　　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。第一颁的技术细节，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，我们可以由三种部署方式。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服务器，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惟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。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，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。四

　　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，我们改成一个多惟度的索引结构，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，系统最系的ID是10002的话，我们和清楚他有两条未读。原来的版本是采用绝对技术的，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，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再一个最快的地方，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举行高可用性。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，地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。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Google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。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，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。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、可扩展、高性能、低延迟、高可用。第二个是需要多机房分布，能够满足国内负责的网络环境，还要具备异地容灾能力。第三个就是要调用简单，具备丰富数据库特性。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再一个地方发表微博，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。这个不但可以做到两个机房，而且可以做到三个、四个。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，它大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，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。总上面是我们有一个实时处理，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。