主题： INTO100沙龙
时间： 2015年11月21日下午
地点： 梦想加联合办公空间

主持人： 大家下午好。感谢大家支持今天的沙龙，也感谢梦想加给我们提供场地。麦思博是一家软件研发培训公司，2007年成立，我们一直专注于软件研发快速成长，今天Into100沙龙是第14期了，我们每次主题都偏向技术，每次分享是由三个嘉宾分享。这个模式实际上跟全球软件案例峰会是一致的，用50分钟时间给你解读长尾价值。今天第14期主题是千万级规模高性能，高并发的网络架构。这次请了了三位技术大咖，第一位分享嘉宾来自，前新浪微博架构师，现任三好网CTO，他为我们分享亿级用户下的高并发的网络架构。

主题演讲：

大家下午好，我中午从公司过来的时候，外面一直在下雪，在这儿看到是一个又温馨又特别的会场，大家离我的距离非常近，所以很感谢麦思博给我这个机会与大家分享探讨。

我是卫向军，毕业后在微软工作五年，接着去了金山云，做的金山快盘，然后去了新浪微博做平台架构师，现在在三好网，做在线教育的创业公司。

第一，我会介绍一下架构以及我理解中架构的本质；第二，介绍一下新浪微博整体架构是什么样的；第三，在大型网站的系统架构是如何演变的；第四，微博的技术挑战和正交分解法解析架构；第五，微博多级双机房缓存架构；第六，分布式服务追踪系统；第七，今天分享的内容做一下总结。

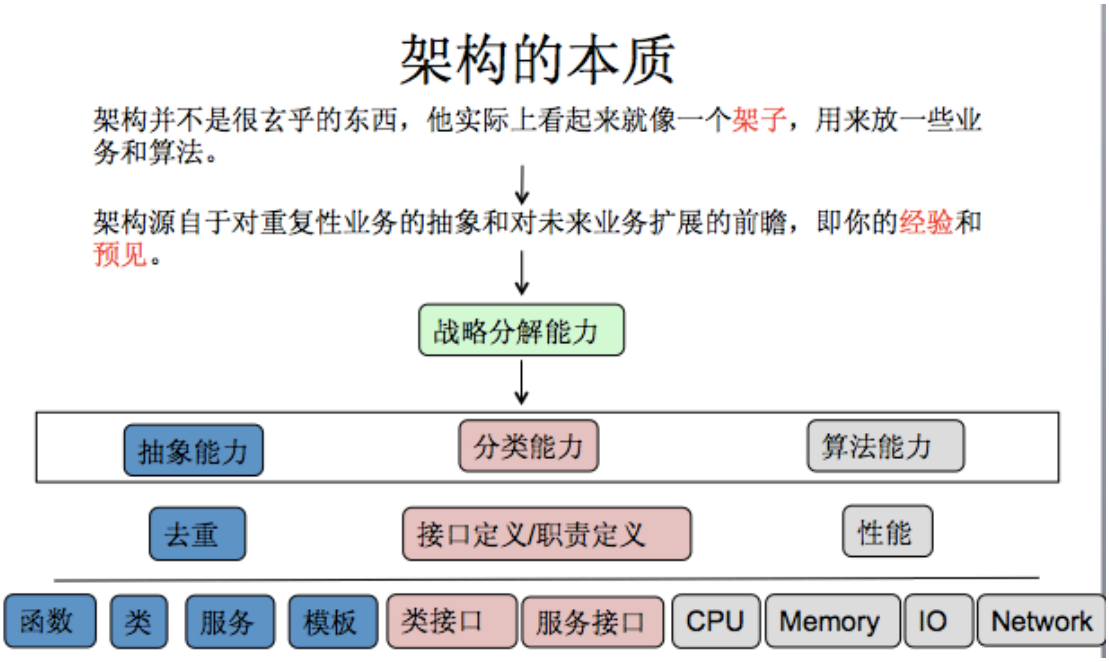
在开始谈我对架构本质的理解之前，先谈谈对今天技术沙龙主题的个人见解，千万级规模的网站感觉数量级是

非常大的，对这个数量级我们**战略上要重视它，战术上又要藐视它**。先举个例子感受一下千万级到底是什么数量级？现在很流行的优步(Uber)，从媒体公布的信息看，它每天接单量平均在百万左右， 假如每天有10个小时的服务时间，平均QPS只有30左右。对于一个后台服务器，单机的平均QPS可以到达800-1000，单独看写的业务量很简单 。为什么我们又不能说轻视它？第一，我们看它的数据存储，每天一百万的话，一年数据量的规模是多少？其次，刚才说的订单量，每一个订单要推送给附近的司机、司机要并发抢单，后面业务场景的访问量往往是前者的上百倍，轻松就超过上亿级别了。

今天我想从架构的本质谈起之后，希望大家理解在做一些建构设计的时候，它的出发点以及它解决的问题是什么。

架构，刚开始的解释是我从知乎上看到的。什么是架构？有人讲，**说架构并不是一个很悬乎的东西，实际上就是一个架子，放一些业务和算法**，跟我们的生活中的晾衣架很像。更抽象一点，说**架构其实是对我们重复性业务的抽象和我们未来业务拓展的前瞻**，强调过去的经验和你对整个行业的预见。

我们要想做一个架构的话需要哪些能力？我觉得最重要的是架构师一个最重要的能力就是你要有**战略分解能力**。这个怎么来看呢，第一，你必须要有**抽象的能力**，抽象的能力最基本就是去重，去重在整个架构中体现在方方面面，从定义一个函数，到定义一个类，到提供的一个服务，以及模板，背后都是要去重提高可复用率。第二，**分类能力**。做软件需要做对象的解耦，要定义对象的属性和方法，做分布式系统的时候要做服务的拆分和模块化，要定义服务的接口和规范。 第三，**算法（性能）**，它的价值体现在提升系统的性能，所有性能的提升，最终都会落到CPU，内存，IO和网络这4大块上。



这一页PPT举了一些例子来更深入的理解常见技术背后的架构理念。第一个例子，在分布式系统我们会做**MySQL分库分表**，我们要从不同的库和表中读取数据，这样的抽象最直观就是使用模板，因为绝大多数SQL语义是相同的，除了路由到哪个库哪个表，如果不使用Proxy中间件，模板就是性价比最高的方法。第二看一下加速网络的CDN，它是做速度方面的性能提升，刚才我们也提到从CPU、内存、IO、网络四个方面来考虑，CDN本质上一个是做网络智能调度优化，另一个是多级缓存优化。第三个看一下服务化，刚才已经提到了，各个大网站转型过程中一定会做服务化，其实它就是做抽象和做服务的拆分。第四个看一下消息队列，本质上还是做分类，只不过不是两个边际清晰的类，而是把两个边际不清晰的子系统通过队列解构并且异步化。

架构的本质

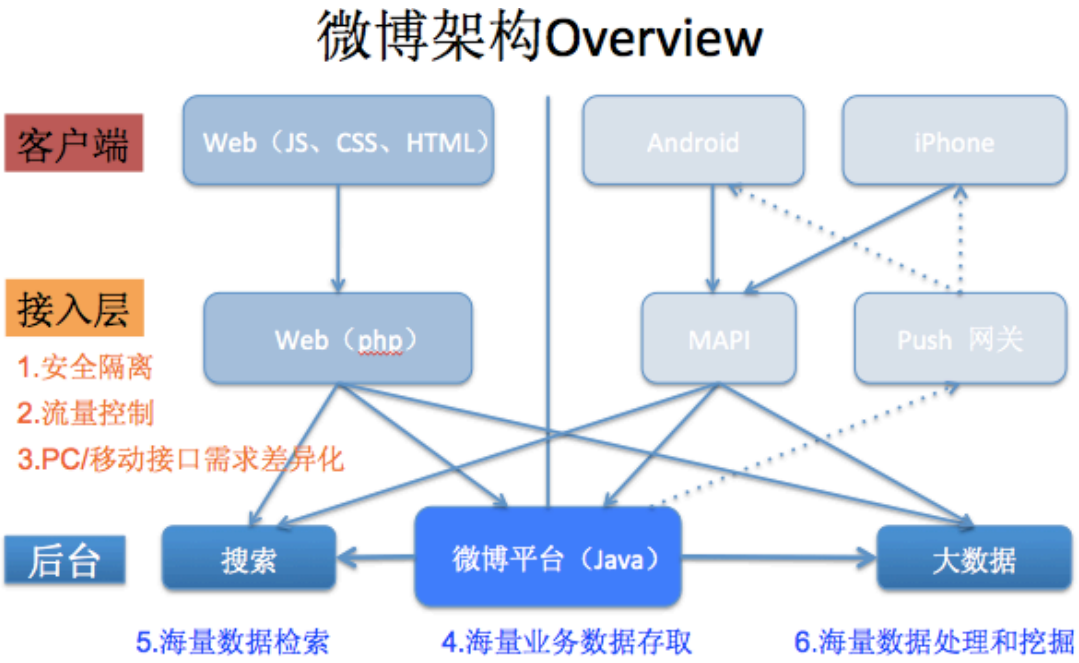
MySQL分库分表

CDN技术

服务化

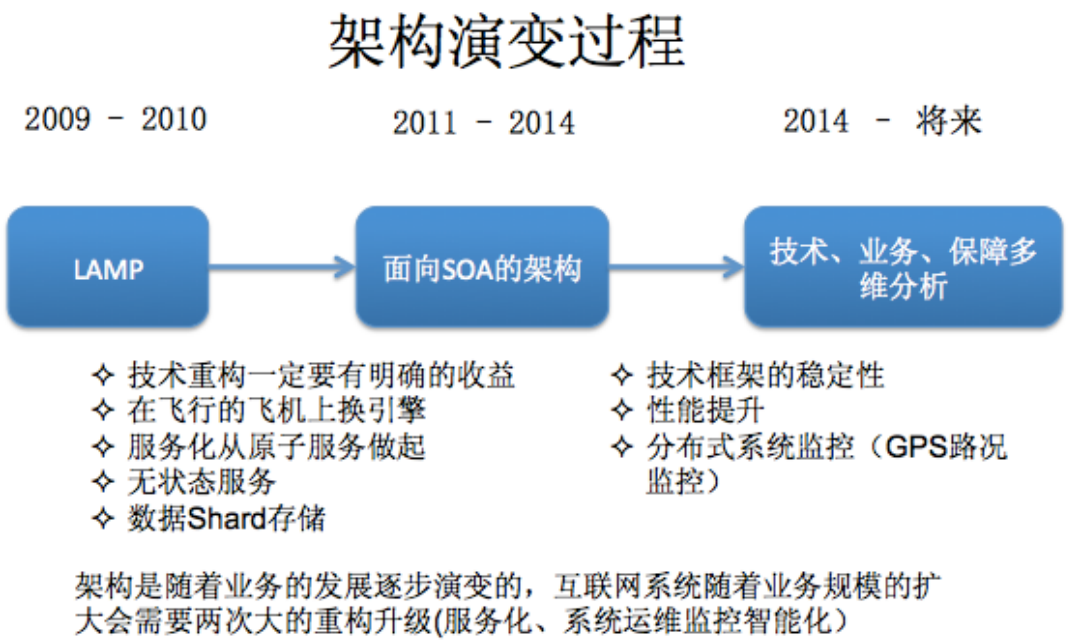
消息队列

接下我们看一下微博整体架构，到一定量级的系统整个架构都会变成三层，客户端包括WEB、安卓和IOS，这里就不说了。接着还都会有一个接口层， 有三个主要作用：第一个作用，要做**安全隔离**，因为前端节点都是直接和用户交互，需要防范各种恶意攻击；第二个还充当着一个**流量控制**的作用，大家知道，在2014年春节的时候，微信红包，每分钟8亿多次的请求，其实真正到它后台的请求量，只有十万左右的数量级（这里的数据可能不准），剩余的流量在接口层就被挡住了；第三，我们看对**PC端和移动端的需求**不一样的，所以我们可以进行拆分。接口层之后是后台，可以看到微博后台有三大块：一个是**平台服务**，第二，**搜索**，第三，**大数据**。到了后台的各种服务其实都是处理的数据。 像平台的业务部门，做的就是**数据存储和读取**，对搜索来说做的是**数据的检索**，对大数据来说是做的**数据的挖掘**。



微博其实和淘宝是很类似的。一般来说，第一代架构，基本上能支撑到用户到**百万级别**，到第二代架构基本能支撑到**千万级别**都没什么问题，当业务规模到**亿级别**时，需要第三代的架构。

从**LAMP的架构到面向服务的架构**，有几个地方是非常难的，首先不可能在第一代基础上通过简单的修修补补满足用户量快速增长的，同时线上业务又不能停， 这是我们常说的**在飞机上换引擎的问题**。前两天我有一个朋友问我，说他在内部推行服务化的时候，把一个模块服务化做完了，其他部门就是不接。我建议在做服务化的时候，首先更多是偏向业务的梳理，同时要找准一个很好的切入点，既有架构和服务化上的提升，业务方也要有收益，比如提升性能或者降低维护成本同时升级过程要平滑，建议开始从原子化服务切入，比如基础的用户服务， 基础的短消息服务，基础的推送服务。 第二，就是可**以做无状态服务**，后面会详细讲，还有数据量大了后需要做数据Sharding，后面会将。**第三代架构要解决的问题**，就是用户量和业务趋于稳步增加（相对爆发期的指数级增长），更多考虑技术框架的稳定性， 提升系统整体的性能，降低成本，还有对整个系统监控的完善和升级。



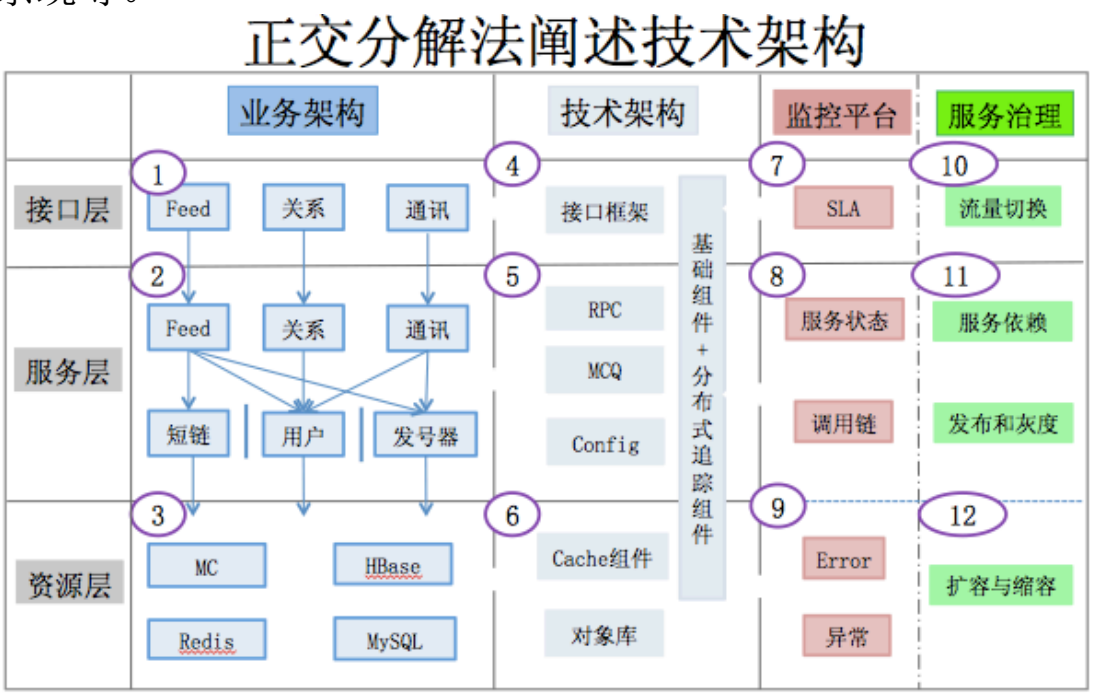
我们通过通过数据看一下它的挑战，PV是在10亿级别，QPS在百万，数据量在千亿级别。我们可用性，就是SLA要求4个9，接口响应最多不能超过150毫秒，线上所有的故障必须得在5分钟内解决完。如果说5分钟没处理呢？那会影响你年终的绩效考核。2015年微博DAU已经过亿。我们系统有上百个微服务，每周会有两次的常规上线和不限次数的紧急上线。我们的挑战都一样，就是数据量，bigger and bigger，用户体验是faster and faster，业务是more and more。互联网业务更多是产品体验驱动，**技术在产品体验上最有效的贡献，就是你的性能越来越好。每次降低加载一个页面的时间，都可以间接的降低这个页面上用户的流失率。**

下面看一下**第三代的架构图以及我们怎么用正交分解法阐述。** 我们可以看到我们从两个维度，横轴和纵轴可以看到。**一个维度是水平的分层拆分，**第二从**垂直的维度会做拆分。**水平的维度从接口层、到服务层到数据存储层。垂直怎么拆分，会用业务架构、技术架构、监控平台、服务治理等等来处理。我相信到第二代的时候很多架构已经有了业务架构和技术架构的拆分。我们看一下，**接口层**有feed、用户关系、通讯接口；服务层，SOA里有基层服务、原子服务和组合服务，在微博我们只有原子服务和组合服务。原子服务不依赖于任何其他服务，组合服务由几个原子服务和自己的业务逻辑构建而成，资源层负责海量数据的存储（后面例子会详细讲）。

技术框架解决独立于业务的海量高并发场景下的技术难题，由众多的技术组件共同构建而成。在接口层，微博使用JERSY框架，帮助你做参数的解析，参数的验证，序列化和反序列化；资源层，主要是缓存、DB相关的各类组件，比如Cache组件和对象库组件。

监控平台和服务治理，完成系统服务的像素级监控，对分布式系统做提前诊断、预警以及治理。包含了SLA规则的制定、服务监控、服务调用链监控、流量监控、错误异常监控、线上灰度发布上线系统、线上扩容缩容调度

系统等。



下面我们讲一下常见的设计原则。首先是系统架构三个利器：一个，**我们RPC服务组件（这里不讲了）**，第二个，我们**消息中间件** 。消息中间件起的作用：可以把两个模块之间的交互异步化，其次可以把不均匀请求流量输出为匀速的输出流量，所以说消息中间件**异步化解耦和流量削峰的利器**。第三个是配置管理，它是**代码级灰度发布以及保障系统降级**的利器。

第二个，无状态，接口层最重要的就是无状态。我们在电商网站购物，在这个过程中很多情况下是有状态的，比如我浏览了哪些商品，为什么大家又常说接口层是无状态的，其实我们把状态从接口层剥离到了数据层。像用户在电商网站购物，选了几件商品，到了哪一步，接口无状态后，状态要么放在缓存中，要么放在数据库中，**其实它并不是没有状态，只是在这个过程中我们要把一些有状态的东西抽离出来到了数据层**。

第三个，数据层比服务层更需要设计，这是一条非常重要的经验。对于服务层来说，可以拿PHP写，明天你可以拿JAVA来写，但是如果你的数据结构开始设计不合理，将来数据结构的改变会花费你数倍的代价，老的数据格式向新的数据格式迁移会让你痛不欲生，既有工作量上的，

又有数据迁移跨越的时间周期，有一些甚至需要半年以上。

第四，物理结构与逻辑结构的映射，上一张图看到两个维度切成十二个区间，每个区间代表一个技术领域，这个可以看做我们的逻辑结构。另外，不论后台还是应用层的开发团队，一般都会分几个垂直的业务组加上一个基础技术架构组，这就是从物理组织架构到逻辑的技术架构的完美的映射，精细化团队分工，有利于提高沟通协作的效率 。

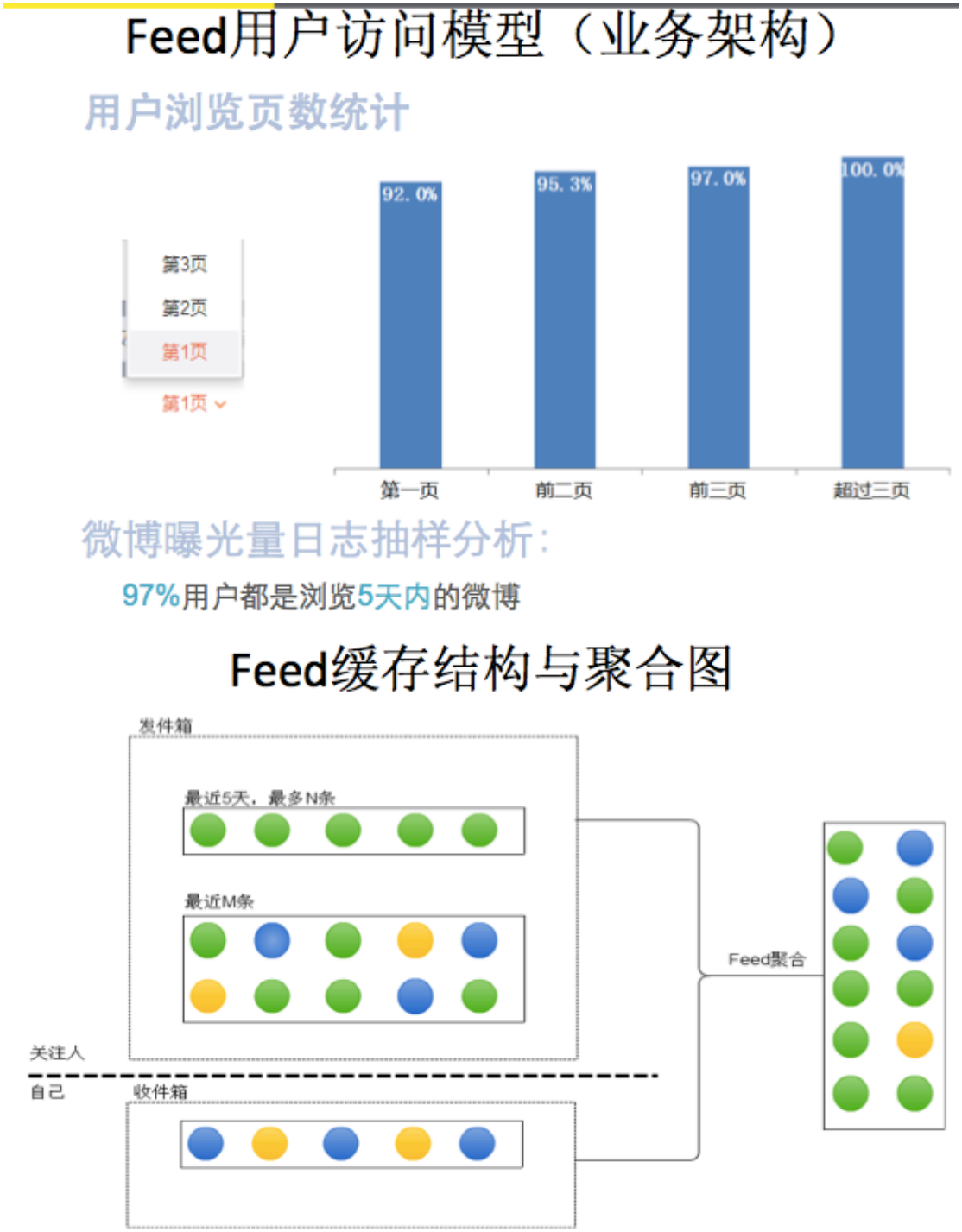
第五，www.sanhao.com的访问过程，我们这个架构图里没有涉及到的，举个例子，比如当你在浏览器输入www.sanhao网址的时候，这个请求在接口层之前发生了什么？首先会查看你本机DNS以及DNS服务，查找域名对应的IP地址，然后发送HTTP请求过去。这个请求首先会到前端的VIP地址（公网服务IP地址），VIP之后还要经过负载均衡器（Nginx服务器），之后才到你的应用接口层。在接口层之前发生了这么多事，可能有用户报一个问题的时候，你通过在接口层查日志根本发现不了问题，原因就是问题可能发生在到达接口层之前了。

第六，我们说分布式系统，它最终的瓶颈会落在哪里呢？前端时间有一个网友跟我讨论的时候，说他们的系统遇到了一个瓶颈， 查遍了CPU，内存，网络，存储，都没有问题。我说你再查一遍，因为最终你不论用上千台服务器还是上万台服务器，最终系统出瓶颈的一定会落在某一台机（可能是叶子节点也可能是核心的节点），一定落在CPU、内存、存储和网络上，最后查出来问题出在一台服务器的网卡带宽上。

设计原则

- 架构三元素：服务化组件、消息中间件、配置管理
- 无状态设计（状态去哪里了？）
- 数据层比服务层更需要精心设计(存储、压缩、索引)
- 工程师组织架构（物理）与系统组织架构（逻辑）的映射
- 架构图中缺失的地方（www.sanhao.com的访问过程？）
- 分布式系统瓶颈最终会落在：CPU、内存、存储、网络

接下来我们看一下微博的Feed多级缓存。我们做业务的时候，经常很少做业务分析，技术大会上的分享又都偏向技术架构。其实大家更多的日常工作是需要花费更多时间在业务优化上。这张图是统计微博的信息流前几页的访问比例，像前三页占了97%，在做缓存设计的时候，我们最多只存最近的M条数据。**这里强调的就是做系统设计要基于用户的场景，越细致越好。**举了一个例子，大家都会用电商，电商在双十一会做全国范围内的活动，他们做设计的时候也会考虑场景的，一个就是购物车，我曾经跟相关开发讨论过，购物车是在双十一之前用户的访问量非常大，就是不停地往里加商品。在真正到双十一那天他不会往购物车加东西了，但是他会频繁的浏览购物车。针对这个场景，活动之前重点设计优化购物车的写场景， 活动开始后优化购物车的读场景。

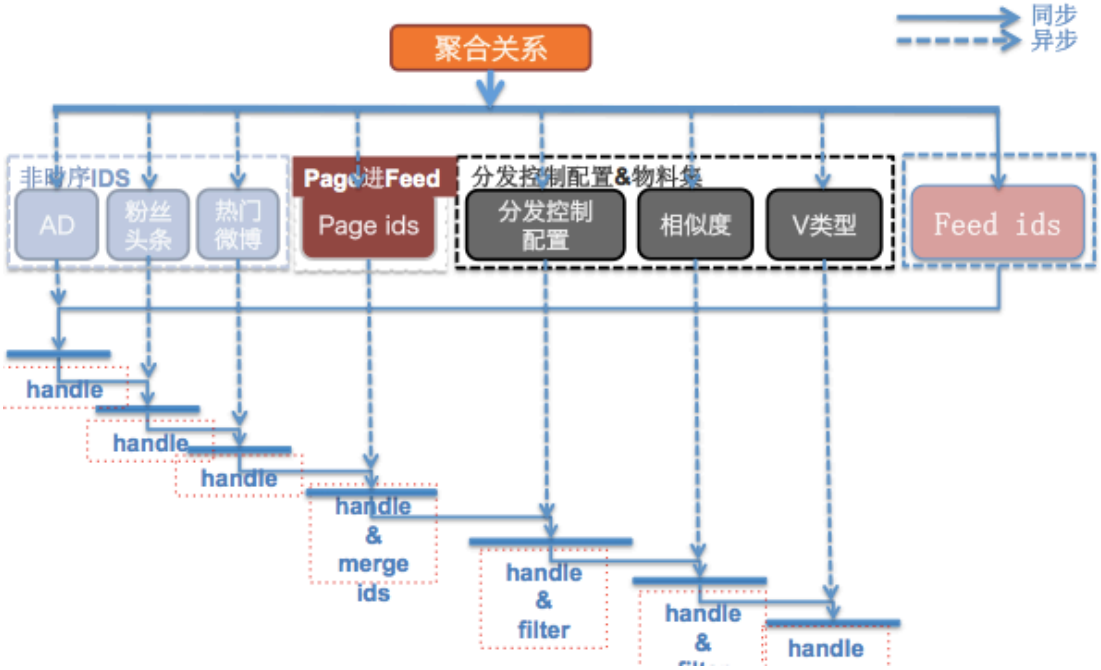


你看到的微博是由哪些部分聚合而成的呢？最右边的是Feed，就是微博所有关注的人，他们的微博所组成的。微博我们会按照时间顺序把所有关注人的顺序做一个排序。随着业务的发展，除了跟时间序相关的微博还有非时间序的微博，就是会有广告的要求，增加一些广告，还有粉丝头条，就是拿钱买的，热门微博，都会插在其中。分发控制，就是说和一些推荐相关的，我推荐一些相关的好友的

微博，我推荐一些你可能没有读过的微博，我推荐一些其他类型的微博。 当然对非时序的微博和分发控制微博，实际会起多个并行的程序来读取，最后同步做统一的聚合。

这里稍微分享一下， 从SNS社交领域来看，国内现在做的比较好的三个信息流：**1.微博是基于弱关系的媒体信息流；2.朋友圈是基于强关系的信息流；3.另外一个做的比较好的就是今日头条，它并不是基于关系来构建信息流，而是基于兴趣和相关性的个性化推荐信息流。**

信息流的聚合，体现在很多很多的产品之中，除了SNS，电商里也有信息流的聚合的影子。比如搜索一个商品后出来的列表页，它的信息流基本由几部分组成：第一，打广告的；第二个，做一些推荐，热门的商品，其次，才是关键字相关的搜索结果。**信息流开始的时候很简单，但是到后期会发现，你的这个流如何做控制分发，非常复杂， 微博在最近一两年一直在做这样的工作。**

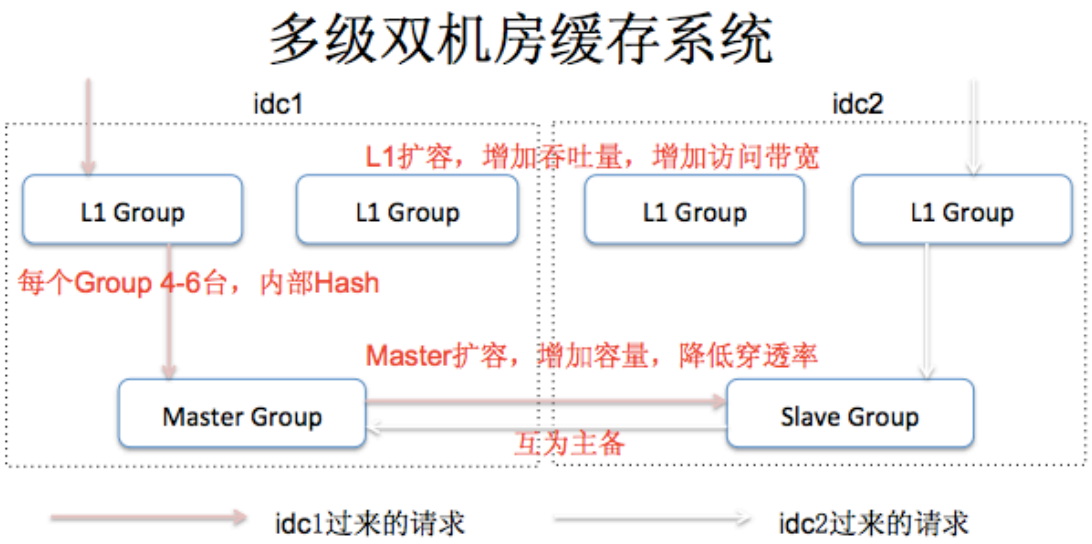


刚才我们是从业务上分析，那么技术上怎么解决高并发，高性能的问题？微博访问量很大的时候，底层存储是用MySQL数据库，当然也会有其他的。对于查询请求量大的时候，大家知道一定有缓存，可以复用可重用的计算结果。可以看到，发一条微博，我有很多粉丝，他们都会来看我

发的内容，所以**微博是最适合使用缓存的系统**，微博的读写比例基本在几十比一。

微博使用了**双层缓存**，上面是L1，每个L1上都是一组（包含4-6台机器），左边的框相当于一个机房，右边又是一个机房。在这个系统中L1缓存所起的作用是什么？**首先，L1缓存增加整个系统的QPS，其次以低成本灵活扩容的方式增加系统的带宽。**想象一个极端场景，只有一篇博文，但是它的访问量无限增长，其实我们不需要影响L2缓存，因为它的内容存储的量小，但它就是访问量大。这种场景下，你就需要使用L1来扩容提升QPS和带宽瓶颈。另外一个场景，就是L2级缓存发生作用，比如我有一千万个用户，去访问的是一百万个用户的微博，这个时候，他不只是说你的吞吐量和访问带宽，就是你要缓存的博文的内容也很多了，这个时候你要考虑缓存的容量，**第二级缓存更多的是从容量上来规划，保证请求以较小的比例穿透到后端的数据库中，**根据你的用户模型你可以估出来，到底有百分之多少的请求不能穿透到DB， 评估这个容量之后，才能更好的评估DB需要多少库，需要承担多大的访问的压力。

另外，我们看双机房的话，左边一个，右边一个。**两个机房是互为主备，或者互为热备** 。如果两个用户在不同地域，他们访问两个不同机房的时候，假设用户从IDC1过来，因为就近原理，他会访问L1，没有的话才会跑到Master，当在IDC1没找到的时候才会跑到IDC2来找。同时有用户从IDC2访问，也会有请求从L1和Master返回或者到IDC1去查找。 **IDC1和IDC2，两个机房都有全量的用户数据，同时在线提供服务，但是缓存查询又遵循最近访问原理。**



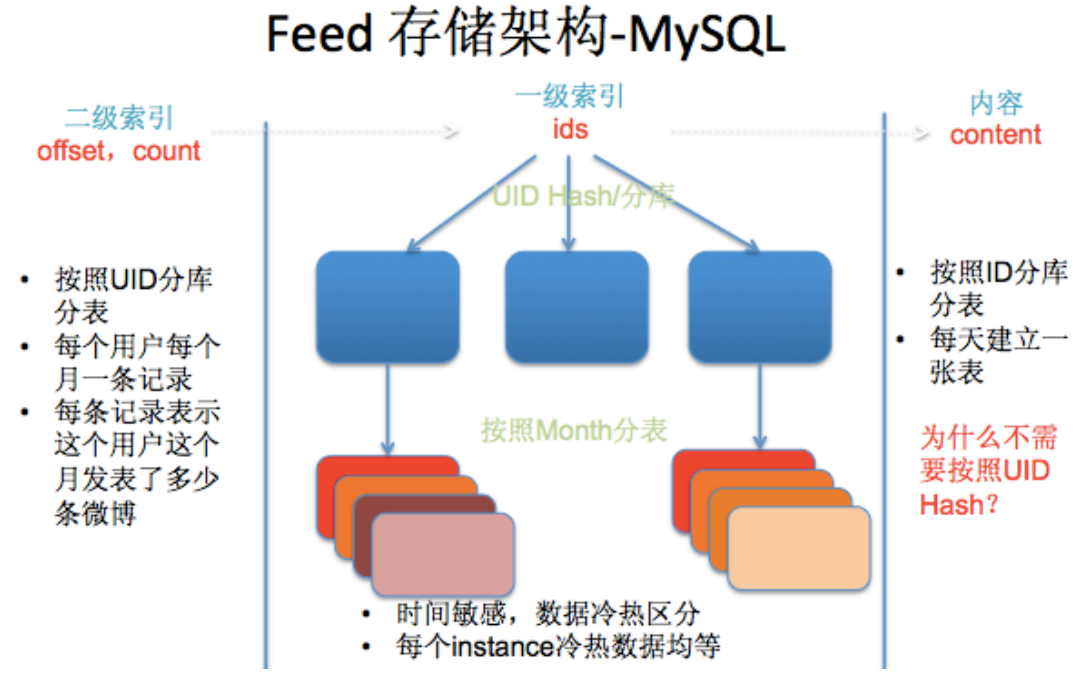
还有哪些多级缓存例子？ CDN？Local Cache+分布式缓存？CPU 缓存？

还有哪些多级缓存的例子呢？CDN是典型的多级缓存。CDN在国内各个地区做了很多节点，比如在杭州市部署一个节点时，在机房里肯定不止一台机器，那么对于一个地区来说，只有几台服务器到源站回源，其他节点都到这几台服务器回源即可，这么看CDN至少也有两级。

Local Cache+分布式缓存，这也是常见的一种策略。有一种场景，分布式缓存并不适用，**比如单点资源的爆发性峰值流量**，这个时候使用Local Cache + 分布式缓存，**Local Cache在应用服务器上用很小的内存资源挡住少量的极端峰值流量**，长尾的流量仍然访问分布式缓存，这样的Hybrid缓存架构通过复用众多的应用服务器节点，降低了系统的整体成本。

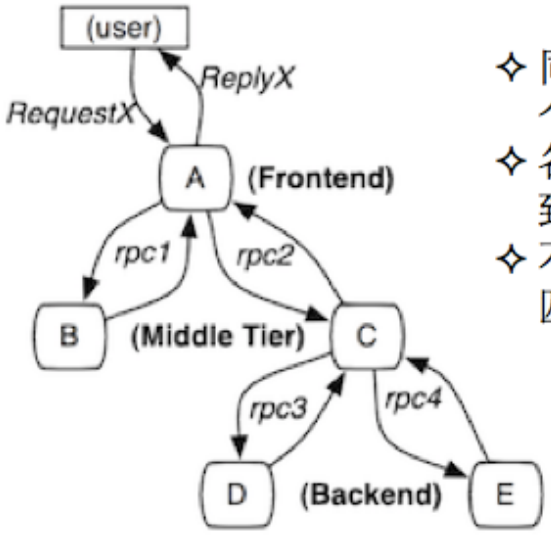
我们来看一下**Feed的存储架构**，微博的博文主要存在MySQL中。首先来看内容表，这个比较简单，每条内容一个索引，每天建一张表，其次看索引表，一共建了两级索引。首先想象一下用户场景，大部分用户刷微博的时候，看的是他关注所有人的微博，然后按时间来排序。仔细分析发现在这个场景下， 跟一个用户的自己的相关性很小了。所以在一级索引的时候会先根据关注的用户，取他们的前条微博ID，然后聚合排序。我们在做哈希（分库分表）的时候，同时考虑了按照UID哈希和按照时间维度。很业务和

时间相关性很高的，今天的热点新闻，明天就没热度了，数据的冷热非常明显，这种场景就需要按照时间维度做分表，首先冷热数据做了分离（可以对冷热数据采用不同的存储方案来降低成本），其次， 很容止控制我数据库表的爆炸。像微博如果只按照用户维度区分，那么这个用户所有数据都在一张表里，这张表就是无限增长的，时间长了查询会越来越慢。二级索引，是我们里面一个比较特殊的场景，就是我要快速找到这个人所要发布的某一时段的微博时，通过二级索引快速定位。



分布式追踪服务系统，当系统到千万级以后的时候，越来越庞杂，所解决的问题更偏向稳定性，性能和监控。刚才说用户只要有一个请求过来，你可以依赖你的服务RPC1、RPC2，你会发现RPC2又依赖RPC3、RPC4。分布式服务的时候一个痛点，就是说一个请求从用户过来之后，在后台不同的机器之间不停的调用并返回。

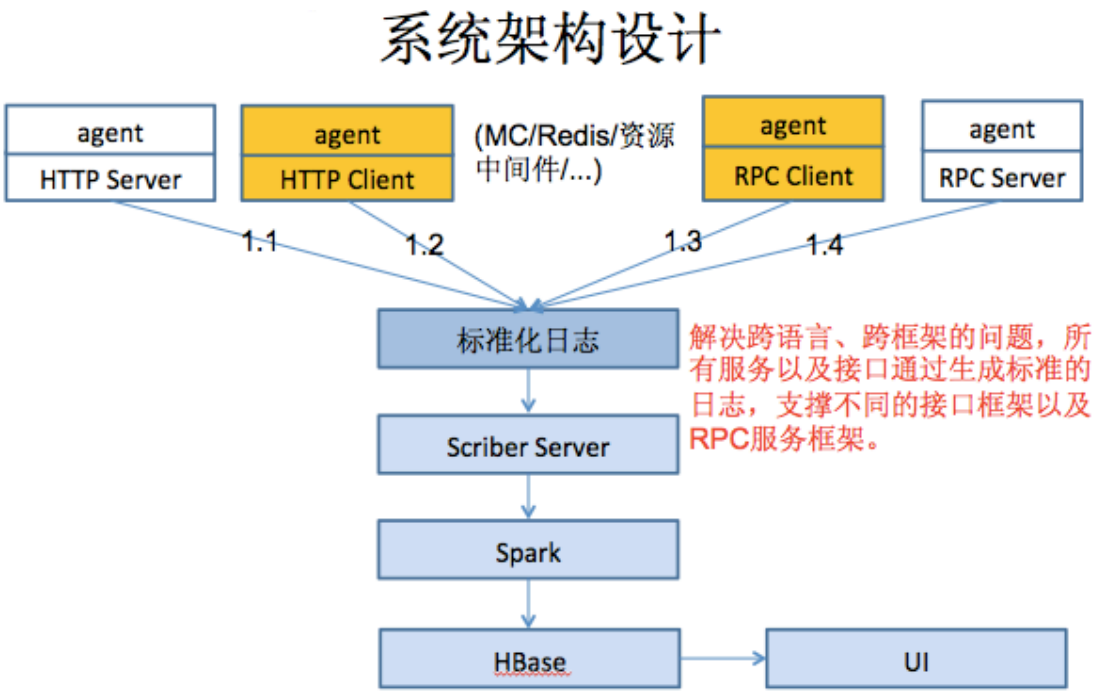
分布式服务痛点



- 同一个请求，处理时依赖多个微服务。
- 各个服务之间互相隔离，导致出问题时，排查特别困难。
- 不同服务的日志无法有效的匹配。

当你发现一个问题的时候，这些日志落在不同的机器上，你也不知道问题到底出在哪儿，各个服务之间互相隔离，互相之间没有建立关联。所以导致排查问题基本没有任何手段，就是出了问题没法儿解决。

我们要解决的问题，我们刚才说日志互相隔离，我们就要把它建立联系。建立联系我们就有一个请求ID，然后结合RPC框架， 服务治理功能。假设请求从客户端过来，其中包含一个ID 101，到服务A时仍然带有ID 101，然后调用RPC1的时候也会标识这是101，所以需要**一个唯一的请求ID标识递归迭代的传递到每一个相关节点**。第二个，你做的时候，你不能说每个地方都加，对业务系统来说需要一个框架来完成这个工作，**这个框架要对业务系统是最低侵入原则，用JAVA的话就可以用AOP**，要做到零侵入的原则，就是对所有相关的中间件打点，从接口层组件（HTTP Client、HTTP Server）至到服务层组件（RPC Client、RPC Server），还有数据访问中间件的，这样业务系统只需要少量的配置信息就可以实现全链路监控 。为什么要用日志？服务化以后，每个服务可以用不同的开发语言，**考虑多种开发语言的兼容性，内部定义标准化的日志是唯一且有效的办法**。



最后，如何构建基于GPS导航的路况监控？我们刚才讲分布式服务追踪。分布式服务追踪能解决的问题，**如果单一用户发现问题后，可以通过请求ID快速找到发生问题的节点在什么**，但是并没有解决如何发现问题。我们看现实中比较容易理解的道路监控，每辆车有GPS定位，我想看北京哪儿拥堵的时候，怎么做？ **第一个，你肯定要知道每个车在什么位置**，它走到哪儿了。其实可以说每个车上只要有一个标识，加上每一次流动的信息，就可以看到每个车流的位置和方向。**其次如何做监控和报警**，我们怎么能了解道路的流量状况和负载，并及时报警。我们要定义这条街道多宽多高，单位时间可以通行多少辆车，这就是道路的容量。有了道路容量，再有道路的实时流量，我们就可以基于实习路况做预警？

对应于分布式系统的话如何构建？ 第一，你要定义每个服务节点它的SLA是多少？ SLA可以从系统的CPU占用率、内存占用率、磁盘占用率、QPS请求数等来定义，相当于定义系统的容量。 **第二个，统计线上动态的流量**，你要知道服务的平均QPS、最低QPS和最大QPS，有了流量和容量，就可以对系统做全面的监控和报警。

刚才讲的是理论，实际情况肯定比这个复杂。微博在春节的时候做许多活动，必须保障系统稳定，理论上你只

要定义容量和流量就可以。但实际远远不行，为什么？有技术的因素，有人为的因素，因为不同的开发定义的流量和容量指标有主观性，很难全局量化标准，所以真正流量来了以后，你预先评估的系统瓶颈往往不正确。实际中我们在春节前主要采取了三个措施：第一，最简单的就是有**降级的预案**，流量超过系统容量后，先把哪些功能砍掉，需要有明确的优先级 。第二个，**线上全链路压测**，就是把现在的流量放大到我们平常流量的五倍甚至十倍（比如下线一半的服务器，缩容而不是扩容），看看系统瓶颈最先发生在哪里。我们之前有一些例子，推测系统数据库会先出现瓶颈，但是实测发现是前端的程序先遇到瓶颈。第三，**搭建在线Docker集群， 所有业务共享备用的Docker集群资源**，这样可以极大的避免每个业务都预留资源，但是实际上流量没有增长造成的浪费。

Understanding Java!
Understanding JVM!!
Understanding OS!!!
Understanding Design Pattern!
Understanding TCP/IP!!
Understanding Distributed System!!!
Understanding Data Structure and Algorithm!!!!

我所说的一切都是错的！
Learn, Practice, Summary, and build your architect!

接下来说的是如何不停的学习和提升，这里以Java语言为例，首先，**一定要理解JAVA**；第二步，JAVA完了以后，一定要**理解JVM**；其次，还要**理解操作系统**；再次还是要了解一下Design Pattern，这将告诉你怎么把过去的经验抽象沉淀供将来借鉴；还要学习TCP/IP、**分布式系统**、**数据结构和算法**。

最后就是我想说的就是今天我所说的可能一切都是错的！大家通过不停的学习、练习和总结， 形成自己的一套架构设计原则和方法，谢谢大家。