系统架构设计

向旸

FOUR STEPS

基本概念 > 架构演进 > 小结 > 淘宝

section 1

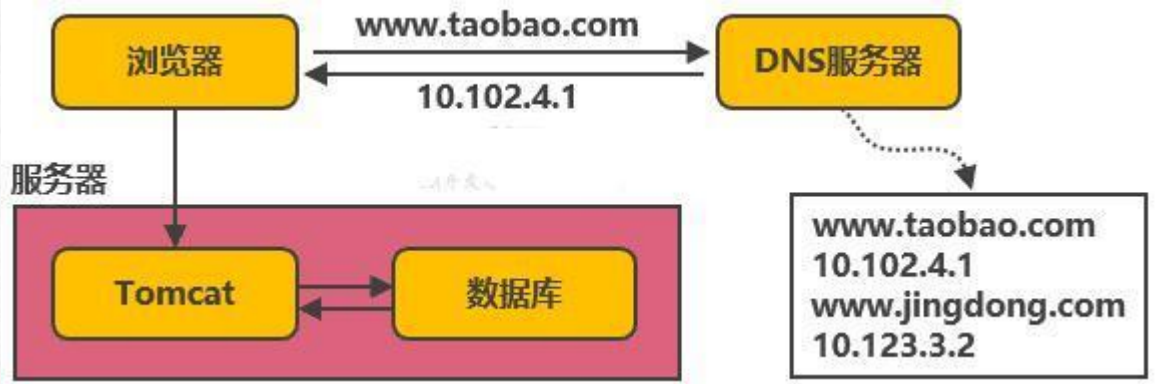
基本概念

1.集群：一个特定领域的软件部署在多台服务器上并作为一个整体提供服务，这个整体称为集群

2.负载均衡

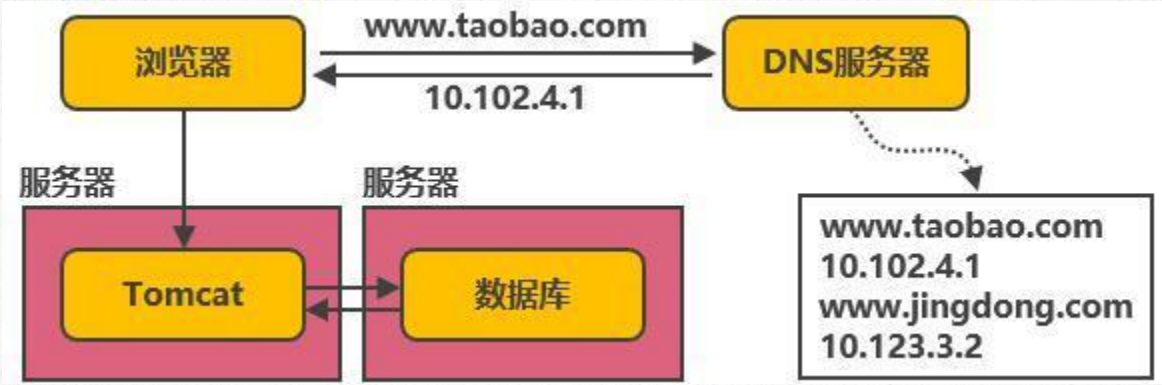
section 2

纯真年代：单机架构



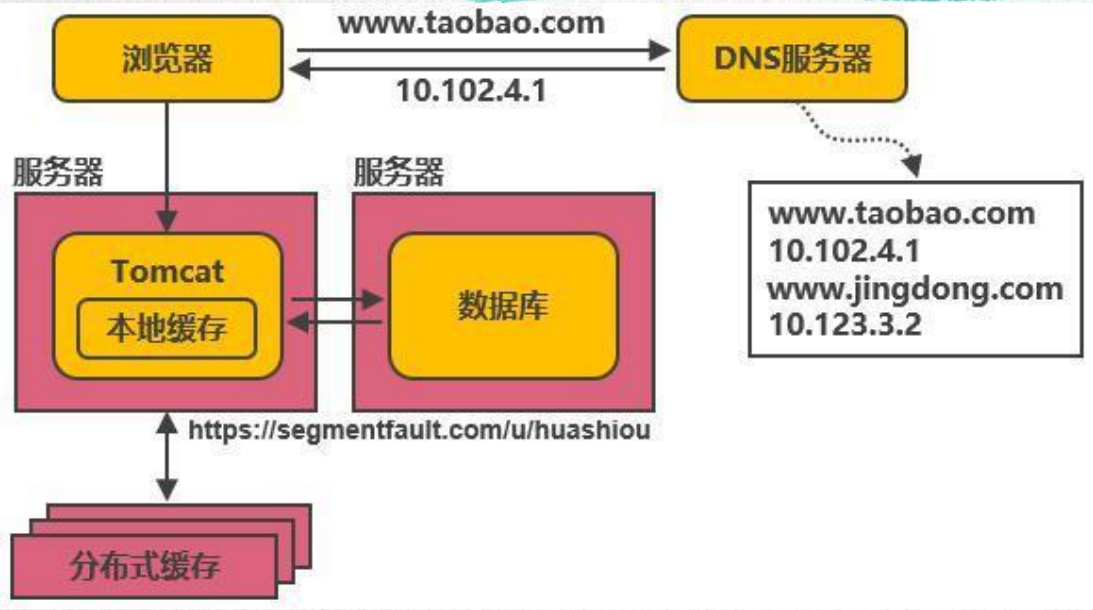
架构瓶颈：随着用户数的增长，Tomcat与数据库之间竞争资源，单机性能不足以支撑业务

1.应用服务与数据服务分离



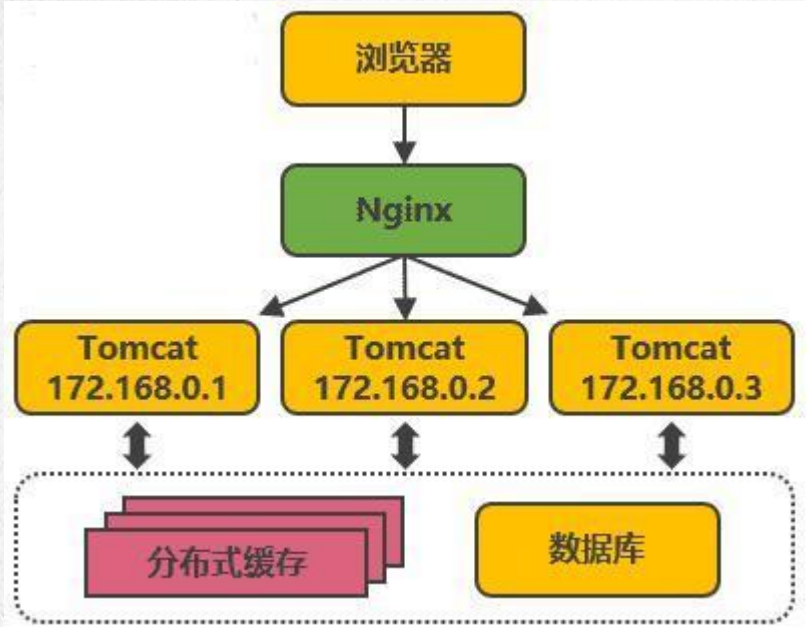
架构瓶颈：随着用户数的增长，并发读写数据库成为瓶颈

2.引入本地缓存和分布式缓存



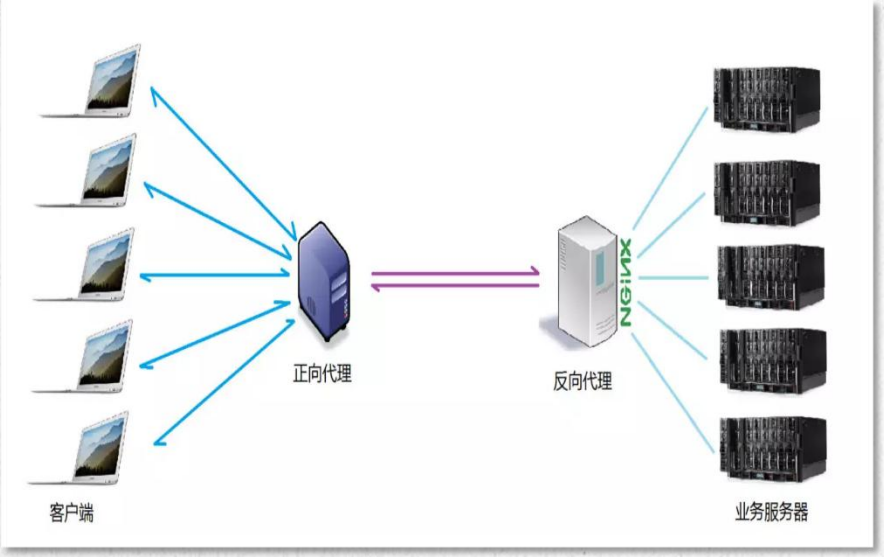
架构瓶颈：缓存抗住了大部分的访问请求，随着用户数增长，并发压力主要落在单机的Tomcat上，响应逐渐变慢

3.引入反向代理实现负载均衡



架构瓶颈：反向代理使应用服务器可支持的并发量大大增加，但并发量的增长也意味着更多请求穿透到数据库，单机的数据库最终成为瓶颈。

正向代理与反向代理



正向代理的用途：1.访问原来无法访问的资源 2.做缓存，加速访问资源 3.对客户端访问授权，上网进行认证 4.上网行为管理，对外隐藏用户信息

反向代理的作用：1.保证内网的安全，通常将反向代理作为公网访问地址 2.负载均衡

常用WEB服务器对比

server	Apache	Nginx	Lighttpd
Proxy代理	非常好	非常好	一般
Rewriter	好	非常好	一般
Fcgi	不好	好	非常好
热部署	不支持	支持	不支持
系统压力比较	很大	很小	比较小
稳定性	好	非常好	不好
安全性	好	一般	一般
静态文件处理	一般	非常好	好
反向代理	一般	非常好	一般

Apache：稳定、开源、跨平台，重量级，不支持高并发

Nginx：跨平台、轻量级高并发服务器、反向代理

某宝：Tengine

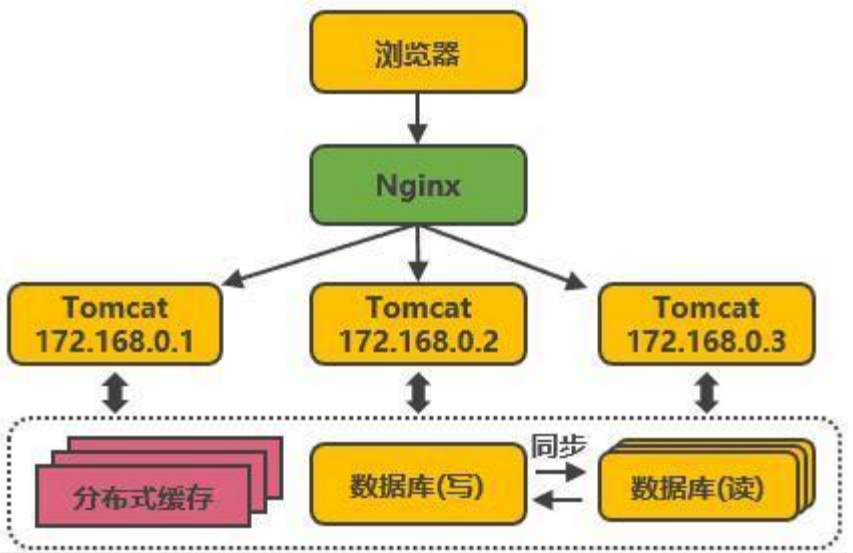
NGINX负载均衡算法

1.weight轮询

2.ip_hash/url_hash

3.fair:智能调度算法

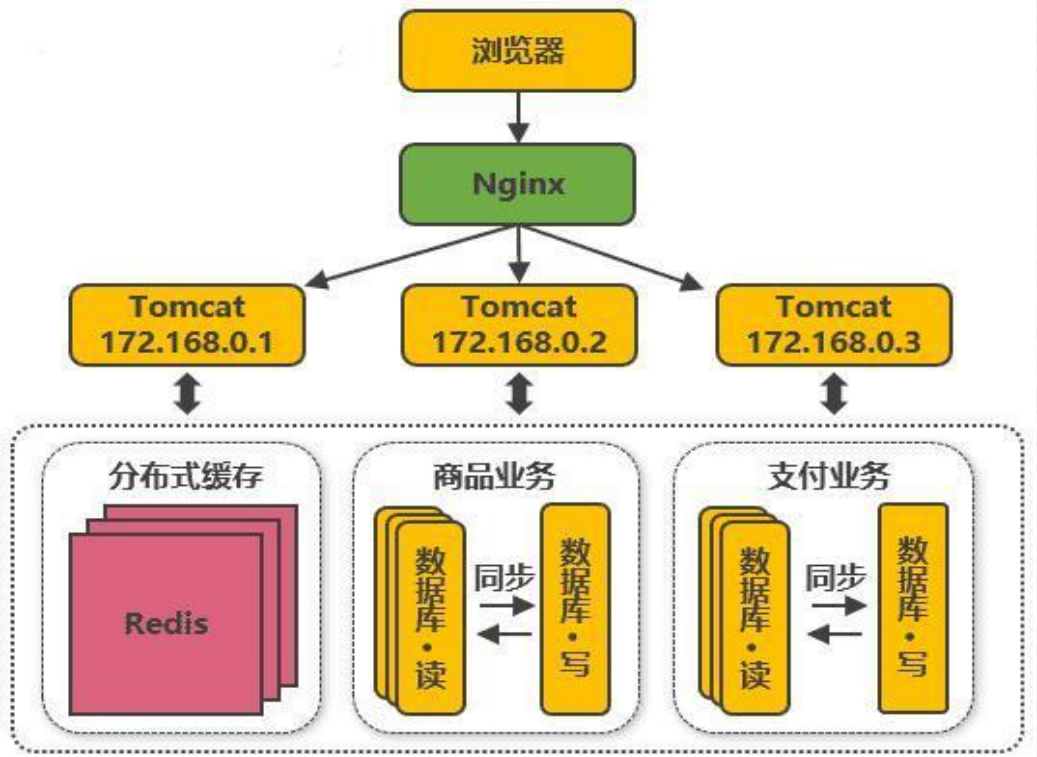
4.数据库读写分离



Mycat：数据库中间件

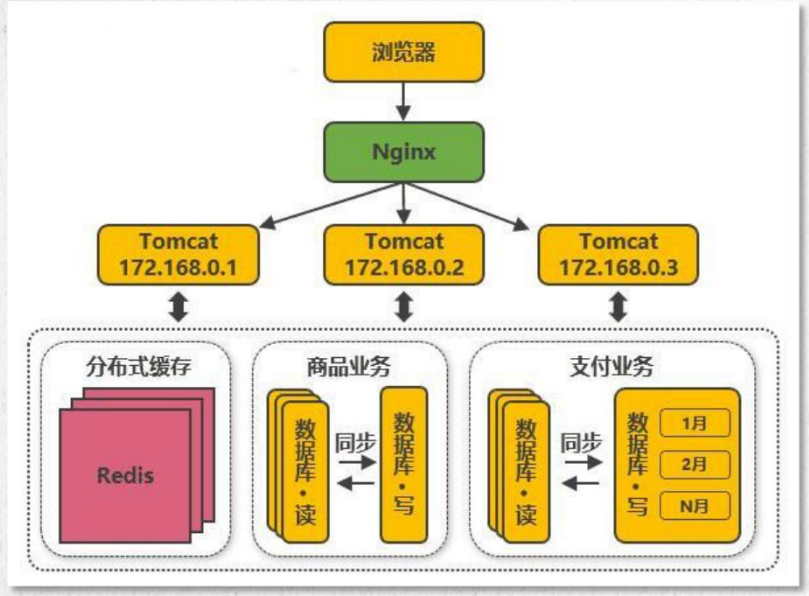
架构瓶颈：业务逐渐变多，不同业务之间的访问量差距较大，不同业务直接竞争数据库，相互影响性能。

5.数据库按业务分库



架构瓶颈：随着用户数的增长，单机的写库会逐渐会达到性能瓶颈。

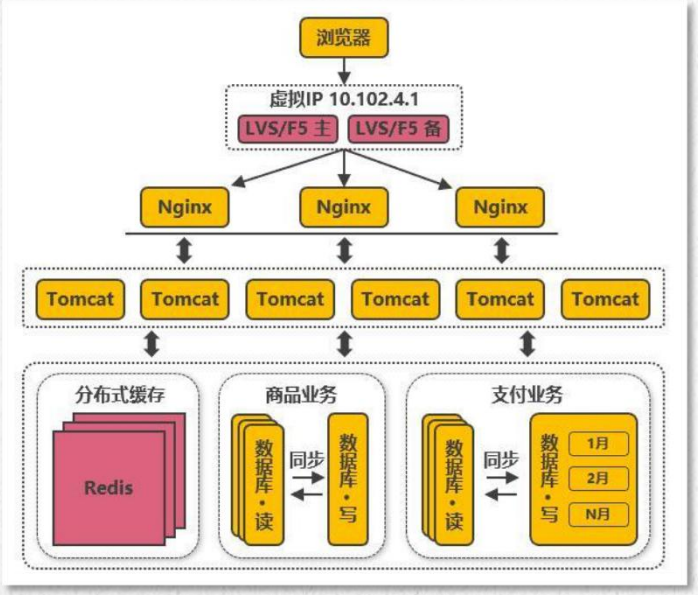
6.把大表拆分为小表



分库分表的管理和请求分发，由Mycat实现，SQL的解析由单机的数据库实现，读写分离可能由网关和消息队列来实现，查询结果的汇总可能由数据库接口层来实现等等。

架构瓶颈：数据库和Tomcat都能够水平扩展，可支撑的并发大幅提高，随着用户数的增长，最终单机的Nginx会成为瓶颈。

7.使用LVS或F5来使多个NGINX负载均衡

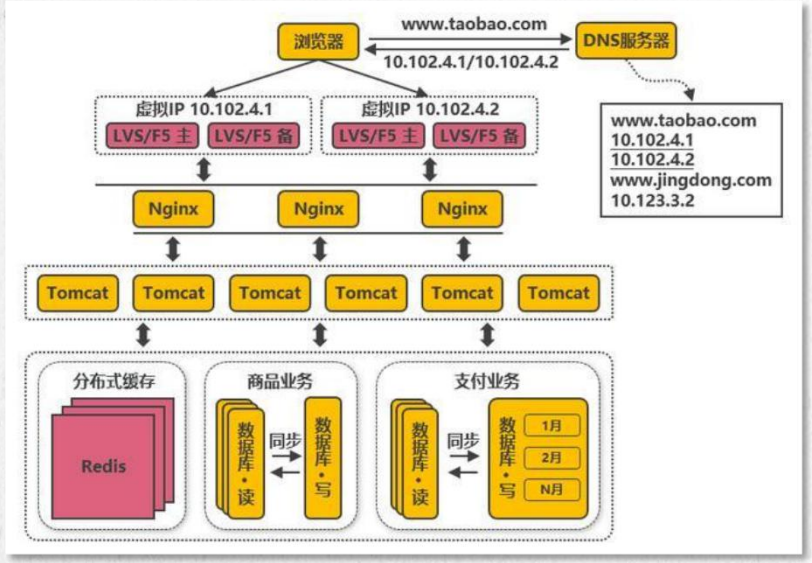


LVS和F5是工作在网络第四层的负载均衡解决方案，其中LVS是软件，运行在操作系统内核态，可对TCP请求或更高层级的网络协议进行转发，因此支持的协议更丰富，并且性能也远高于Nginx。

F5是一种负载均衡硬件，与LVS提供的能力类似，性能比LVS更高，但价格昂贵。

架构瓶颈：由于LVS也是单机的，随着并发数增长到几十万时，LVS服务器最终会达到瓶颈，此时用户数达到千万甚至上亿级别，用户分布在不同的地区，与服务器机房距离不同，导致了访问的延迟会明显不同。

8.通过DNS轮询实现机房间的负载均衡

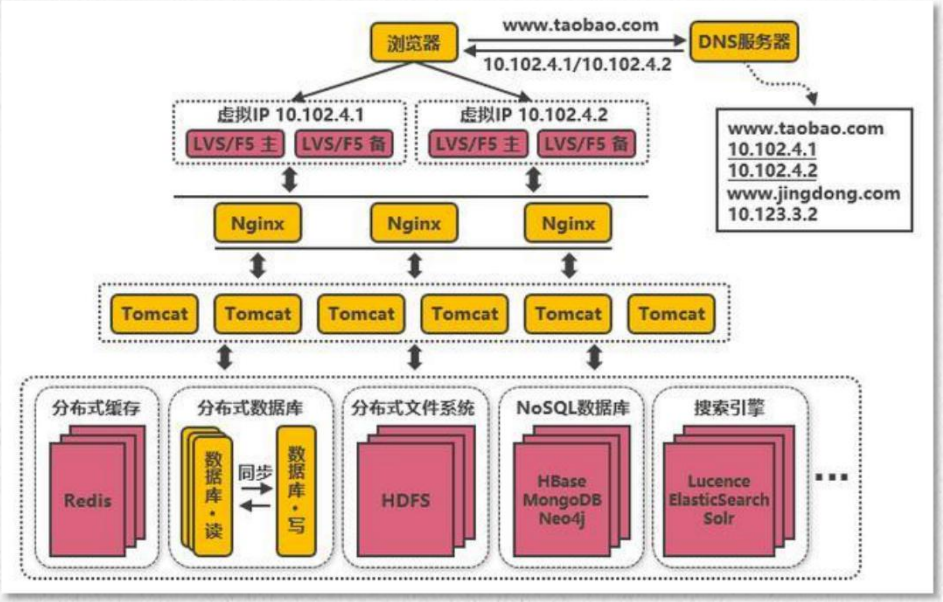


在DNS服务器中可配置一个域名对应多个IP地址，每个IP地址对应到不同的机房里的虚拟IP。

当用户访问taobao时，DNS服务器会使用轮询策略或其他策略，来选择某个IP供用户访问。

架构瓶颈：随着数据的丰富程度和业务的发展，检索、分析等需求越来越丰富，单单依靠数据库无法解决如此丰富的需求。

9.引入NOSQL数据库和搜索引擎等技术

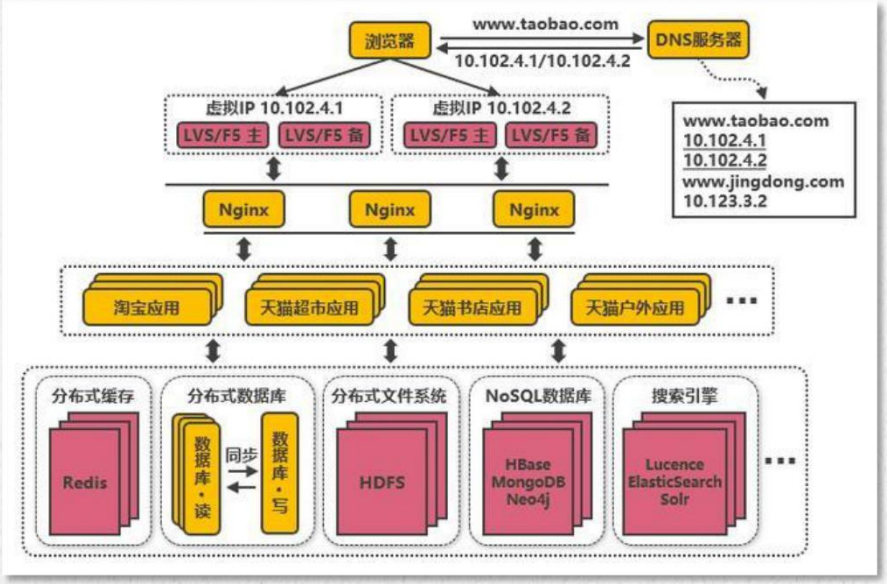


当数据库中的数据多到一定规模时，数据库就不适用于复杂的查询了，往往只能满足普通查询的场景。

对于全文检索、可变数据结构等场景，数据库天生不适用。因此需要针对特定的场景，引入合适的解决方案。

架构瓶颈：引入更多组件解决了丰富的需求，业务维度能够极大扩充，随之而来的是一个应用中包含了太多的业务代码，业务的升级迭代变得困难。

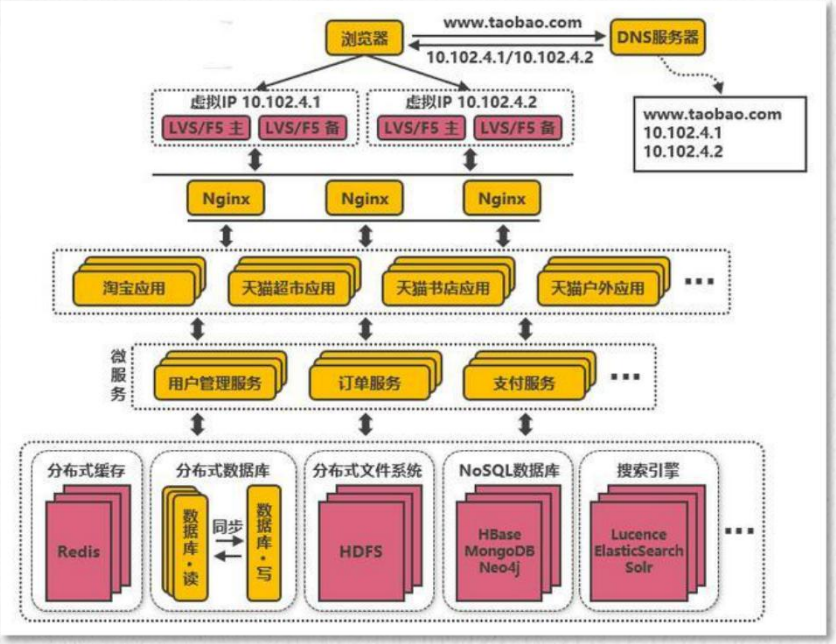
10.大应用拆分成小应用



按照业务板块来划分应用代码，使单个应用的职责更清晰，相互之间可以做到独立升级迭代。这时候应用之间可能会涉及到一些公共配置，可以通过分布式配置中心*Zookeeper*来解决。

架构瓶颈：不同应用之间存在共用的模块，由应用单独管理会导致相同代码存在多份，导致公共功能升级时全部应用代码都要跟着升级。

11.服用的功能抽离成微服务

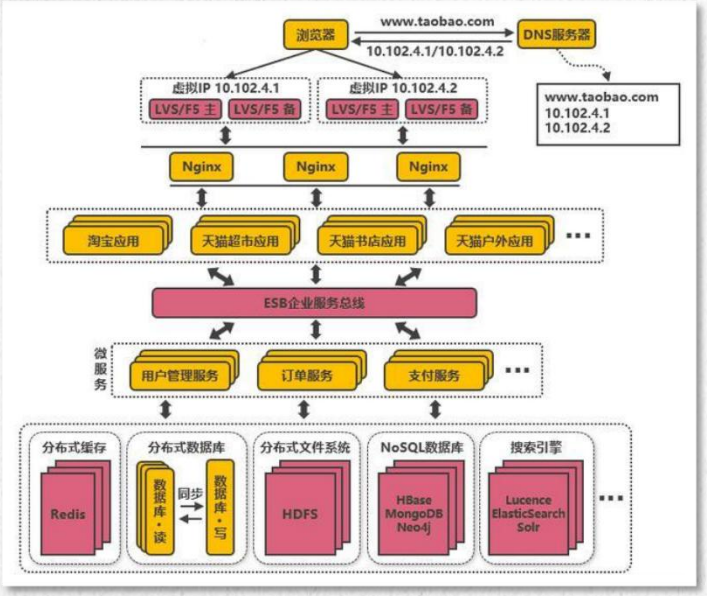


如用户管理、订单、支付、鉴权等功能在多个应用中都存在，那么可以把这些功能的代码单独抽取出来形成一个单独的服务来管理

这样的服务就是所谓的微服务，应用和服务之间通过*HTTP*、*TCP*或*RPC*请求等多种方式来访问公共服务，每个单独的服务都可以由单独的团队来管理。

架构瓶颈：不同服务的接口访问方式不同，应用代码需要适配多种访问方式才能使用服务，此外，应用访问服务，服务之间也可能相互访问，调用链将会变得非常复杂，逻辑变得混乱。

12.引入企业服务总线ESB屏蔽服务接口的访问差异

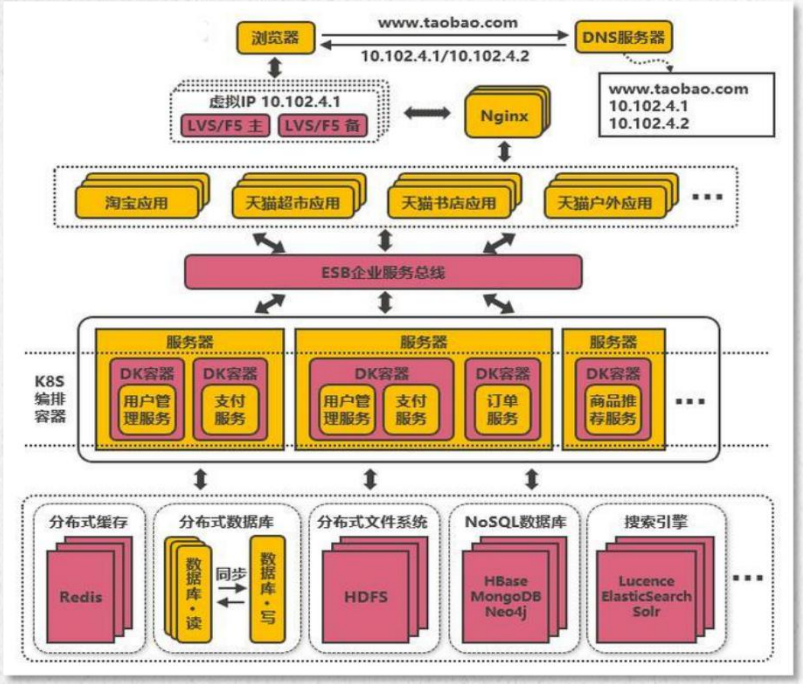


通过ESB统一进行访问协议转换，应用统一通过ESB来访问后端服务，服务与服务之间也通过ESB来相互调用，以此降低系统的耦合程度。

这种单个应用拆分为多个应用，公共服务单独抽取出来来管理，并使用企业消息总线来解除服务之间耦合问题的架构，就是所谓的SOA（面向服务）架构

架构瓶颈：业务不断发展，应用和服务都会不断变多，应用和服务的部署变得复杂，同一台服务器上部署多个服务还要解决运行环境冲突的问题

13.引入容器化技术实现运行环境隔离与动态服务管理

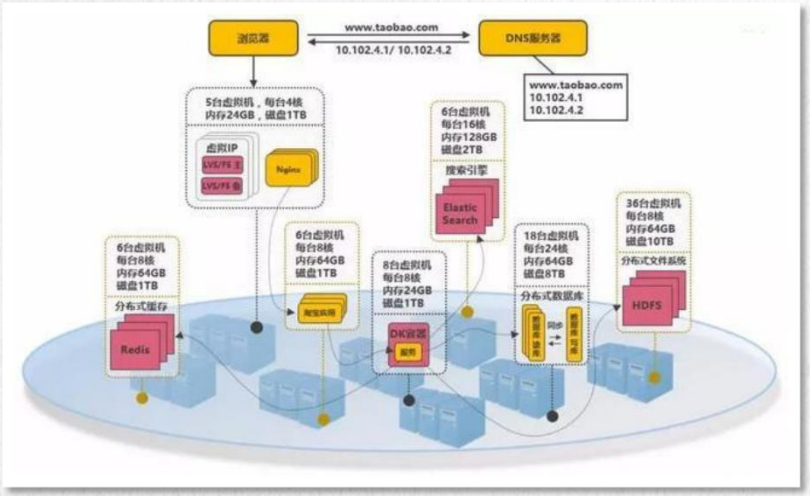


目前最流行的容器化技术是Docker，最流行的容器管理服务是Kubernetes(K8S)，应用/服务可以打包为Docker镜像，通过K8S来动态分发和部署镜像。

把整个“操作系统”打包为一个镜像后，就可以分发到需要部署相关服务的机器上，直接启动Docker镜像就可以把服务起起来，使服务的部署和运维变得简单。

架构瓶颈：使用容器化技术后服务动态扩缩容问题得以解决，但是机器还是需要公司自身来管理，在非大促的时候，还是需要闲置着大量的机器资源来应对大促，机器自身成本和运维成本都极高，资源利用率低。

14.以云平台承载系统



系统可部署到公有云上，利用公有云的海量机器资源，解决动态硬件资源的问题

在大促的时间段里，在云平台中临时申请更多的资源，结合Docker和K8S来快速部署服务，在大促结束后释放资源，真正做到按需付费，资源利用率大大提高，同时大大降低了运维成本。

在云平台上可按需动态申请硬件资源（如CPU、内存、网络等），并且之上提供通用的操作系统，提供常用的技术组件（如Hadoop技术栈，MPP数据库等）供用户使用，甚至提供开发好的应用

section 3

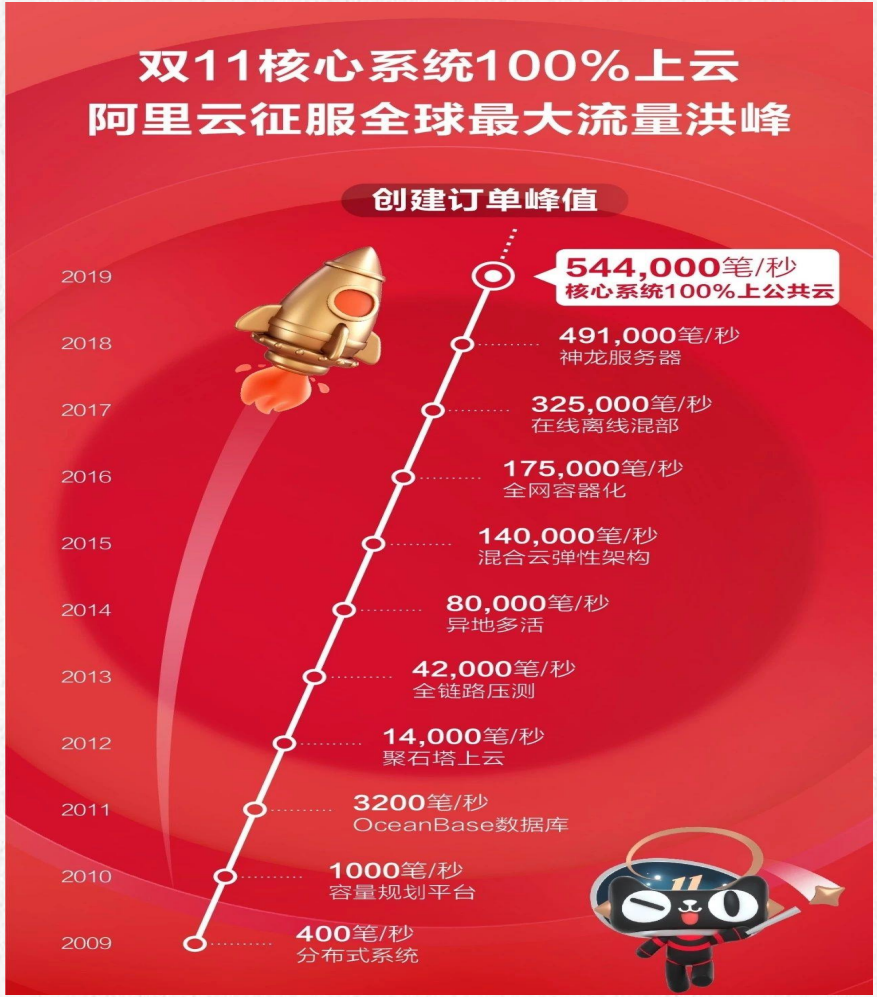
小结

1.分而治之

2.抽离复用

3.解耦

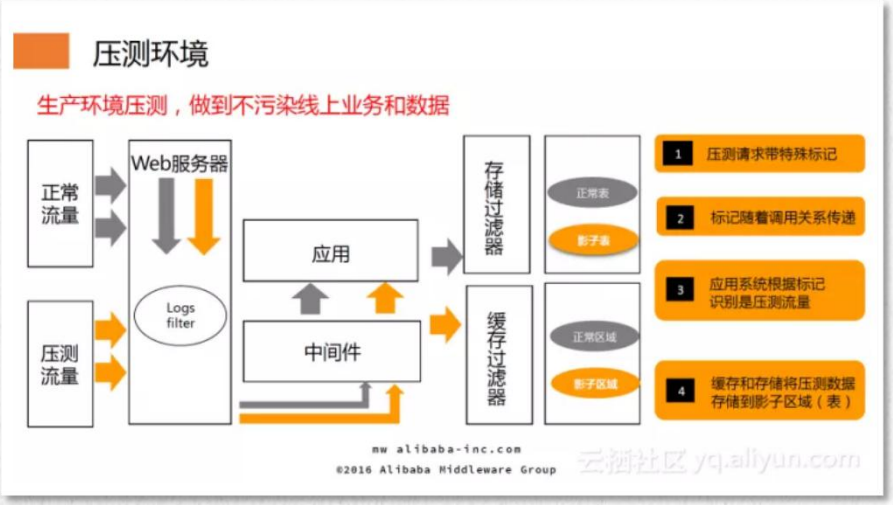
section 4


双11核心系统100%上云
阿里云征服全球最大流量洪峰
创建订单峰值
2019
544,000笔/秒
核心系统100%上公共云
2018
491,000笔/秒
神龙服务器
2017
325,000笔/秒
在线离线混部
2016
175,000笔/秒
全网容器化
2015
140,000笔/秒
混合云弹性架构
2014
80,000笔/秒
异地多活
2013
42,000笔/秒
全链路压测
2012
14,000笔/秒
聚石塔上云
2011
3200笔/秒
OceanBase数据库
2010
1000笔/秒
容量规划平台
2009
400笔/秒
分布式系统

全链路压测-站点稳定性保障最有效的解决方案

全链路压测的本质是让双11零点这一刻提前在系统预演（用户无感知），模拟“双11”同样的线上环境、用户规模、业务场景、业务量级，之后再针对性地进行系统调优，是站点的一次高仿真模拟考试

全链路压测-站点稳定性保障最有效的解决方案



由于是在生产环境做双11的全链路压测模拟，因此防止压测数据和流量污染和干扰生产环境是及其重要的。

要实现这一目标，首先要求压测流量能被识别，采用的做法是所有的压测流量都带有特殊的标记，并且这些标记能够随中间件协议的调用关系进行传递；此后，应用系统根据标记识别压测流量；在缓存和存储时，通过存储和缓存过滤器将压测数据存储到影子区域（表）而不是覆盖原有数据。

相关文章分享

阿里怎么做双11全链路压测？
https://developer.aliyun.com/article/721643?spm=a2c6h.12873639.0.0.18655392Nf986r

12306抢票,极限并发带来的思考
?https://juejin.im/post/5d84e21f6fb9a06ac8248149


Thank you