

笔记整理：技术架构涵盖内容和演变过程总结

原创 小傅哥 bugstack虫洞栈 2021-03-05 07:55

收录于合集

#架构设计

15个

持续坚持原创输出，点击蓝字关注我吧



作者：小傅哥

博客：<https://bugstack.cn>




沉淀、分享、成长，让自己和他人都能有所收获！😄



目录

- 一、前言
- 二、架构演变
 - 1. 单体架构
 - 2. 应用与数据库分离
 - 3. 使用缓存抗量

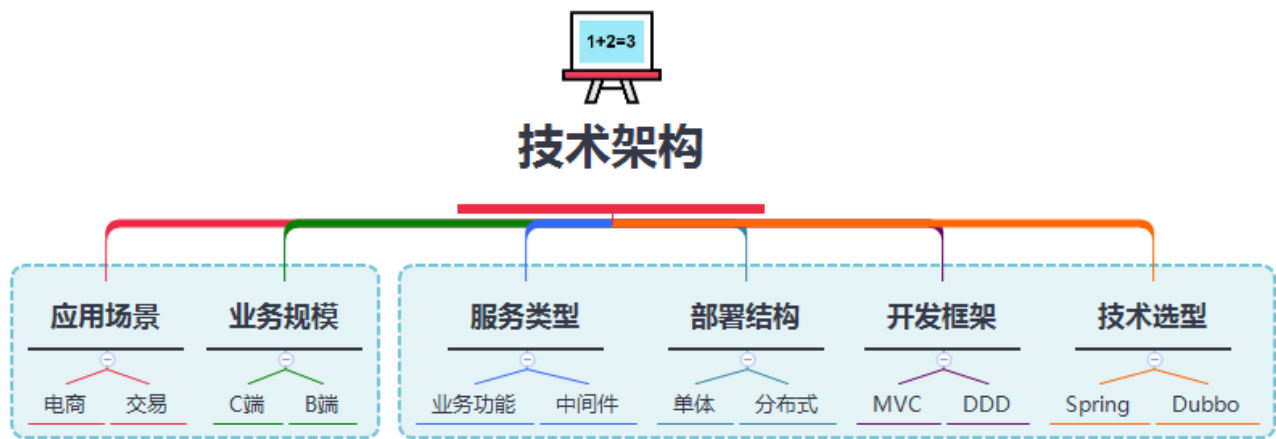
- 4. 多应用部署和Nginx反向代理
- 5. 数据库读写分离
- 6. 应用分组部署
- 7. 应用分库设计
- 8. RPC 分布式部署
- 9. 应用细分和网关引入
- 10. 低代码编程和可复用
- 三、架构图  下载
- 四、总结
- 五、系列推荐

一、前言

架构，说的是开发用的框架吗？

对于刚接触编程的新人来说，可能并不能很清楚的知道架构是怎么来的，都包括什么内容。如果非得说什么架构，那么可能就是目前在 IDEA 中打开的工程就是架构。

抛开技术圈内的架构而已，盖房子的图纸算不算架构、做豆腐的步骤算不算架构、结婚的流程算不算架构？归纳得出，所有的这些步骤都在计算成本、耗材、执行和产出，那么架构就可以看做是一个用于完成目标结果的指导蓝图，现在在放到技术架构的层面来看，架构就不只是我们研发人员用到的技术框架，还需要根据场景、规模，设定技术选型、实施标准、部署结构，综合来完成一个项目的交付。



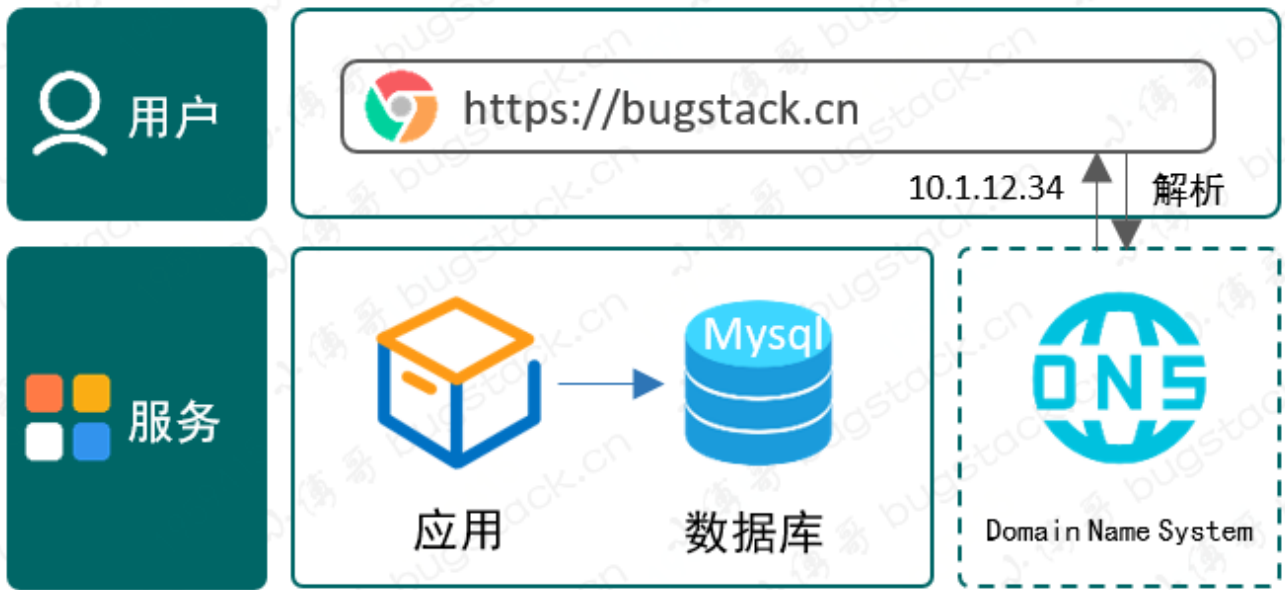
- **应用场景**：你的应用场景是最先决定你采用哪种架构的，这可能会包括：电商、交易、社交、视频、音乐、出行、外卖等等，当然除了互联网中的应用场景，还会有一些基于物联网的应用，例如：PLC 应用、IO 板卡、中继器打码以及你熟悉的小区自提柜。
- **业务规模**：这决定了你的用户范围和体量，如果你是在当下正火的抖音里开发商城，那你的用户体量基数从上线之初就会特别大，但如果你还是一个初创团队小电商，那么每天的QPS维持在 5~10，可能这个阶段你就不需要有能承载多大体量的系统架构。这也类似网络上的笑话，团队初期招聘某大厂大佬，上来就是超级架构的建设，没等系统开发完呢，公司没了！
- **服务类型**：有了场景和规模的设定，接下来要考虑的内容就是整个技术实现层面的内容，服务类型可能是整个团队最初对系统拆分模块和如何支持的考量，有了业务的分层就可以划分出由各个团队来协同支持开发。当然如果是小团队那么这一环节最好缩小，哪怕把所有的功能都开发到一个系统里去，先快速验证市场是主要的。
- **部署结构**：是部署结构决定了开发模式，单体部署、集群部署、分布式部署、云环境等，这些都会决定技术的选型和框架的结构。例如不引用RPC，那么就很难实现分布式部署，如果不使用分库分表和大数据环境，也很难支撑起分布式部署下的数据应用。
- **开发框架**：MVC、DDD，这应该是研发人员最先接触到整个系统架构中的代码开发部分，也就是具体功能的具体实现层，如果是单体应用那么基本一个 MVC 结构就够了，但如果是大体量的分布式部署，那么你的系统开发里可能有的是操作数据库的，有的是专门做业务的，有的是用于支持分布式任务和消费MQ消息的。
- **技术选型**：其实开发框架，无论是 MVC 还是 DDD，都是不影响技术选型的，任何一种语言都可以放在同样的架构框架中进行开发，比如你说 Java、PHP、GO，只不过它们都是在自己语言下有自己的解决方案。

综上，就是我们研发人员在做架构设计时要考虑的核心内容，随着我们技术的不断迭代也会有更多更新的思想，就像20年热起来中台、21年热起来的低代码，都是为了更好的让架构降本增效的实施方案。

但如果想了解和学习架构，最好还是要从它是一颗小树苗时候看起，看看它是如何一点点长大的。在头脑中有了这样一个这样的架构体系，也能让大家更好的理解和设计你需要的架构。

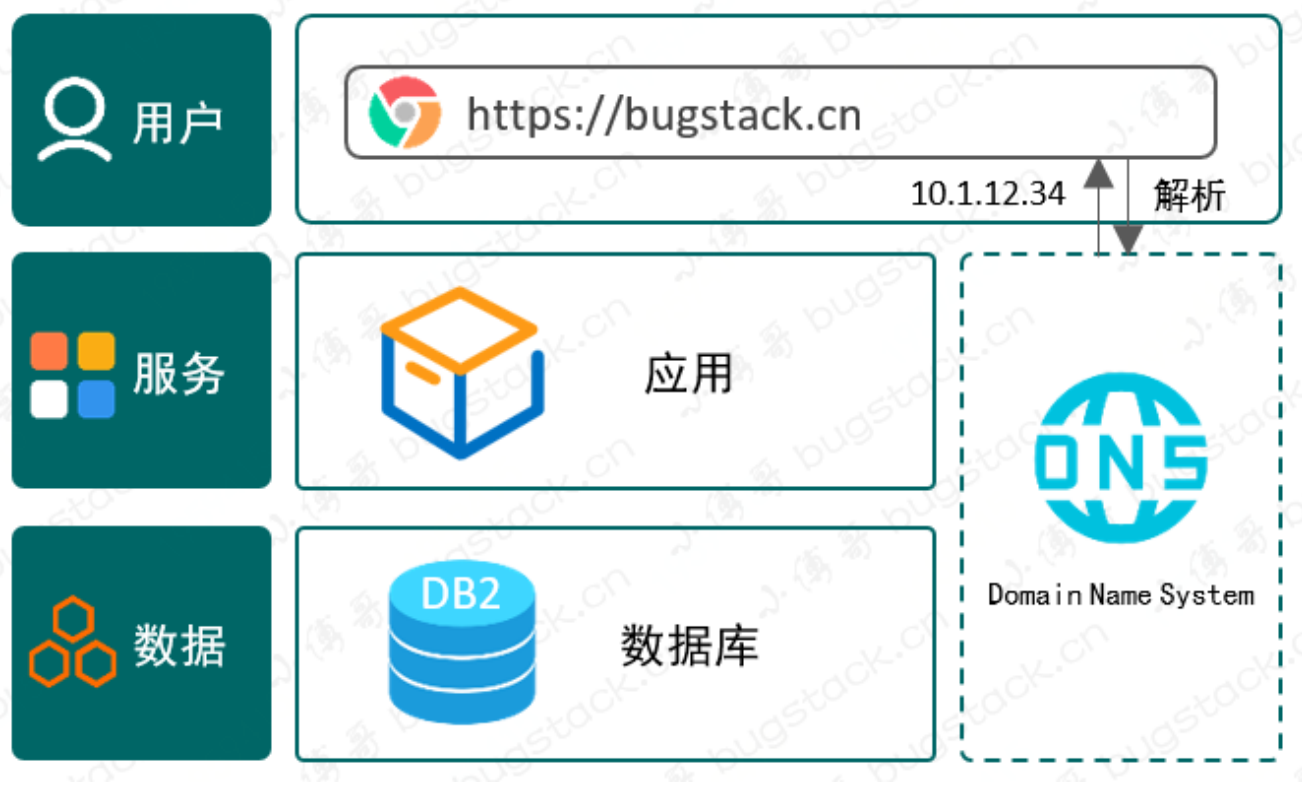
二、架构演变

1. 单体架构



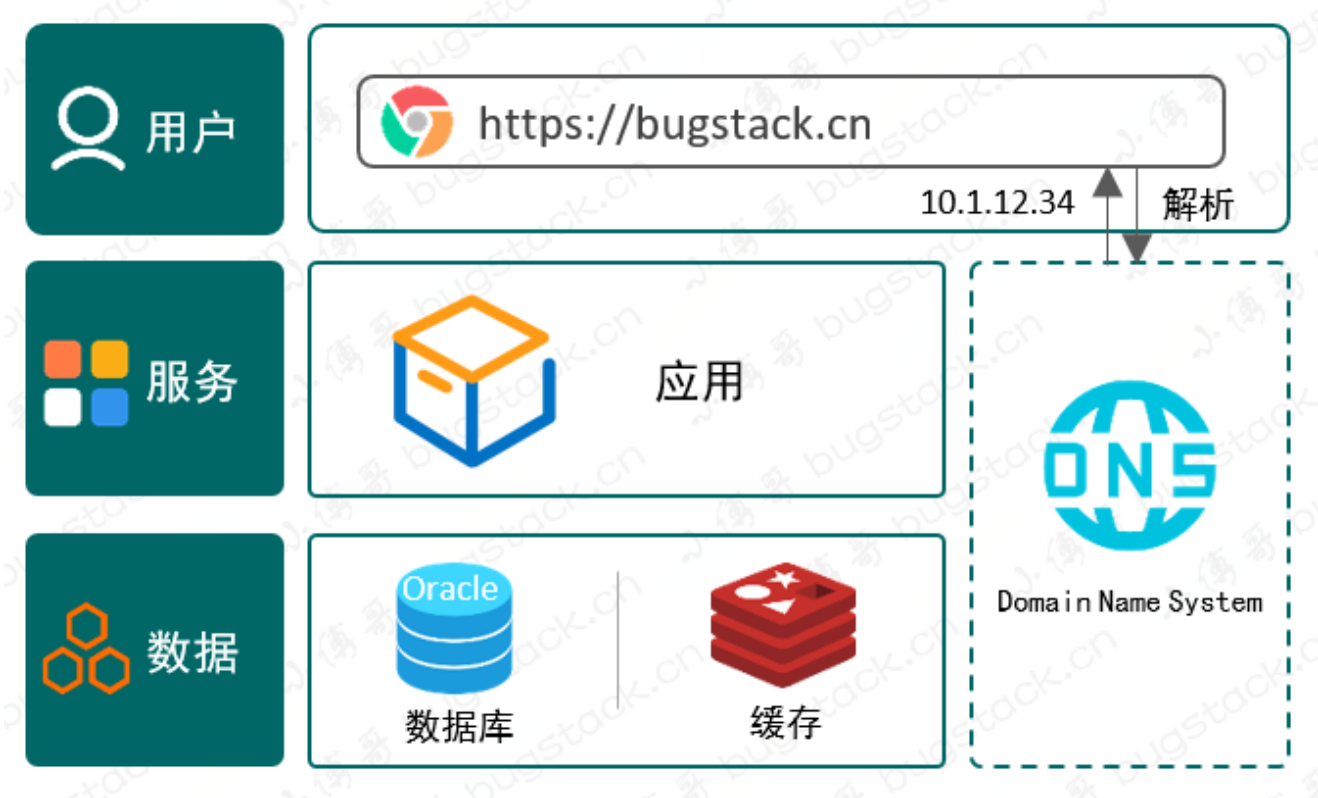
- 体量：★
- 技术：tomcat、weblogic、Java、Mysql、MVC
- 描述：我的博客 bugstack.cn 基本就是这种架构，只不过开发语言不是Java的。这种结构适合体量较小的业务场景，通常都是大佬在互联网初期自研的网站，不过现在这种模型并没有过期，依旧有它的应用场景。

2. 应用与数据库分离



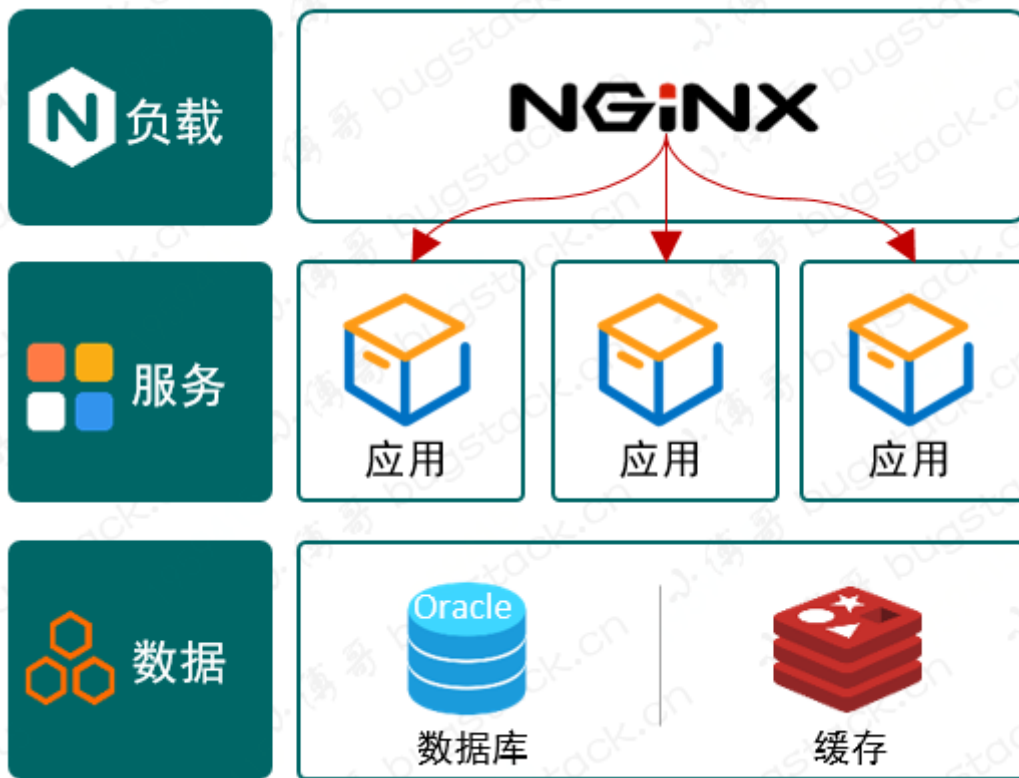
- 体量：★
- 技术：tomcat、weblogic、Java、DB2、MVC
- 描述：这一阶段的拆分其实没有太多变体，主要是由于原有的单体架构应用和数据库部署在一台服务器上，导致性能不足。那么最简单高效的拆分就是把应用和数据库分离开了，让它们在各自的服务器上发挥最大性能。

3. 使用缓存抗量



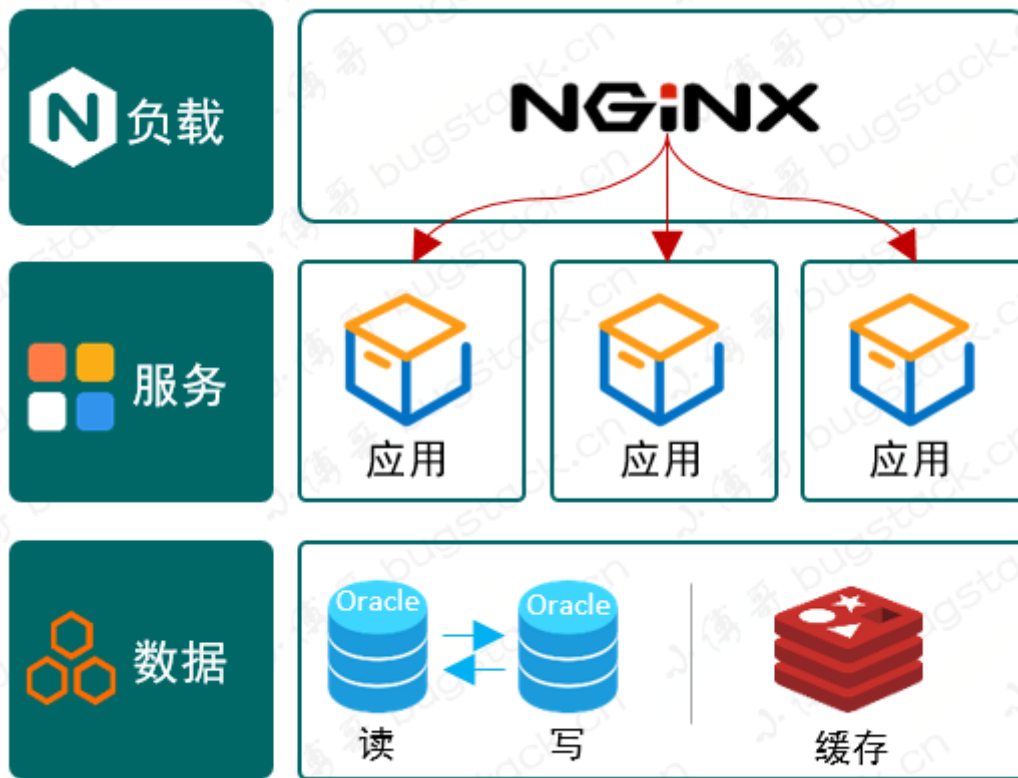
- 体量：☆☆
- 技术：tomcat、weblogic、Java、Oracle、MVC、Redis
- 描述：在这个阶段大家发现，我们需要频繁的从数据库中拉取数据，非常耗费性能。也尝试把一部分数据存放在本地内存，但在服务重启后这部分数据就没有了。因此引入了Redis这样的缓存服务，在这个阶段还是非常大的提升了整体服务的性能。

4. 多应用部署和Nginx反向代理



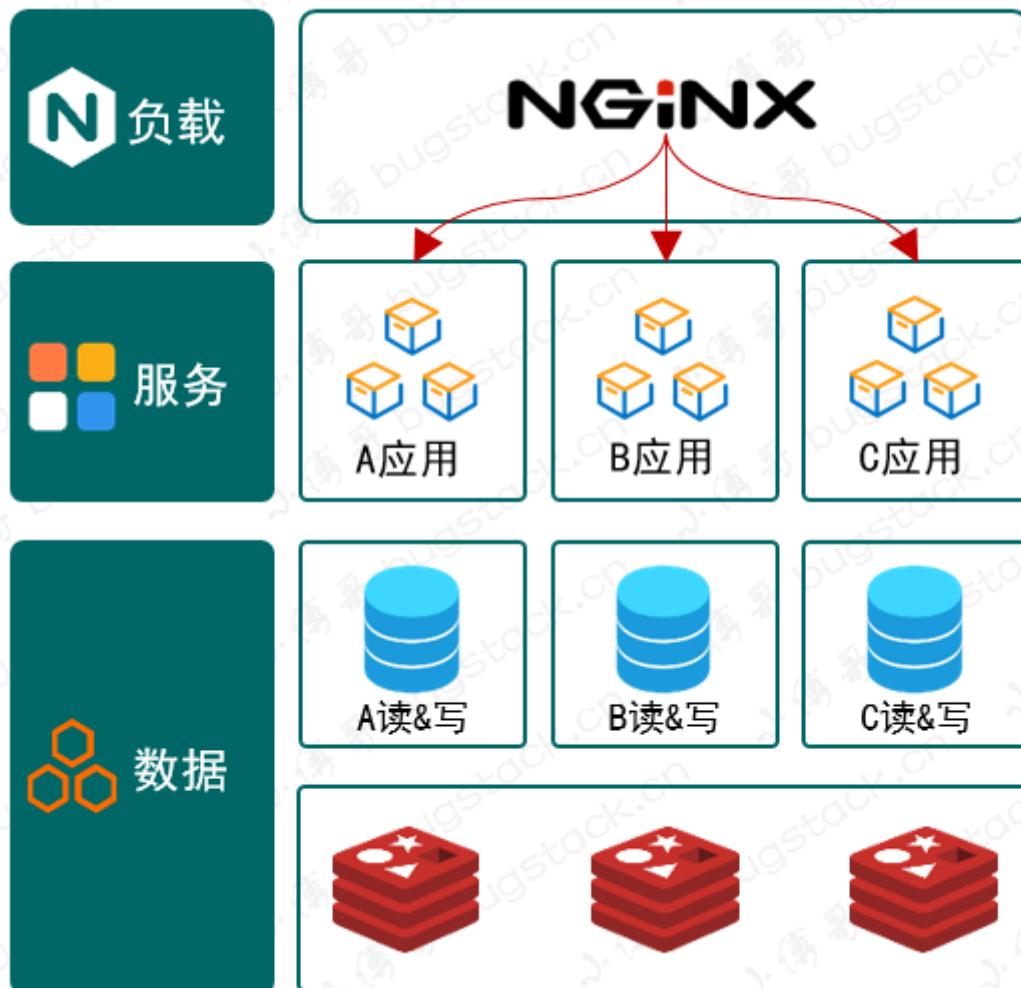
- 体量：☆☆☆
- 技术：tomcat、weblogic、Java、Oracle、MVC、Redis、Nginx
- 描述：当单个服务的承载体量已经到极限了以后，就能想到的就是把服务部署多套，因为这些服务都是做着同样的事，数据库又都是统一一套的，那么通过Nginx的反向代理，就可以把用户的请求分散到不同的服务上去，大大的减轻了服务压力。

5. 数据库读写分离



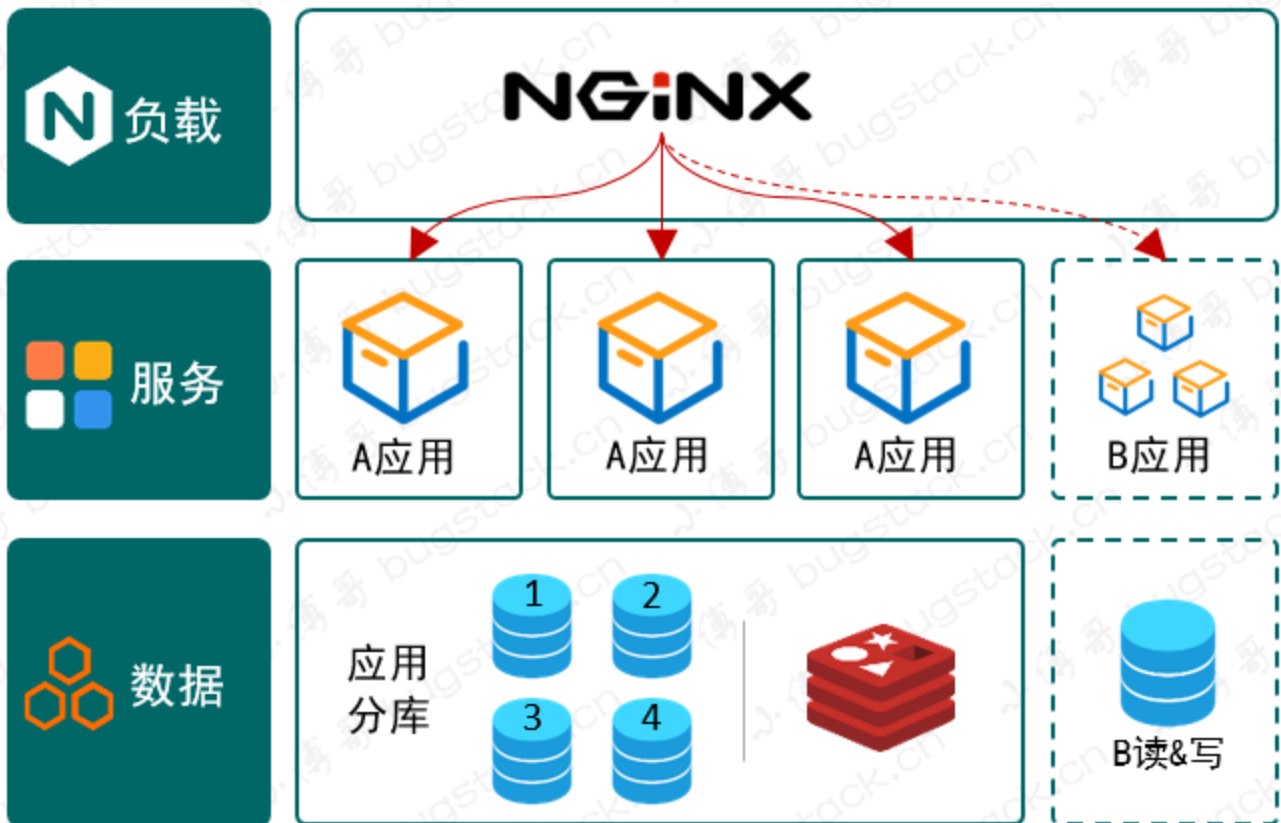
- 体量：☆☆☆
- 技术：tomcat、weblogic、Java、Oracle、MVC、Redis、Nginx
- 描述：数据库的读写分离设计，更多的是因为某些业务场景需要大量的事务性写入，影响到需要读操作的业务。但读写分离的设计并没有太大程度上提升系统性能，因为很大程度的读操作已经使用 Redis 抗住。不过这样的设计思路却为后续的架构模型提供了新的思路。

6. 应用分组部署



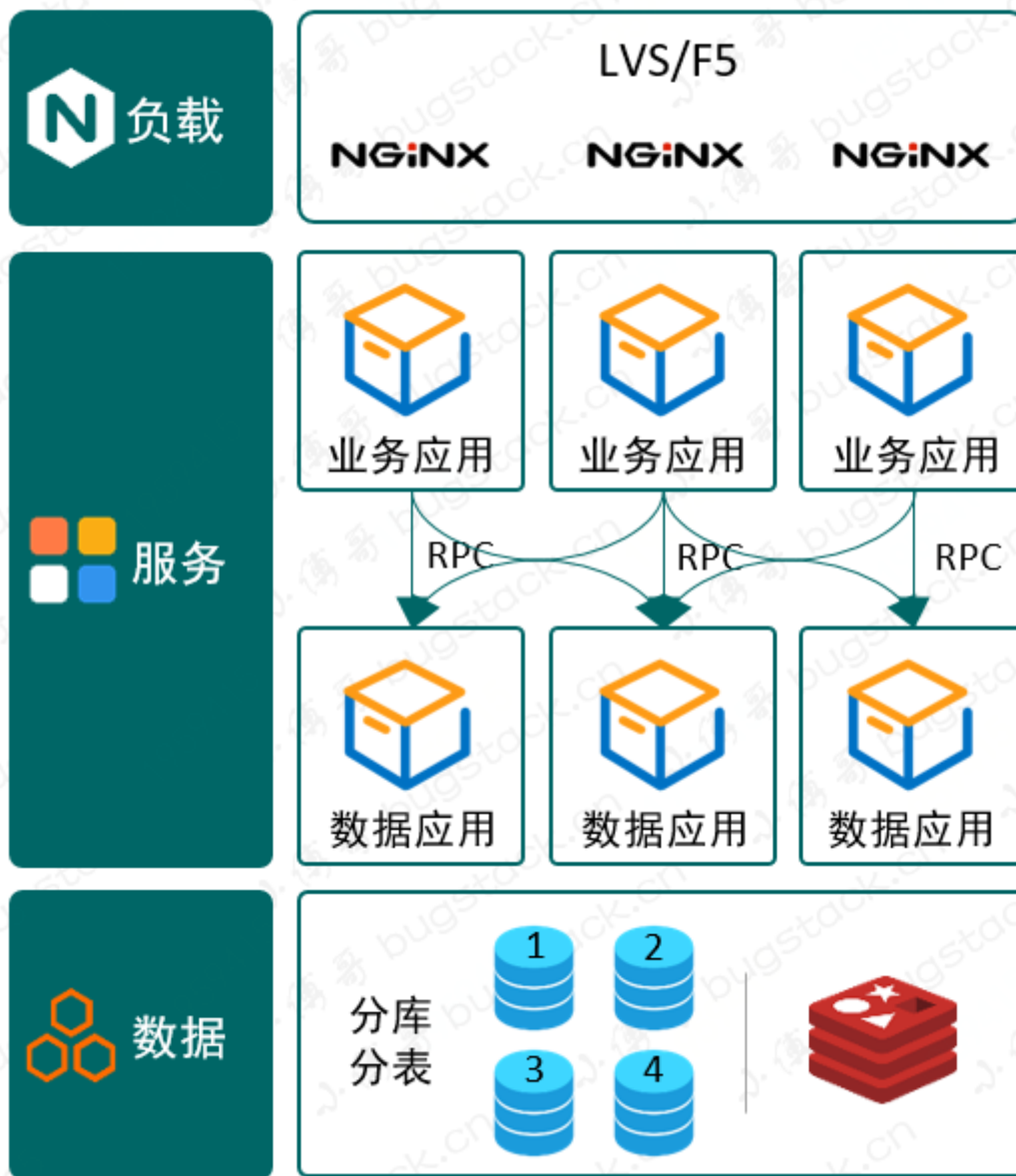
- 体量：★★★★★
- 技术：tomcat、weblogic、Java、Oracle、MVC、Redis、Nginx
- 描述：所有业务都开发在一个应用上所能承载的用户体量已经到极限了，那么接下来最好架构方式就把不同的业务拆分为不同的应用，这些应用都配有自己的数据库，也分别部署在自己的服务器内。这样一来就大大提升了整体的负载能力。

7. 应用分库设计



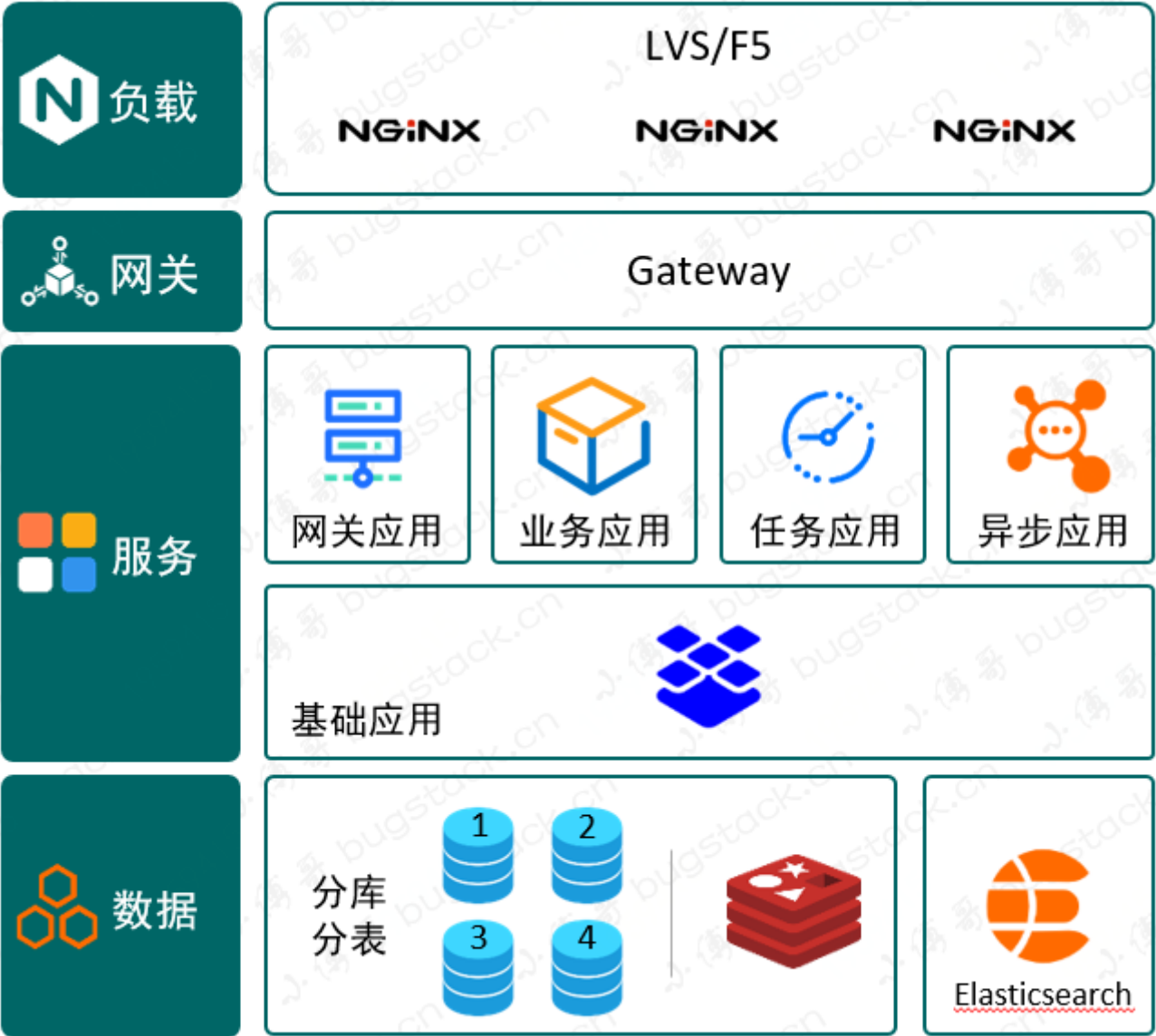
- 体量：★★★★
- 技术：tomcat、weblogic、Java、Oracle、MVC、Redis、Nginx、MyCat
- 描述：当应用按照不同的业务各自系统拆分以后，接下来的瓶颈就在于已经独立的应用用户体量依旧很大，对应的数据库热连接数持续增高。所以在当前条件下，开始设计应用分库操作，同时后可能也会在这个阶段引入分表操作。这样一来单个应用的负载能力又得到了一大截的提升，但是拆库以后也需要引入分布式事务、数据汇总等其他技术的使用，来解决拆库新增的问题。

8. RPC 分布式部署



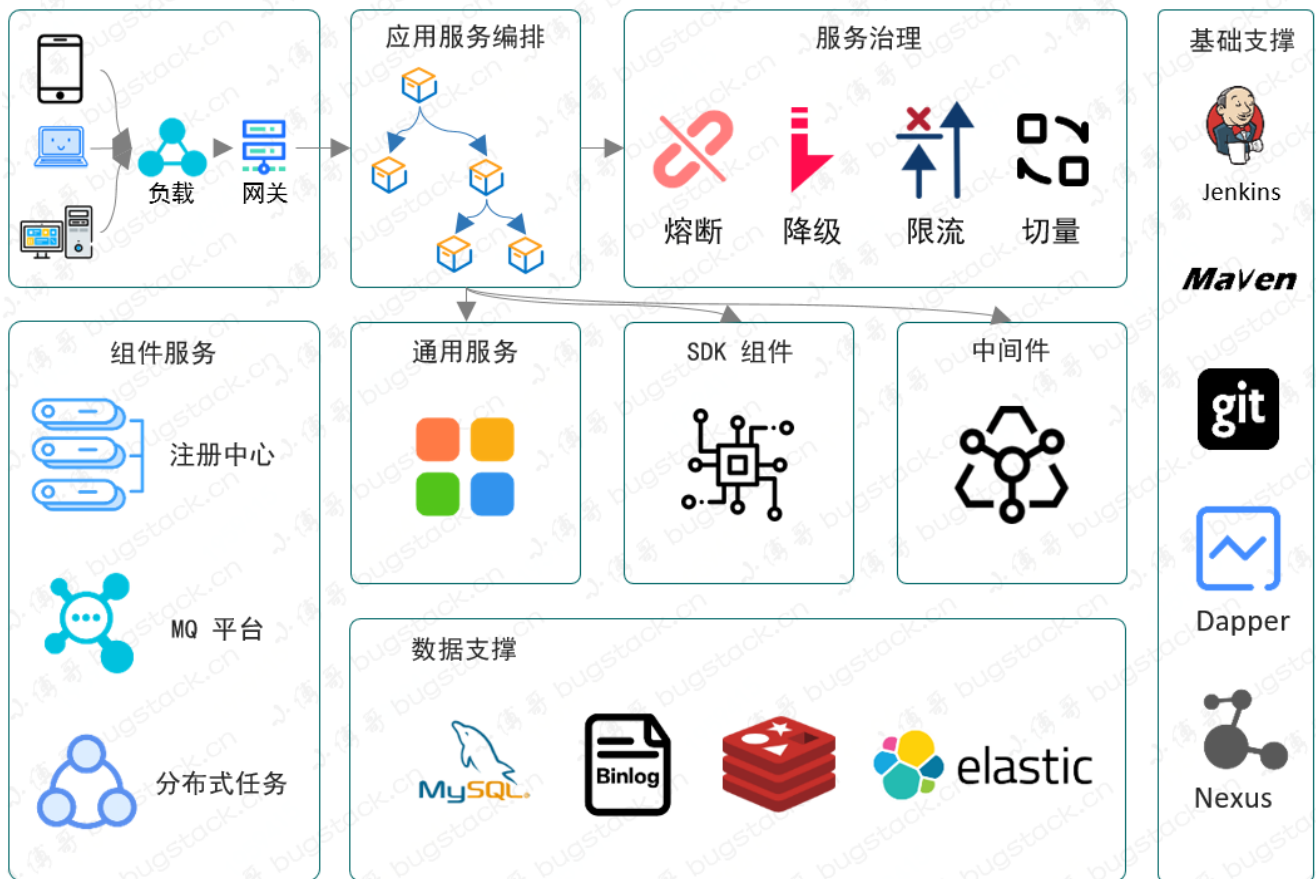
- 体量：★★★★★
- 技术：tomcat、weblogic、Java、Oracle、MVC、Redis、Nginx、MyCat、RPC、LVS/F5
- 描述：在系统不断的再精细化设计以后，其实某些服务并不需要持续的连库操作，它们可能更多的是业务逻辑的包装，同时这些数据库层的操作应用属于底层系统，那么就可以把这样系统用于连库操作，而上层服务通过RPC框架来连接这样的服务。那么，现在就可以通过分布式部署的方式，提升整体的服务性能。

9. 应用细分和网关引入



- 体量：★★★★★
- 技术：tomcat、weblogic、Java、Oracle、MVC、Redis、Nginx、MyCat、RPC、LVS/F5、网关、MQ、分布式任务、Elasticsearch
- 描述：从上到下的整个架构演变过程，我们不断的拆分应用、单独部署一直到应用细分，都是在不断的提升应用服务的能力，让各自应用体负责独立的事情。这个阶段已经开始体现出微服务的能力了，同时这个阶段也引入了上层的网关统一接入和下层的数据使用能力。

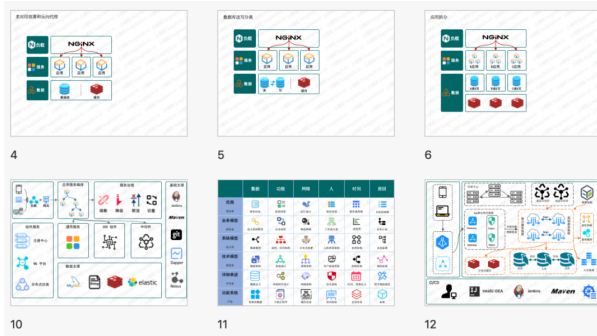
10. 低代码编程和可复用



- 体量：★★★★★
- 技术：tomcat、weblogic、Java、Oracle、MVC、Redis、Nginx、MyCat、RPC、LVS/F5、网关、MQ、分布式任务、Elasticsearch、SDK、低代码、支撑服务
- 描述：在目前这个阶段服务框架基本已经可以很好的支撑用户体量，所以也开始考虑如何更高效的开发和交付问题。那么也就引入了服务编排、服务治理以及通用的模块、组件和中间件。而这些设计其实压缩来看基本就是以前你开发的一个应用而已，不过把所有非业务逻辑的通用性功能不断的拆分出来了，再通过这些细分的组件和服务能力的编排，提供所需接口，这样一来也就大大的提升了可持续交付集成的效率。

三、架构图 下载

有小伙伴反馈看了架构图，也有了点自己的想法，但是动手画的时候就很懵，不知道从哪开始。那么小傅哥把画的架构图原稿分享给大家，可以让感兴趣的小伙伴下载使用。



下载方式

- 公众号：bugstack虫洞栈，回复：[架构图](#)，即可获得最新的下载链接。*后续更新和补充会更换链接*
- 添加小傅哥微信(fustack)，备注：[架构图](#)

四、总结

- 本章也是小傅哥在整理系列架构内容资料的一个总结，让 [新人码农](#) 对架构有一个从小到大的认识。在总结整理时也结合现在的架构简化了一部分内容，因为只有剥丝抽茧的看懂最主干的内容，大家才好不断的扩展枝叶。
- 从演变的过程我们可以看到，业务体量会影响部署，部署形态会改变架构，架构会呼应开发方式，最终语言只是当前最合适某种架构的工具，各项技术栈的运用也是为了技术需求而存在。
- 最后，就是如果你也想让图表达出你的意思，那么可以尝试画一画、总结总结，找到一种能适合你表达出结果的画图结构。

五、系列推荐

- [工作两三年了，整不明白架构图都画啥？](#)
- [技术扫盲：关于低代码编程的可持续性交付设计和分析](#)
- [方案设计：基于IDEA插件开发和字节码插桩技术，实现研发交付质量自动分析](#)
- [半年招聘筛选了400+份简历，告诉你怎么写容易被撩！](#)
- [《Java 面经手册》PDF，全书 417 页 11.5 万字，完稿&发版！](#)

- END -

下方扫码关注 [bugstack虫洞栈](#)，与小傅哥一起学习成长、共同进步，做一个码场最贵Coder！

- 回复【设计模式】，下载《重学Java设计模式》，这是一本互联网真实案例的实践书籍，从实际业务中抽离出，交易、营销、秒杀、中间件、源码等众多场景进行学习代码设计。
- 回复【面经手册】，下载《面经手册·拿大厂Offer》，这是一本有深度的Java核心内容，从数据结构、算法、并发编程以及JVM系8不断深入讲解，让懂了就是真的懂。



你好，我是 小傅哥 。一线互联网 [java](#) 工程师、架构师，开发过交易&营销、写过运营&活动、设计过中间件也倒腾过中继器、IO板卡。不只是写Java语言，也搞过C#、PHP，是一个技术活跃的折腾者。

2020年写了一本PDF《[重学Java设计模式](#)》，全网下载量30万+，帮助很多同学成长。同年 github 的两个项目，[CodeGuide](#) 、 [itstack-demo-design](#) ，持续霸榜 Trending，成为全球热门项目

收录于合集 [#架构设计](#) 15

上一篇

工作两三年了，整不明白架构图都画啥？

下一篇

方案设计：基于IDEA插件开发和字节码插桩技术，实现研发交付质量自动分析

[阅读原文](#) 文章已于2021-03-05修改

喜欢此内容的人还喜欢

字节面试，管你是不是刚毕业！

bugstack虫洞栈

技术问题
昨天字节二面被
Lycoyas的主题

Redis 独占锁、分段锁、发布/订阅，自动注入Spring容器高级编码分享！

bugstack虫洞栈



又完结一个新项目，小而美、小而精！

bugstack虫洞栈

