绝对计数的，这个用户有几条未读都是用一个存储结构的话，就容易产生一致性的问题，采用这种偏移的技术基本上不会出错。

　　另外基础服务DB冷热分离多维度拆分，在微博里面我们是按照时间拆分的，但是一个大型的系统里面有很多业务需要有不同的考虑。比如说私信这个就不能按照时间来拆分，这个按照UID来拆分可能更简单。然后我们突出存储还做了一个去中心化，就是用户上传图片的速度会极大的提高，另外察看其他用户的图片速度也会极大的提高。另外是动态内容支持多IDC同时更新，这个是在国内比较新颖的。

　　下面给大家介绍一下新浪微博怎么样打造一个高性能架构。到目前为止有五千万用户使用新浪微博，最高发表3000条以上每秒，然后一个明星用户发表的话，会被几百万用户同时读到。这些问题的本质是我们架构需要考虑高访问量、海量数据的情况下三个问题。易于扩展、低延迟、高可用和异地分布。我们每天有数十亿次外部网页以及API接口的需求，我们知道微博的特点是用户请求是无法cache的。因此面对这个需求我们怎么样扩展？几点思路。第一我们的模块设计上要去状态，我们任意一个单元可以支持任意节点。另外是去中心化，避免单点及瓶颈。另外是可线性扩展。最后一个是减少模块。

　　我们要做一个高性能的系统，要具备一个低延迟、高实时性，微博要做到高实时性这是核心的价值，实时性的核心就是让数据离CPU最近（me：就是使得数据离用户最近），避免磁盘的IO。我们看淘宝核心系统专家余锋说过的一句话“CPU访问L1就像从书桌拿一本书，L2是从书架拿一本书，L3是从客厅桌子上拿一本书，访问主存就像骑车去社区图书馆拿一书”。我们微博如果要做到非常实时的话，我们就需要把数据尽量离CPU节点最近。所以我们看一下cache设计里面怎么达到这个目标。首先INBOX，这个数据我们需要放在一个最快的地方，因为用户随时访问。OutBOX里面的最近发表就是L1cache，还有一个是中期的，这个因为访问少一点，它可以被踢。最后一部分内容体有三部分。L0是本地的，我们需要把一些经常访问的，比如说明星发表微博的内容体本地化，因为它被访问的概率非常大。然后L1里面存放着最近发表的，还有一个是中期的。我们通常用L2就可以了，L1我们可以理解成它就是一个存储。

　　一个好的架构还需要具备高可用性。我们看一下业界的指标，S3是99.9%，EC2是99.5%，我们另外一个同行Face book在这方面它是没有承诺的，就是接口可用写。微博平台目前承诺的是99.95%，就是说一天365天故障率应该小于9个小时。这个怎么达到？第一我们要做容量规划，第二个是要做好监控以及入口的管理，就是说有些服务如果访问量过了的话，我们要有一个开关可以拦住他（me：消息队列）。我们通过这个图表可以清楚的看到，比如说我们要做L1的cache，我们剩余空间有多少，比如说80%，就说明这个数据有可能会丢失，有可能会对我们的系统造成影响。

　　另外一个层面就是接口监控，我们目前有多个维度的接口监控，包括访问错误失败率。然后要做架构，给大家一个很重要的经验分享，就是说监控的指标尽量量化（me:什么都最好量化，google的任何工作都需要量化的分析）。比如说他延迟30秒是小问题，如果是延迟10分钟我们就要立即采取措施了，就是所有可以量化的指标都要量化。

　　然后我们看监控怎么样更好的做？我们看亚马逊的VP说过的一句话，就是说监控系统确实特别好，可以立即告诉我们哪里有故障，但是有20%的概率我们人是会出错的。所以我们一个大型系统就应该要为自动化设计，就是说尽可能的将一些运作自动化。比如说发布安装、服务、启用、停止。我们再看另外一句，Google的工程师是怎么做的。他是这么做的，比如说第一周是处理线上的业务，这一周他处理了很多事情，处理了很多系统的情况，剩下的系统问题是不需要他做的，他只要把这一周碰到的情况用程序的方法来解决，下次再碰到这种情况很简单的一个按钮就可以处理了。我们目前也在向自动化这方面努力，就是我们的工具在持续增加。

　　另外一个异地分布，在国内网络环境下，比如说IDC灾难（me：容灾措施，多机部署，数据冗余），机房检修甚至是机房掉电，我们也碰到过中国最好的机房也会掉电，所以要每个服务单元都能支持多机房部署。另外做多机房部署有一个好处，就是用户的访问速度会提高。多IDC分布静态内容就不说了，基本上大的互联网公司都会做，它非常成熟基本上没有什么问题，比如说图片等等的静态内容。动态内容的CDN分布是业内的难点，国内很少有公司能够做到非常成熟的多机房动态内容发布的成熟方案，它的核心就是分布式存储（me：分布式存储可不止是存储的事情，它涉及存储-应用的方方面面，包括分布式缓存、分布式诗数据库、分布式session、分布式文件系统、分布式RPC和分布式消息中间件等）。一款理想的分布式存储产品它有哪些需求呢？首先它要支持海量规模（me：通过可扩展实现）、可扩展（me：数据库、文件系统分布式）、高性能（me:服务的负载均衡，读写分离）、低延迟（me：分时缓存）、高可用（me：通过主从，master-slave，热备等来实现）。第二个是需要多机房分布，能够满足国内复杂的网络环境，还要具备异地容灾能力。第三个就是要调用简单，具备丰富数据特性。因此分布式存储需要解决一个多对多的数据复制。

　　如果要做复制无非是三种策略，第一个是Master/Slave，但是它也两个缺点，第一个是Master是中心化的，如果Master在北京那广州访问就非常慢。第二个缺点是有单点风险的，比如说Master在北京，能立即迁到广州吗？这样时间窗口就丢失了，而且需要人工的干预，而且日常广州的用户访问北京的Master是有很大问题的，所以一般来说要做的非常优秀是不会考虑第一种方案的。第二种就是Multi-Master方案，它需要应用避免冲突，就是我们不能多处改变。这个对于微博来说不会特别难，我们的用户通常只会在一个地方发表微博，很难既在广州又在北京发表或者是修改自己的资料，这样的话我们应用上就可以避免这种情况。第三个就是Paxos就是可以达到强一致写，就是一条数据如果成功肯定是多个机房都成功了，这个也显而易见就是延迟性非常大。因此总结一下Multi-Master是最成熟的策略，但是它现在没有成熟的产品，因为确实没有。

　　我们再来看微博的方案，所以我们自己实现了一个多机房同步的方案。就是我们前端应用将数据写到数据库，再通过一个消息代理，相当于通过我们自己开发的一个技术，将数据广播到多个机房（me：异步数据同步）。这个不但可以做到两个机房，而且可以做到三个、四个（me：可扩展）。具体的方式就是通过消息广播方式将数据多点分布，就是说我们的数据提交给一个代理，这个代理帮我们把这些数据同步到多个机房，那我们应用不需要关心这个数据是怎么样同步过去的。

　　用这种消息代理方式有什么好处呢？可以看一下Yahoo是怎么来做的？第一个是数据提供之后没有写到db之后是不会消失的，我只要把数据提交成功就可以了，不需要关心数据怎么到达机房。第二个特点YMB是一款消息代理的产品，但是它唯一神奇的地方是为广域网设计的，它可以把多机房应用归到内部，我们应用不需要关注这个问题。这个原理跟我们目前自己开发的技术相似。

　　然后我们再看一下目前即将推出的微博平台的新架构。我们知道API大部分的请求都为了获取最新的数据。API请求有一个特点，它目前调用都是空返回的，比如说一款手机的客户端每隔一分钟它都要调用服务器一下，就是有没有新数据（me：所以说web和app不同，但是web也可以通过定时循环请求以保持连接），目前的调用都是空返回，就是说不管服务器有没有数据都要调用一次。这次询问到下一次询问中间，如果有新的数据来了，你是不会马上知道的。因此我们想API能不能改用推的方式，就是客户端不需要持续的调用，如果有新数据就会推过去。技术特点，显而易见低延迟，就是从发表到接受1秒内完成，实际上可能用不了1秒。然后服务端的连接就是高并发长连接服务，就是多点都连接在我们的服务器上，这个比传统的API要大很多。

　　我们看一下推送架构怎么从架构底层做到实时性的。从左上角的一条微博在我们系统发布之后，我们把它放在一个消息队列里面，然后会有一个消息队列的处理程序把它拿过来，处理以后放到db里面。假如说我们不做持久化，因为我们推送数据也不能丢失，我们就要写一个很复杂的程序，将S数据异步去存，这样就会非常复杂，而且系统也会有不稳定的因素。从另外一个角度来说，我们做持久化也是做过测试的。我们推送整个流程可以做到100毫秒和200毫秒之间，就是说我们在这个时间能把数据推送出去。

　　我们再看一下内部细节，就是我们收到数据之后首先要经过最上面RECEIVER。然后推到我们的引擎里面，这个引擎会做两个事情，首先会把用户的关系拿过来，然后按照用户关系马上推送给他相应的粉丝。所以我们调研方已经在那儿等待了，我们需要有一个唤醒操作，就是说在接口这儿把它唤醒，然后把它发送过去。最后是一个高并发的长连服务器，就是一台服务器支持10万以上的并发连接。最右边中间有一个圆圈叫做Stream Buffer，我们需要Stream Buffer是要保存用户最近的数据。因为用户可能会有断线的，比如说他发送数据的时候断线半分钟，我们需要把这半分钟补给他。这就是我们的推送架构。

　　下面介绍一下平台安全部分。由于我们的接口是完全开放的，所以我们要防范很多恶意行为，有很多人担心我们接口是开放的，是不是有人通过这个接口发垃圾广告，或者是刷粉丝，我们技术架构怎么来防范这一点呢？这是我们的安全架构，做了三个层面的事情。最上面是我们有一个实时处理（me：opentsdb或者storm集群实时计算），比如说根据频度、内容的相似性来进行判断，判断你发的是不是广告或者是垃圾内容。中间这个是一个处理器，我们会根据一些行为进行判断，比如说如果我们只是实时拦截的话，有些行为很难防止，我们做了个离线纠正的模块，比如说他潜伏的几个月开始发广告了，我们可以事后把这些人清除掉，以保证我们平台的健康。最后是通过监控的维度来保证内容的安全。目前内容安全的架构大概是51的体系，就是说我们的实时拦截可以做到50%的防止，离线分析大概可以做到40%的防止。

　　微博平台需要为用户提供安全及良好的体验应用，以及为开发者营造一个公平的环境，所以我们的接口需要清晰安全的规则。从一个APP调用我们的接口，需要几个阶层，需要划分不同的业务模块。第二个是安全层。第三个是权限层。这是我们平台安全的两个维度，一个接口安全，一个是内容安全。

　　我今天讲的是架构方面的问题，在座大部分是开发者，可能大家都在处理不同的架构问题，架构很多地方是相通的。我们需要做一个软件系统需要解决的本质问题是什么？微博第一版解决发布规模问题，第二版是解决数据规模的问题，第三版是解决服务化的问题。将复杂的问题简单化之后，我们才可以设计出一个容易扩展的大规模架构。我今天介绍就这么多，我们微博实际上是很需要各方面的技术人员，大家对我们的架构如果感兴趣的话、对我们的系统感兴趣的话，也希望各方面的人员参与我们微博的团队，随时可以给我微博上发私信。

=================================================================

FEED系统架构分析

张铁安：我今天跟大家分享这个内容是人人网系统架构，里面我们会讲到跟新鲜事相关的一些技术和开源一些项目，希望对大家今后工作有一些帮助。首先我要讲我们新鲜事系统在SNS的主要功能。我要在人人网发一个日志，可以很及时高效迅速的在我朋友圈、粉丝圈子里面可以看到，我朋友可以很快回复跟我进行一个很快的交互。我必须保证系统高效运转，同时要稳定。对于我们这样一个SNS网站来说，包括SNS还有微博这样一些系统，很重要一点是当发生特殊事件时会有一个爆发效应。前两天世界杯，我不是一个足球迷，那天晚上我就睡了。两点我手机不停的响，我说怎么回事，我以为同事更新服务，想了想可能今天晚上是什么比赛比较火，第二天早上说是德国队进球了。系统遇到这种事情会有一个脉冲式的爆发，去年春节晚会赵本山小品刚开始，整个系统会非常爆炸式的报警，所以对于我们系统来说我们需要解决很多的突发事件给我们带来的压力，保证我们系统有足够的稳定性。  
  
另外要说我们这个系统里面所有数据有很大一部分来自网站各个业务，还有一些来自于其他的门户网站，其他一些跟我们有合作关系的网站，开放平台支持很多第三方应用或者链接他们产生实践的时候，可以把数据发给我们FEED的系统。我们这个INPUT内容会非常复杂，我们要有很好的处理不同数据的能力。我们需要一个很好的数据规范，保证我们系统能够接受不同类型的数据。另外一个是我们输出包括几个方面，一个是登陆人人首页个人主页列表，同时还有一个PC客户端叫人人桌面还有手机客户端等等。但是对于各个不同需要展示业务要求不一样。手机展示要求我不是所有事都想要，我只想要其他一部分，会有一些选择的需求。从各个方面我们现在这个系统设计复杂度是很高的，跟各个业务连接也是非常复杂，最终导致这个系统有一个很高的复杂度。  
  
下面我想说一下我们这个系统面临一些挑战。对于人人网这样一个网站来说，活跃用户是非常多的，一天可能有几千万用户。我们计算一下，当然这个数据可能不是一个真实的数据，我们认为每秒会产生一千条Feed、一千个客户会产生一些内容，到系统里面我们要处理原始数据可能是几十亿的规模。再说一下Feed的特点，当我改一个状态我好友所有收到这些信息就是一个扩散问题，我们需要把这个数据给这些所有想要收到数据的人看到，所以这个Feed扩散范围很大。如果我有100个好友我要扩散到100人，如果我是一个明星就更多人会看到。  
  
新鲜事物有这么一个特点，我发了一篇日志就两个朋友看了觉得很有意思就把这个日志分享了，如果另外一个人是那两个人的朋友，他的页面上有两个一样内容分享，这样可能会有问题。我们会采取一种策略，把两个相关的新鲜事合并，或者做一些替换，排序，合并这些是比较复杂。另外就是用户请求量对人人网最大的请求量就是登陆的请求量。最后一点我刚才已经讲过各个业务需求要求对新鲜事做不同的筛选。  
  
然后讲一下关于系统设计当中的两个问题，推的模式和拉的模式。两个模式区别在于什么地方？推的模式意思就是说当一个事件产生的时候，我把这个事件产生时间点做N次拷贝发给他想要的人。拉是另外一种方法，当一个用户登陆页面的时候，首页要显示所有好友关注人的新鲜事。这个时候用拉的模式实现。就是说我登陆了，我查我的所有跟我有关系的列表，拿到这些列表根据这些人对应新鲜事列表里面取所有的新鲜事再做排序，归并的策略。推可能是非常快的操作，推过去以后，那边立马有了。我们登陆列表是现成，取的时候会非常快。但是有一个问题，比如说我有几个亿用户，但是活跃用户只有几千万，剩下几个亿的用户他们可能是半年来一次，或者说一个月两周过来一次。这些数据给他以后他可能根本没有机会看到，这样就浪费了很多资源。拉模式不会有这个问题，但是会有另外一个问题。你请求量很大，当用户登陆必须很快返回数据的时候，运算量是非常大的（me：就是说推是服务器注定，拉是客户端主动）。综合所有考虑，因为我们要做的是一个要求实时度很高的系统，我们还是选择推的模式，但是在用推的时候有些地方是可以做一些权衡的，把不必要系统开销可以去掉。  
  
这是我们现在Feed这个系统的各个层面。第一是新鲜事分发，就是说我发了一个东西以后，要把这个事情告诉所有跟我有关系的人，这个事就是页dispatch完成的。后面有newsFeed索引的服务，跟我们新鲜事有关的东西，包括用户的反馈，还有我们一些排序方法，跟好友关系，整个在SNS当中的朋友圈子有关系的一些东西，比如说哪些好友跟你关系很亲密，你跟你老婆关系可能很亲密跟他悄悄话我们都知道，还有一些你经常一起玩的朋友，你们这样一些人的关系可能会相对比较紧密一些。我们在考虑新鲜事排序权重（me：多维度抽取，综合排序）时我们会考虑把你老婆心情放在排序最上面，要第一时间响应领导的指示。  
  
这个是跟我们新鲜事排序相关，包括Feed排序一些算法，还有跟社会化网络相关的。我们正在做的基于新鲜事内容的一些兴趣把内容分类，有点像百度百科，我们知道哪些用户对音乐感兴趣，哪些用户对科技或者对政治感兴趣等等。这些我们会通过一些系统计算，最后反映在新鲜事排序里面。下面是MIINFeed就是自己发的新鲜事的列表，另外还有一个是新鲜事本身内容，我发了一个日志新鲜事，能够看到就是这个摘要几十个字简短的摘要。下面说的是我们新鲜事对于索引数据量是非常大的，我们会讲一下，索引数据对我们来说有什么意义。当我们用户取新鲜事需要查他的索引，以后再去取内容，这个东西内存CACHE丢失这个用户页面上什么都没有了。所以我们要做持久化。INdexdb数据会有一个列表，写到硬盘里面，最后是我们渲染引擎，我们有很多的输入和很多输出，不同输出要求不一样，比如说我们给手机输出格式和客户端格式是完全不一样。所以这两个东西都是由一个系统完成。  
  
这个是我们看到新鲜事的简单结构图。里面内容不是我们现在线上系统的整个东西，可能只是其中一部分，我是把最重要的东西拿出来。一个笑脸就是一个人在上面很开心，他发了一个日志通过我们网站日志相关的负责日志模块系统把这个日志内容发新鲜事系统里面，首先拿到就是Dis 把这个数据进行一些处理，把这个内容最终分发到三个不同的地方，第一就是newsFeed，比如说我发miniFeed有需要，第三是要把这个新鲜事产生本身内容要cache起来，会发给我们一个集群。下面会了解我们持久化这一块，MINIFeed量很小，我们做一个数据表就可以存下来。我们闪存100份，ID结尾为1放一起，这样一种散表的策略分散在机器上分担压力。我们再说一下当一个用户登陆人人网要取新鲜事的逻辑。如果是一个网站用户登陆以后设备要访问一个服务器，会并节一些新鲜事的内容，我们并没有用传统意义上的服务器，特点就是说能够支持很高的用户的并发量，同时速度会非常快。我们整个网站新鲜事的WEB服务器只有四台，会提供一个对外的PUSH的东西，也会提供一个拉模式，网站取数据就是拉的模式。这个地方做的工作其实就是用来对新鲜事的数据和模板进行匹配，然后合并产生成HTML，把数据和模板匹配在一起形成一个模块。  
  
后面是一些技术细节的东西。第一是分发系统；第二是cache；第三是持久化存储有渲染等等。  
  
整个系统我们现在涉及到Opnesource相关的，第一是ICE我们整个人人团队里面引擎这一块使用量最大的一个通讯框架，为我们提供了一个很好的cache集群，为我们很好的进行数据交互网络通信方面的一些东西。其实我们很多系统是基于这个开发的，第三是memcache，所有SNS的公司如果没用这个就不算2.0，我们也用。我们在下层应用层之间有一层代理，有点像代理服务器的感觉，实现了一些负载均衡策略等等。下面google protobuf对象的序列化及反序列化，这个东西其实可以说是非常好的，包括谷歌内部都是使用这样一个东西，我觉得非常好。下面二进制数据压缩，还有多索引结构，海量存储引擎大体就是这些东西。  
  
下面是Feed的分发系统。用户发送一个新鲜事的时候传给我们系统数据包括新鲜事数据内容，还有一些合并策略一些权重数据。一个对象是很大的，加起来可能有几K或者几百个字节大小不等。首先要做的事情是要把这个数据拆一下，拆成公共用于数据再展示使用的一些文本数据，另外还有一个就是用来做排序。怎么定位Feed这样一个索引结构？索引结构我们系统内部INDEX架构大概一个尺寸只有32个字节，CONTENT就很大了，这两个数据会分别发到不同的地方，索引数据一个给NEWSFeed另外一个给MINIFeed。我们发一条新鲜事，比如说有一个100个好友发一个日志用推模式，发一条新鲜事，我要索引结构告诉我在我好友列表里要追加这样一个新鲜事的索引，要知道我好友有哪些该把这个内容发给谁，这个操作是什么量级？一秒钟要有一千次。我查列表有一千次，有100个好友就是100。我是一个名人，有上百万粉丝这个是吃不消的。我们第一次查列表可以到数据库取，第二次就要到内存，不能到数据库上面查。最早的系统，大概一年多以前没有内存cache，说你这个东西搞得我们天天数据库是红的，我们做了以后就很好了，机器基本上没有什么负载。第三异步线程池，有的时候会有一些脉冲爆发，我们要做一些控制。当我一秒钟有一万次请求，有一个脉冲一下来了一万个请求，一次给我，我可以做一个细水长流慢慢消化掉（me：处理队列）。对用户来说朋友看到你的改的状态是在几秒钟以后，不算特别迟。但是你俩又没有在一台电脑面前，所以他感觉不到，稍微把脉冲数据做一个平滑的曲线。对于系统的负载能力有一个很好的提升，在分发里面对线程池数量是一个很重要的东西。  
  
再讲一个Feedcache的内存优化，讲设计模式的时候，叫Flygeight（me:享元模式）。当时在书里面一个例子，说我们在WPS做文本编辑，每一个文字有各种属性，字体大小等等，但是一篇文章同一个字出现N次。我们把大的数据描述数据对象在全局只有一份，我们需要使用这个字的时候，对应那个位置存一个字根，对象在一个系统里面重复出现概率很大。这样的操作对我们的帮助是很大的。我们曾经试图用这个做，但是发现在性能方面有一些问题。  
  
我们把新鲜事内容分成两种，一种是数据内容，另外一种是索引数据。索引数据相对来说比较小，我们在另外一个cache存储这样一个索引，其实从宏观上也满足flyweight理念。我们索引要发给100或者500人，他们拿到只是一个索引对象，真正指向内容都是同一件事。对于每一个索引cache内部我们也利用了同样的思路，因为比如说我们散服务，我们把前站用户放在十台机器上面，也就是说我如果有100个好友，每台机器上面平均算每一个服务上面有十个人的对于同一个新鲜事索引可能在每个服务里面出现十次，做这样一个东西我们认为一个索引结构32字节，用最小东西指向32字节又可以节约一些内存开销。

//================201711101  
  
然后我们INDEX五要支持不同业务对不同的选择条件。有同学问内存怎么建一个索引。类似一个人存数据库数据库支持什么，叫一个多索引，一个数据库表里面可以建N个不同的索引，甚至有联合索引，但是我们很少在内存里面能够实现这样一个结构。如果我们自己实现可能很复杂，对于新鲜事我要按照不同纬度建立索引怎么办？其实提供了一个数据结构，我们可以对不同的纬度做同一个索引，对对象里面同一个内容做更新，字节也会自动跟着做变动。看到下面云里面放了四个对象，形状不一样，第一是按照形状对四个对象做一个排序，第三是大小，对同一个是四个不同对象，这样类似对象能够支持不同的索引，我们使用它可以很方便实现多索引的结构。  
  
关于内存的压缩存储，可以很明显节约内存。右边图是quicklz对比图，这个压缩和解压缩速度都是非常好，使用过程中我们就使用了一种方式就是把对象进行序列化，再做压缩，在我们系统能够节约30%-35%的内存。  
  
然后讲一下我们为什么要用memcache。第一我们要支持高并发，一个用户页面显示30条新鲜事，我要进行30次，把30次我想要的对象取出来再发给前端做显示。对于人人网这么大一个应用可能每秒PV就好几万，我们需要这个东西搞定内存的cache。还有一个就是我们数据量大，大家也知道现在服务器的内存也是越来越大，原先刚到公司我们用的是16G的内存觉得已经比我们PC机大很多，再过一年，变成32、36，现在服务器搞到72G。我们要做内存的cache，对数据查询要求，随着内存里面cache内容不断增加，我们要保证查询性能不断增强。我们保证相对在我们数据量不断增加时查询性能有些下降但是不能特别大。另外一个，当我们cache不可能放在一台机器上面，当一些服务器被重启，我们需要cache量更大，要加一些机器进来。我们要保证整个cache能够有扩容性，同时可以很方便摘掉一些机器，我们需要所有cache互相之间有一些冗余。最后我希望我们cache策略、机器足够多，我们现在有十几，二十几cache服务器，当我们做到上百台，几百台机器的cache时，我们需要保证对于所有的cache服务器管理更加的方便，不是说要重新部署一次。我们是跟FACEBOOK学的，想做这样一个东西，我讲一下自己开发的东西只是MEMcache的PROXY，做这个开源项目有两个，但是这两个项目我们调研了一下不是特别理想，另外有一个动力让我们自己做这个东西的原因是因为我之前是做客户端服务器，对这种通讯等等东西还是比较有信心，另外一个就是说mbmcache协议是很简单的，所以我觉得这个东西我们有把握做好，我们就做了一下，结果做成了。  
  
基本的一个功能是什么，就是说在这个层面上我们把所有的cache的管理都放在上面，包括策略也放在上面，我们有一个cacheLOADER。我们新鲜事操作都是PV6，到数据库里面查也是ID等于什么，这样的话我做一个cacheloader可以很好跟memcache做配合，比如说我不做新鲜事，我要加东西的时候只需要在cacheloaler做一个配制。这样的话避免了开发人员重复开发一些用于加载的服务。另外一个就是为什么要有关cacheporxy，因为如果没有这些，我们跟所有散服务必须放在客户端上面，这个事情会给开发使用这个集群的人带来很多不方便，随着我们客户端不断增加，如果其他的业务不断增加，使用这个集群的人越来越多，会带来相同困扰，有这么一层我们就可以保证这个问题。  
  
下面一个进索引持久化系统。为什么要做这个东西，是因为我们在一年以前，还没有这个东西的时候，当时经常会有一些问题。新鲜事有一些大改动，要把我们索引cache重启，但是我这些cache在数据库里面是没有办法存的，因为这个量很大，我们刚才说每天如果我们产生的总的新鲜事量是千级万级以上的量，平均每个人有100个好友，其实总的一天产生新鲜事索引在几十亿规模。我们想把这些索引都存下来大概需要多少台机器？可能需要上百台机器。所以如果一秒钟处理十几万，或者几十万至少也要100台以上的机器，我们必须解决这个问题。另外我们不解决这个问题怎么做，内存索引cache没有的情况之下，我们需要把原先所有用户产生的这些新鲜事的过程从头到尾再放一次。  
  
刚才说传说中的解决方案，MYSPL是不欣，APENSOURCE还是不够快，第三就是说GFS可能解决这个问题。但是我们这个系统买不到。我们做这个的时候，我们做了一些调研，这个里面包括新浪支持，还有百度支持的。大致上我们需要在每秒钟解决十万次。第三就是我们所有的每天产生的索引数据总量每天100G以上，解决的思路是什么？第一就是普通机器我们随即读写访问就是IOPS也就能道800+的量。既然硬盘只能这样，我们怎么解决？据盘读取数据是一块一块读，我们索引很小，一个索引大小改动，我们会浪费很多写的资源，我们必须要把随即大量随即写变成一种顺序写文件，我们就要把这种所有的随机的东西变成一种顺序的问题，如果能够变成顺序的东西，我们用普通的机器可以搞定这个问题。  
  
另外一个就是说如果我们要做这个事情的话，要做一些比如说IO五方面的东西，在使用的时候用了异步IO马行，直接操作硬盘，使用的时候我们也跟英特尔做调研，选择他们SSD提高硬盘写的性能和读的性能。  
  
我们必须要把所有写合并，把大量随机变成顺序写操作，既然需要做合并，肯定我们需要先把所有的随机索引的写操作放在内存当中做一些滞后，整合以后再做写读。我们会把所有的写读操作通过LOG文件，把LOG记录下来，机器宕机我们可以通过回放把这些数据读出来，我们使用TT保存索引，为什么很快？因为他所有数据跑一遍都在内存里面，所以跟内存操作是一样。我们使用TT做了一个东西，TT支持存储方式比较简单，作数据节点上面IO模型我们选择异步IO。为什么用direct为IO屏蔽OS五的cache策略，最后使用SSD解决大量的并发读取。  
  
这个是整个系统节点，nidexNode责任存储userid到最信一块data，block的位置信息，我们把5亿用户的用户ID到索引块信息都放在内存，也用不到10G，用TT保证系统里面所有文件至少全部在内存里面放下。我们用32G机器放这个文件。另外TT实现的时候用的是共享内存实现方式。只要机器不死，节点服务被我杀掉，操作系统还在，内存还在，系统会把数据刷回硬盘。下面是DATAFILE，这就是DATAFILE的结构，左边是FILE1，右边是FILE2。  
最后讲一下模板渲染。说到数据格式的一致性，我们现在新鲜事数据格式是用Feed的输入很多来自各个不同业务，必须保证数据格式的一字形，输出时，通过渲染引擎将数据变化为不同的VIEW，提供给各业务。技术方案Ctemplate提供高效的模板选择能力，还有谷歌的方式。  
  
我今天讲了很多，大家想要有深入的了解欢迎大家加入我们的团队，谢谢。  
  
提问：刚才PPT很快过了，一些常见数据库，mebcacheDB为什么不要了？  
  
张铁安：我们对这个东西了解不够多，我们写了数据200多个动不了，我们也不知道是什么问题。我们觉得不是特别可靠，因为这个东西上了以后必须保证不出问题，出了问题必须要知道怎么解决。  
  
提问：你说光良有几百万粉丝，我们选择推模式，我们要把几百万粉丝里面每一个粉丝把信息推送到他们上面去，快速获得这些粉丝信息，粉丝信息是放在内存里面。所以我想了解，如果是我这几百个放在内存是一个方式，但是几百个粉丝是怎么组织的？  
  
张铁安：就是一个列表，我们发送会有一个表达式，我们放的时候不是说把所有放在里面，我们其实只做了一个几万的队列。为什么这么做？有这么几个目的，我们只对好友新鲜事，对于粉丝有几百万，这个列表实时有人加进来，这种情况下没有办法做一个像好友准确策略。我们做一个队列，不要太长，我做一个几万，用户登陆行为是这样，上人人网以后在这个网站玩就是几十分钟就会关掉，关掉以后cache就没有多大意义，所以逐出数据比较多。  
  
提问：你说现在我把用户放到内存里面，查询索引是通过ID查。我们数据库里面是通过ID把这个ID和索引放到cache里面。  
  
张铁安：我们两个内存cache是用ICE做的。  
  
提问：就是说这里面有一个像硬查询，很多里面都是ID，把这个硬查询也拆借出来。  
  
张铁安：我们memcache机器再多也不可能把所有新鲜事放到里面，我们现在要取一千个列表，会有不到一千个列表MIS有一个长尾效应，大部分热点数据要进行cache，对时间特别长的以前数据是不需要cache的。