13 | 高可用架构案例（一）：如何实现O2O平台日订单500万?

2020-03-20 王庆友

架构实战案例解析

进入课程 >

讲述：王庆友

时长 14:44　大小 13.50M

你好，我是王庆友。在上一讲中，我和你介绍了高可用系统的设计原则和常见手段。今天呢，我会通过一个实际的案例，告诉你具体如何落地一个高可用的架构，让你能够深入理解和运用这些高可用手段。

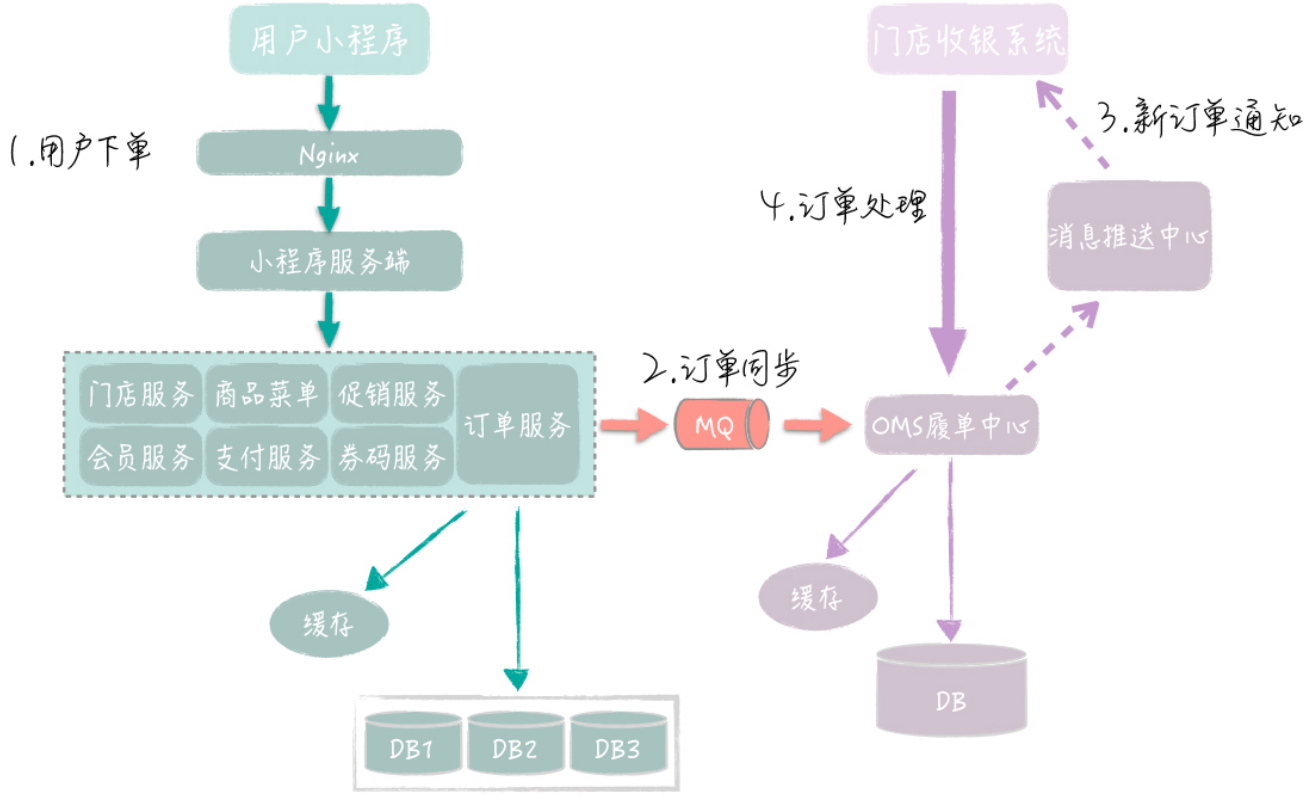
项目背景介绍

先说下项目的背景。这是一个小程序点餐平台，用户在小程序上点餐并支付完成后，订单会先落到订单库，然后进一步推送到门店的收银系统；收银系统接单后，推送给后厨系统下生产；同时返回小程序取餐码，用户可以凭取餐码去门店取餐或收取外卖。

这个项目服务于一家大型的餐饮公司，公司在全国有大量的门店，他们准备搞一个长期的大型线上促销活动，促销的力度很大：用户可以在小程序上先领取优惠券，然后凭券再支付 1 元，就可以购买价值数十元的套餐。

结合以往的经验，以及这次的促销力度，我们预计在高峰时，前端小程序请求将会达到**每秒 10 万 QPS**，并且预计**首日的订单数量会超过 500 万**。在这种高并发的情况下，我们为了保证用户的体验，**系统整体的可用性要达到 99.99%**。

你可以先了解一下这个点餐平台的具体架构：



这里呢，我具体说下系统主要的调用过程，以便于你更好地理解它：

1. 小程序前端通过 Nginx 网关，访问小程序服务端；
2. 小程序服务端会调用一系列的基础服务，完成相应的请求处理，包括门店服务、会员服务、商品服务、订单服务、支付服务等，每个服务都有自己独立的数据库和 Redis 缓存；
3. 订单服务接收到新订单后，先在本地数据库落地订单，然后通过 MQ 同步订单给 OMS 履单中心；

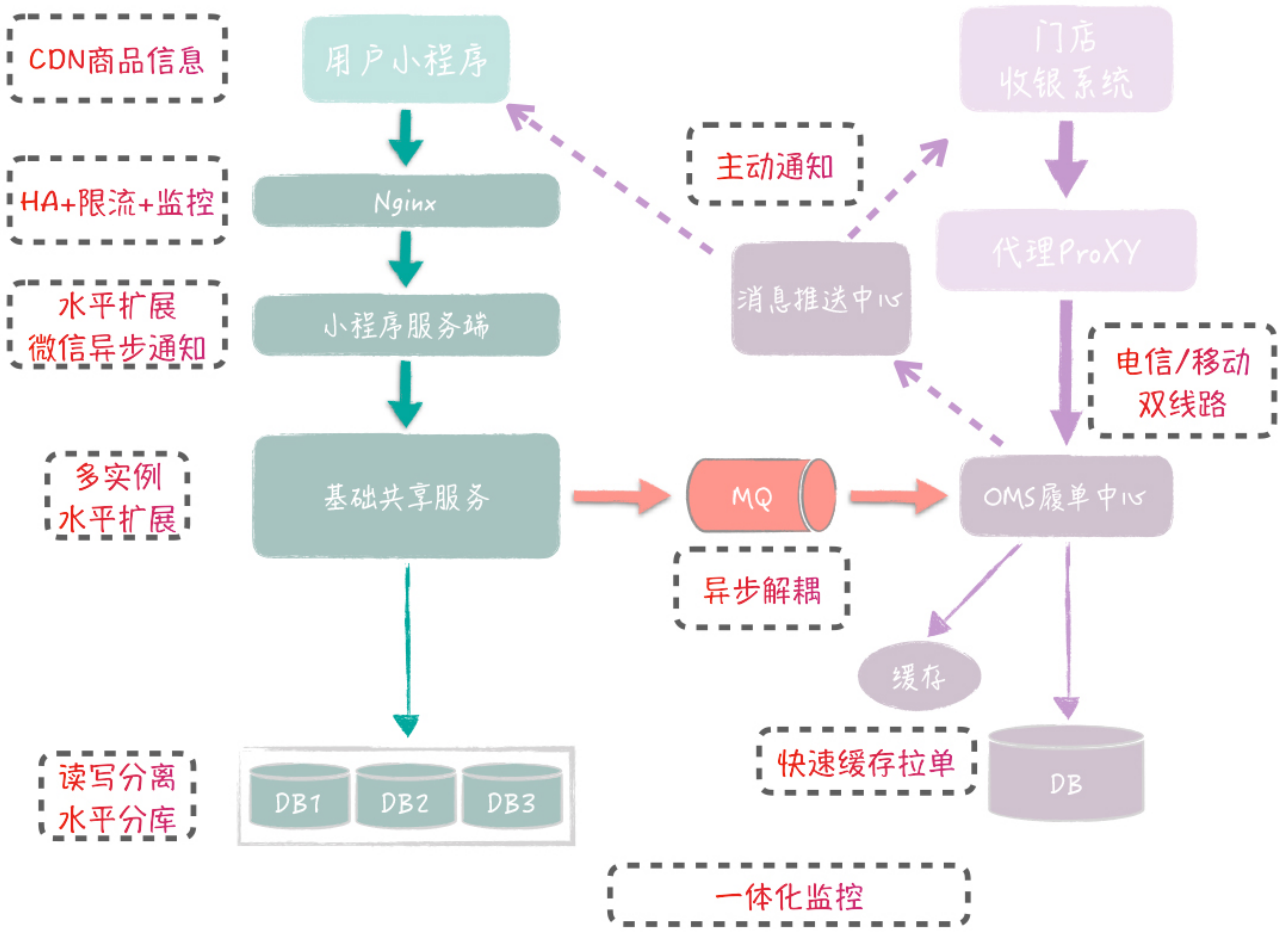
4. 门店的收银系统通过 HTTP 远程访问云端的 OMS 履单中心，拉取新订单，并返回取餐码给 OMS，OMS 再调用小程序订单服务同步取餐码；
5. 小程序前端刷新页面，访问服务端获得取餐码，然后用户可以根据取餐码到门店取餐或等待外卖。

高可用系统改造措施

我在前面也介绍了，这次活动的促销力度很大，高峰期流量将达到平时的数十倍，这就要求系统能够在高并发的场景下，保证高可用性。

所以，基于访问量、日订单量和可用性的指标，我们对原有系统进行了一系列改造，最终顺利地实现了首日 500 万订单，以及在大促期间，系统 4 个 9 的可用性目标。这个 500 万的订单量，也创造了中国单商户线上交易的历史记录。

在下面的系统架构图中，我标出了具体的改造点，主要有 10 处，接下来我就给你分别具体介绍一下，你可以通过这些具体的改造措施，来真正理解高可用系统的设计手段。



前端接入改造

这里的前端有两个，C 端的小程序和 B 端的门店收银系统。前端部分主要是对三个点进行改造，包括小程序端的 CDN 优化、Nginx 负载均衡，以及收银端的通信线路备份。

小程序端的 CDN 优化

用户点餐前，需要先浏览商品和菜单，这个用户请求的频率很高，数据流量大，会对服务端造成很大的压力。所以，针对这一点，我们通过 CDN 供应商，在全国各地构建了多个 CDN 中心，储存静态的商品数据，特别是图片，这样小程序前端可以就近访问 CDN，流量无需通过小程序服务端，缓解了服务端的压力。

Nginx 负载均衡

这个小程序点餐平台，之前是直接利用云服务商提供的 LB，它只有简单的负载均衡能力。为了能应对这次的高并发流量，现在我们独立搭建了数十台的 Nginx 集群，集群除了负载均衡，还提供限流支持，如果 QPS 总数超过了 10 万，前端的访问请求将会被丢弃掉。

另外，Nginx 在这里还有一个好处，就是可以实时提供每个接口的访问频率和网络带宽占用情况，能够起到很好的接入层监控功能。

> **补充说明：**一台 Nginx 一般可以支持数万的并发，本来这里无需这么多台 Nginx，这是因为云服务商对单个 LB 的接入有网络带宽的限制，所以我们要通过提升 Nginx 的数量，来保证接入有足够的带宽。

收银端的通信线路备份

门店的收银系统会通过前置代理服务器，来访问云端的 OMS 系统，这个代理服务器部署在商户自己的 IDC 机房，原来只通过电信线路和云端机房打通。在这次改造中，我们**增加了移动线路**，这样当电信主线路出问题时，系统就可以快速地切换到移动线路。

应用和服务的水平扩展

首先，针对小程序服务端的部署，我们把实例数从十几台提升到了 100 台，水平扩展它的处理能力。在上面的架构图中，你可以看到，小程序服务端依赖了 7 个基础服务，每个基础服务也做了相应的水平扩展，由于应用和基础服务都是无状态的，因此我们很容易扩充。

这里的基础服务是 Java 开发的，原来是用虚拟机方式部署的，现在我们把基础服务全部迁移到了**容器环境**，这样在提升资源利用率的同时，也更好地支持了基础服务的弹性扩容。

订单水平分库

在大促情况下，下单高峰期，订单主库的**写访问**频率很高，一个订单会对应 6~7 次的写操作，包括了创建新订单和订单状态变更；订单的**读操作**，我们之前通过一主多从部署和读写分离，已经得到了支持。

但负责写入的主库只有一个实例，所以这次我们通过**订单的水平分库**，扩充了订单主库的实例数，改造后，我们有 4 个主库来负责订单数据写入。数据库的配置，也从原来的 8 核 16G 提升到了 16 核 32G，这样我们通过硬件的垂直扩展，进一步提升了数据库的处理能力。

这里的订单水平分库在实现上比较简单，我们是**通过订单 ID 取模进行分库，基于进程内的 Sharding-JDBC 技术，实现了数据库的自动路由**。后面的课程中，我会专门介绍电商平台的订单水平分库，它会更加复杂，到时你可以做个比较，如果有需要的话，也可以在实际项目参考落地。

异步化处理

你可以看到，在前台订单中心和后台 OMS 之间，我们需要同步订单数据，所以这两者是紧密耦合的。不过这里，我们通过**消息系统**对它们进行了解耦。 一方面，前台下单要求比较快，后台 OMS 的订单处理能力比较弱（OMS 库没有进行水平分库），通过消息的异步化处理，我们实现了对订单流量的削峰；另一方面，如果 OMS 有问题，以异步的方式进行数据同步，也不会影响前台用户下单。

还有在小程序服务端，在用户支付完成或者后台生成取餐码后，我们会以**微信消息**的方式通知用户，这个在代码中，也是通过异步方式实现的，如果微信消息发送不成功，用户还是可以在小程序上看到相关信息，不影响用户取餐。

主动通知，避免轮询

在原来的架构中，前台小程序是通过轮询服务端的方式，来获取取餐码；同样，商户的收银系统也是通过轮询 OMS 系统拉取新订单，这样的收银系统有上万个，每隔 10s 就会拉取一次。这种盲目轮询的方式，不但效率低，而且会对服务端造成很大的压力。

经过改造后，我们落地了**消息推送中心**，收银系统通过 Socket 方式，和推送中心保持长连接。当 OMS 系统接收到前台的新订单后，会发送消息到消息推送中心；然后，收银系统就可以实时地获取新订单的消息，再访问 OMS 系统拉取新订单。为了避免因消息推送中心出问题（比如消息中心挂掉了），导致收银系统拿不到新订单，收银系统还保持对 OMS 系统的轮询，但频率降低到了 1 分钟一次。

同理，小程序前端会通过 Web Socket 方式，和消息推送中心保持长连接。当 OMS 系统在接收到收银系统的取餐码后，会发送消息到消息推送中心。这样，小程序前端可以及时地获取取餐码信息。

缓存的使用

我们知道，缓存是提升性能十分有效的工具。这里的改造，就有两个地方使用了缓存。

当收银系统向 OMS 拉取新订单时，OMS 不是到数据库里查询新订单，而是把新订单先保存在 Redis 队列里，OMS 通过直接查询 Redis，把新订单列表返回给收银系统。

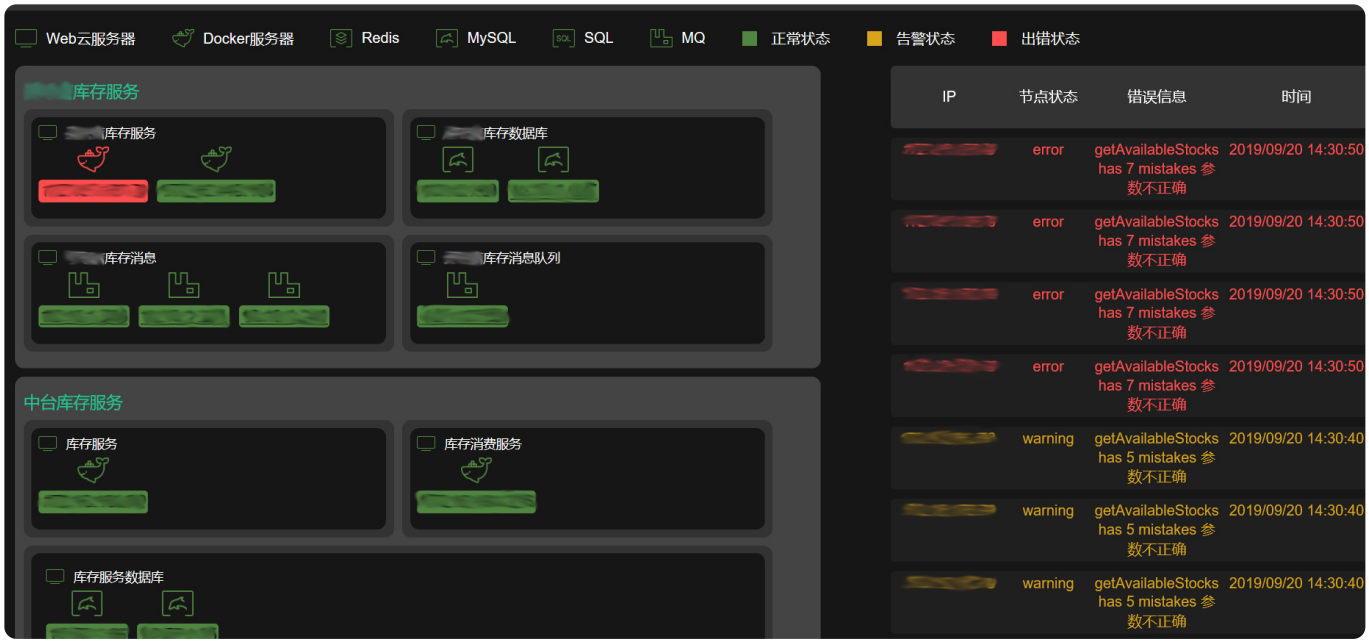
在商品服务中，菜单和商品数据也是放在了 Redis 中，每天凌晨，我们通过定时任务，模仿前端小程序，遍历访问每个商品数据，实现对缓存的预刷新，进一步保证缓存数据的一致性，也避免了缓存数据的同时失效，导致缓存雪崩。

一体化监控

在前面各个节点可用性优化的基础上，我们也在系统的监控方面做了很多强化。除了常规的 Zabbix 做系统监控、CAT 做应用监控、拉订单曲线做业务监控以外，我们还对系统实现了一体化的监控。

在这里，所有的节点都在一个页面里显示，包括 Web 应用、Redis、MQ 和数据库，页面也会体现节点之间的上下游关系。**我们通过采集节点的状态数据，实时监测每个节点的健康程度，并且用红黄绿三种颜色，表示每个节点的健康状况。**这样，我们就可以非常直观地识别出，当前的哪些节点有问题。

监控的效果如下图所示，在下一讲中，我就会为你具体地介绍这个监控系统。



在实践中，这套监控系统也确实发挥了巨大的作用。很多时候，在系统问题还没有变得严重之前，我们就能够识别出来，并能进行主动干预。

比如说，小程序服务端的部分节点有时候会假死，这在 Zabbix 监控里往往看不出来，但在我们的监控页面中，这些节点就会飘红，我们就可以通过重启节点来快速恢复。还有好几次，系统有大面积的节点出问题了，我们通过节点的上下游关系，很容易地定位出了真正出现问题的地方，避免所有人一窝蜂地扑上去排查问题。

除了这里我介绍的优化措施以外，我们也为系统可能出问题的地方做了各种预案。比如说，我们保留了部分虚拟机上部署的基础服务实例，这样如果容器出现了问题，基础服务可以快速切回到虚拟机上的实例。

系统改造小结

到这里为止，系统主要的优化措施就介绍完了，**但我们是如何知道要配置多少个节点，有没有达到预定的效果呢?**

对于这个问题，我们的做法是，**按照 10 万 QPS 和 99.99% 的可用指标要求，通过大量的压测来确定的。**

首先，我们对每个节点进行接口压测，做各种性能优化，确定好需要的机器数量；

然后，我们利用 JMeter，模拟小程序前端发起混合场景的调用，以此检验系统的抗压能力，以及在压力下，系统的可用性是否达到了预定的要求；

最后，我们在生产环境中根据压测环境，按照服务器 1:1 的数量进行部署，保证性能不打折，最终这个小程序下单平台总的机器规模，也达到了数百台的量级。

这里，我想结合着上一讲和你介绍的架构原则，来让你更深刻地理解这次系统可用性的改造过程。

从正面保障的角度来看，我们首先在各个环节都**避免了单点**，包括远程通信线路，这样能保证任意一个节点出了问题，都有其他实例可以顶上去；其次，我们通过节点的**垂直扩展和水平扩展**，大幅度提升了系统的处理能力，包括应用、服务和数据库的扩展；我们也有效地利用了 **Redis 缓存**，对高频的订单和菜单数据的读取进行了优化。

在**柔性处理**方面，我们通过异步处理，来优化系统的性能和避免大流量的直接冲击，包括使用消息系统解耦前台下单系统和后台 OMS 系统，以及通过及时的消息推送，避免前端盲目轮询服务端。

同时，我们在**系统接入层**，通过 Nginx 进行限流，为系统的可用性进行兜底，这样在流量超过预估时，能够有效地避免后端系统被冲垮。

最后，我们通过**强有力的监控手段**，可以实时全面地了解系统运行状况，随时为异常情况做好准备。

总结

今天，我与你分享了一个实际的 O2O 点餐平台，在面对高并发流量时，我们是如何对系统进行升级改造，保证系统的高可用的。相信你在上一讲理论的基础上，通过进一步结合实际的场景，能够深入地理解如何运用各种高可用的手段了。

高可用的处理方式有很多，我这里给你介绍的也只是一部分，希望你能够在实践中，结合具体的业务场景，灵活地落地高可用的设计。

不过，无论我们采取多么周密的措施，总会有些地方我们没有考虑到，系统可能会出现各种各样的问题，这个时候对系统进行全面的监控就非常重要了。下一讲我会就如何做好系统的监控，和你做详细的介绍。

最后，给你留一道思考题：你当前的系统有单点吗？这个单点有没有出过问题呢？

欢迎你在留言区与大家分享你的问题和思考，我们一起讨论。如果觉得有收获，也欢迎你把这篇文章分享给你的朋友。感谢阅读，我们下期再见。

上一篇　12 | 高可用架构：如何让你的系统不掉链子?

下一篇　14 | 高可用架构案例（二）：如何第一时间知道系统哪里有问题?

精选留言 (14)

写留言

每天晒白牙
2020-03-20

很期待老师说的监控提前预警的方案
老师我请教个和监控报警相关的问题，就是我们自己通过 prometheus+grafana 搭建了一套简单的业务监控报警，主要通过在代码中埋点，现在报警短信有点多，但里面有些报警也并不是很严重，所以就忽略了，但有时又会错过严重的报警短信，从而影响线上问题，所以后面把报警的阈值提高了，但感觉这样又有风险。...

展开

作者回复: 监控数据一般有收集，分析和告警的过程，你这里缺乏后端的监控数据的分析系统，需要结合各种规则做综合判断后才告警。

2

lyshrine
2020-03-21

请问老师，CAT的全程是什么？

展开

作者回复: Central Application Tracking，一个开源的应用监控组件，国内用的比较广泛。

1

Jeff.Smile
2020-03-20

这讲最大的价值就是让普通公司的工程师见识到了大流量的应对架构设计，有了一个总体的轮廓和方向。

1

正在减肥的胖籽。
2020-03-20

下单后，订单放到redis中，如果redis数据写入失败？有做补偿吗?需要请教老师你们redis和数据库之间的数据怎么保证一致性？如果不保持一致性那其他系统是否就拉取不到订单?

作者回复: 一种是把redis作为前置数据库，如果下单时，缓存写入失败，等于业务失败。当缓存写入后，如果缓存崩溃，这个问题不大，可以通过持久化缓存数据，重启后恢复。

如果只是把redis当做缓存来用，我比较推荐写db的时候立即更新缓存或删除缓存，保证缓存和db数据的一致性。

1 1

每天晒白牙
2020-03-20

感谢老师的实战经验分享，很干

展开

1

夜空中最亮的星（华仔...

2020-03-20

这里讲，很棒，都是硬核

展开

1

刘楠

2020-03-20

规模没这么大，只能想想，学习下

展开

1

深山小书童

2020-03-20

非常棒！一早起来读到干货满满的文章，心情美美哒

展开

1

Geek_0e5f26

2020-03-24

老师 请教两个问题。问题一：Nginx集群，他们的物理架构是怎么样的，谁来做路由和负载均衡？ 问题2：通过提升 Nginx 的数量，来保证接入有足够的带宽。这个带宽量是如何估算的？比如每个用户请求平均大小，再到响应数据平均大小，这个估算过程请您指点一下吧，最好请您分享一下参考值，谢谢老师！

展开

天天向善

2020-03-23

每秒10万，能不能再给一些数字，改造后没有用云lb,自建nginx集群，这个云上也是有vip是吗，这个流量是每台机器设多少固定带宽?还有小程序服务端与基础服务共100个实例，还是仅小程序服务端，基础服务当时用了多少实例，另外一个容器大约cpu与内存什么配置

展开

川杰

2020-03-22

老师好，消息推送中心通过长连接的方式保持，但是每个长连接都有一定的资源消耗；如果上游的请求过多，这个资源消耗过大的问题怎么处理？

展开

作者回复: 这个只是简单连接，实际数据还是通过服务接口获取，几万个连接问题不大，也可以加机器增强处理能力。

zeor
2020-03-21

老师您好 请问下单时怎么保证超卖 请指教具体方案和实现

作者回复: 办法很多，这里举两个例子:
悲观锁，
select for update 提前锁定库存
乐观锁
库存记录有个字段标识它的版本，读库存的时候，获取版本信息，比如1。后面更新的时候，检查记录的版本是不是还是1，如果不是，则写失败，如果是，则写成功，同时更新版本为2。

image
2020-03-21

请教一下，生产环境使用redis做缓存，一般的部署模式是什么？sentinel，cluster， m/s?

作者回复: 我们部署在公有云，之前是自搭的三节点cluster，现在直接买云的缓存服务，背后是m/s。


Alex
2020-03-20

限流、负载、分流、缓存、异步、仿真、监控都上了满满的干货