淘寶為什麼能抗住雙11 ？ 看完這篇文章你就明白了!

huashiou Java後端 今天

作者| huashiou

鏈接| segmentfault.com/a/1190000018626163

上篇| Nginx從入門到實戰

雙11 即將來臨，本文以設計淘寶網的後台架構為例，介紹從一百個並發到千萬級並發情況下服務端的架構的14次演進過程，同時列舉出每個演進階段會遇到的相關技術，讓大家對架構的演進有一個整體的認知。

文章最後匯總了一些架構設計的原則。

基本概念

在介紹架構之前，為了避免部分讀者對架構設計中的一些概念不了解，下面對幾個最基礎的概念進行介紹。

1）什麼是分佈式？

系統中的多個模塊在不同服務器上部署，即可稱為分佈式系統，如Tomcat和數據庫分別部署在不同的服務器上，或兩個相同功能的Tomcat分別部署在不同服務器上。

2）什麼是高可用？

系統中部分節點失效時，其他節點能夠接替它繼續提供服務，則可認為系統具有高可用性。

3）什麼是集群？

一個特定領域的軟件部署在多台服務器上並作為一個整體提供一類服務，這個整體稱為集群。

如Zookeeper中的Master和Slave分別部署在多台服務器上，共同組成一個整體提供集中配置服務。

在常見的集群中，客戶端往往能夠連接任意一個節點獲得服務，並且當集群中一個節點掉線時，其他節點往往能夠自動的接替它繼續提供服務，這時候說明集群具有高可用性。

4）什麼是負載均衡？

請求發送到系統時，通過某些方式把請求均勻分發到多個節點上，使系統中每個節點能夠均勻的處理請求負載，則可認為系統是負載均衡的。

5）什麼是正向代理和反向代理？

系統內部要訪問外部網絡時，統一通過一個代理服務器把請求轉發出去，在外部網絡看來就是代理服務器發起的訪問，此時代理服務器實現的是正向代理；

當外部請求進入系統時，代理服務器把該請求轉發到系統中的某台服務器上，對外部請求來說，與之交互的只有代理服務器，此時代理服務器實現的是反向代理。

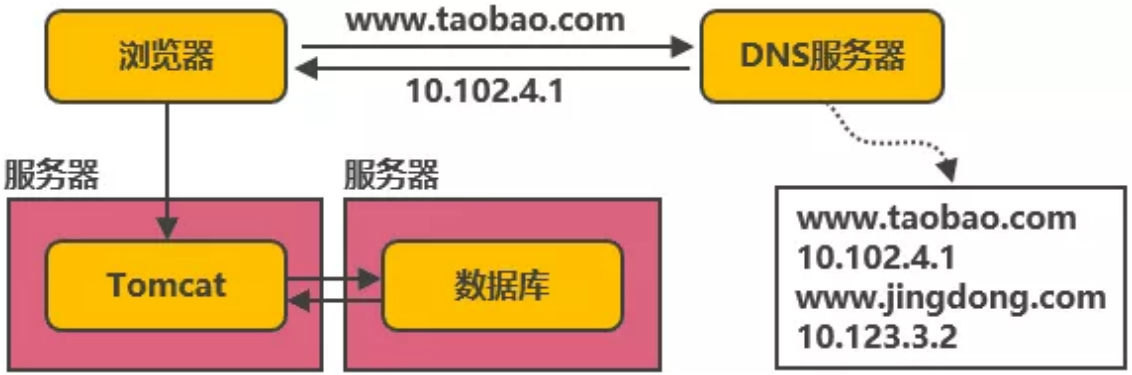
簡單來說，正向代理是代理服務器代替系統內部來訪問外部網絡的過程，反向代理是外部請求訪問系統時通過代理服務器轉發到內部服務器的過程。

純真年代：單機架構

以淘寶作為例子：在網站最初時，應用數量與用戶數都較少，可以把Tomcat和數據庫部署在同一台服務器上。瀏覽器往www.taobao.com發起請求時，首先經過DNS服務器（域名系統）把域名轉換為實際IP地址10.102.4.1，瀏覽器轉而訪問該IP對應的Tomcat。

架構瓶頸：隨著用戶數的增長，Tomcat和數據庫之間競爭資源，單機性能不足以支撐業務。

第一次演進：Tomcat與數據庫分開部署

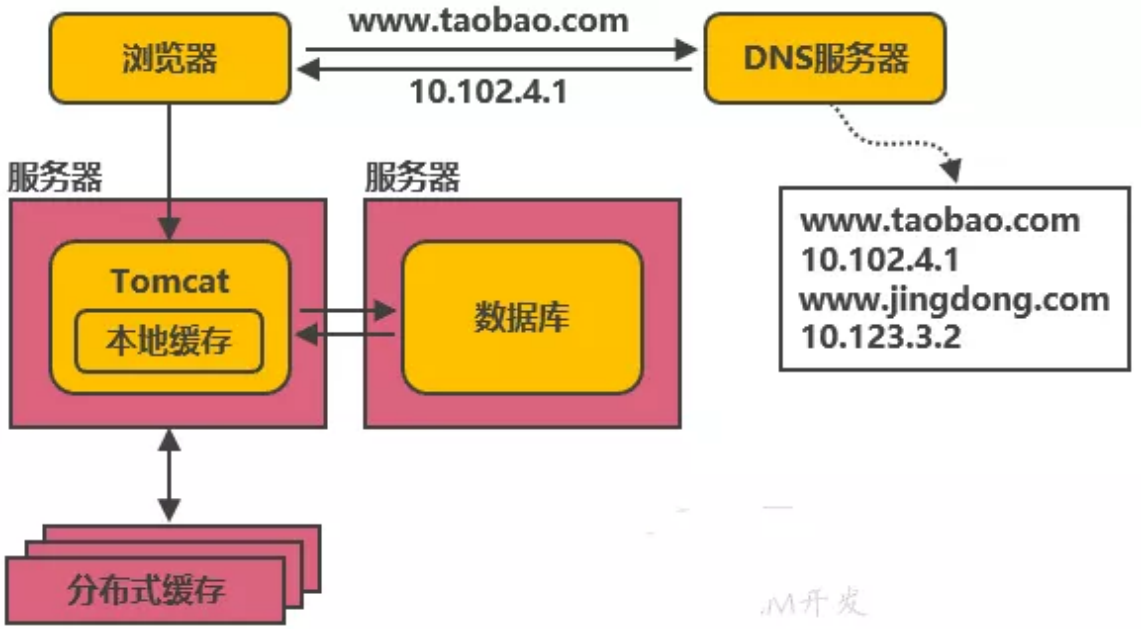


Tomcat和數據庫分別獨占服務器資源，顯著提高兩者各自性能。

架構瓶頸：隨著用戶數的增長，並發讀寫數據庫成為瓶頸。

Tips：歡迎關注微信公眾號：Java後端，獲取更多技術博文推送。

第二次演進：引入本地緩存和分佈式緩存

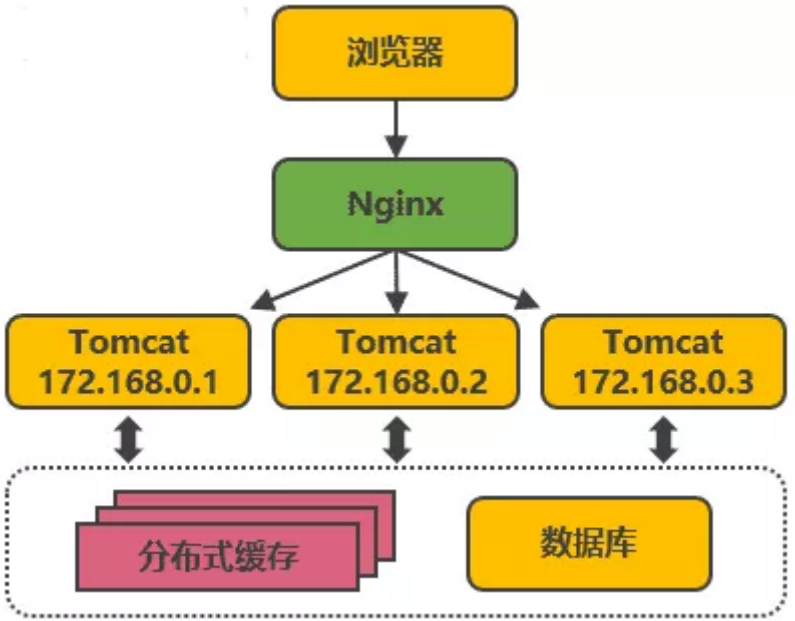


在Tomcat同服務器上或同JVM中增加本地緩存，並在外部增加分佈式緩存，緩存熱門商品信息或熱門商品的html頁面等。通過緩存能把絕大多數請求在讀寫數據庫前攔截掉，大大降低數據庫壓力。

其中涉及的技術包括：使用memcached作為本地緩存，使用Redis作為分佈式緩存，還會涉及緩存一致性、緩存穿透/擊穿、緩存雪崩、熱點數據集中失效等問題。

架構瓶頸：緩存抗住了大部分的訪問請求，隨著用戶數的增長，並發壓力主要落在單機的Tomcat上，響應逐漸變慢。

第三次演進：引入反向代理實現負載均衡

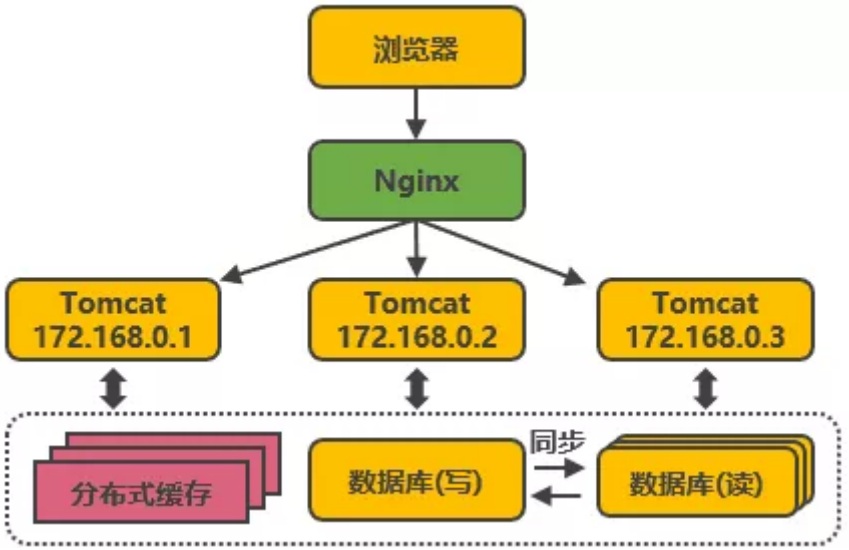


在多台服務器上分別部署Tomcat，使用反向代理軟件（Nginx）把請求均勻分發到每個Tomcat中。此處假設Tomcat最多支持100個並發，Nginx最多支持50000個並發，那麼理論上Nginx把請求分發到500個Tomcat上，就能抗住50000個並發。

其中涉及的技術包括：Nginx、HAProxy，兩者都是工作在網絡第七層的反向代理軟件，主要支持http協議，還會涉及session共享、文件上傳下載的問題。

架構瓶頸：反向代理使應用服務器可支持的並發量大大增加，但並發量的增長也意味著更多請求穿透到數據庫，單機的數據庫最終成為瓶頸。

第四次演進：數據庫讀寫分離

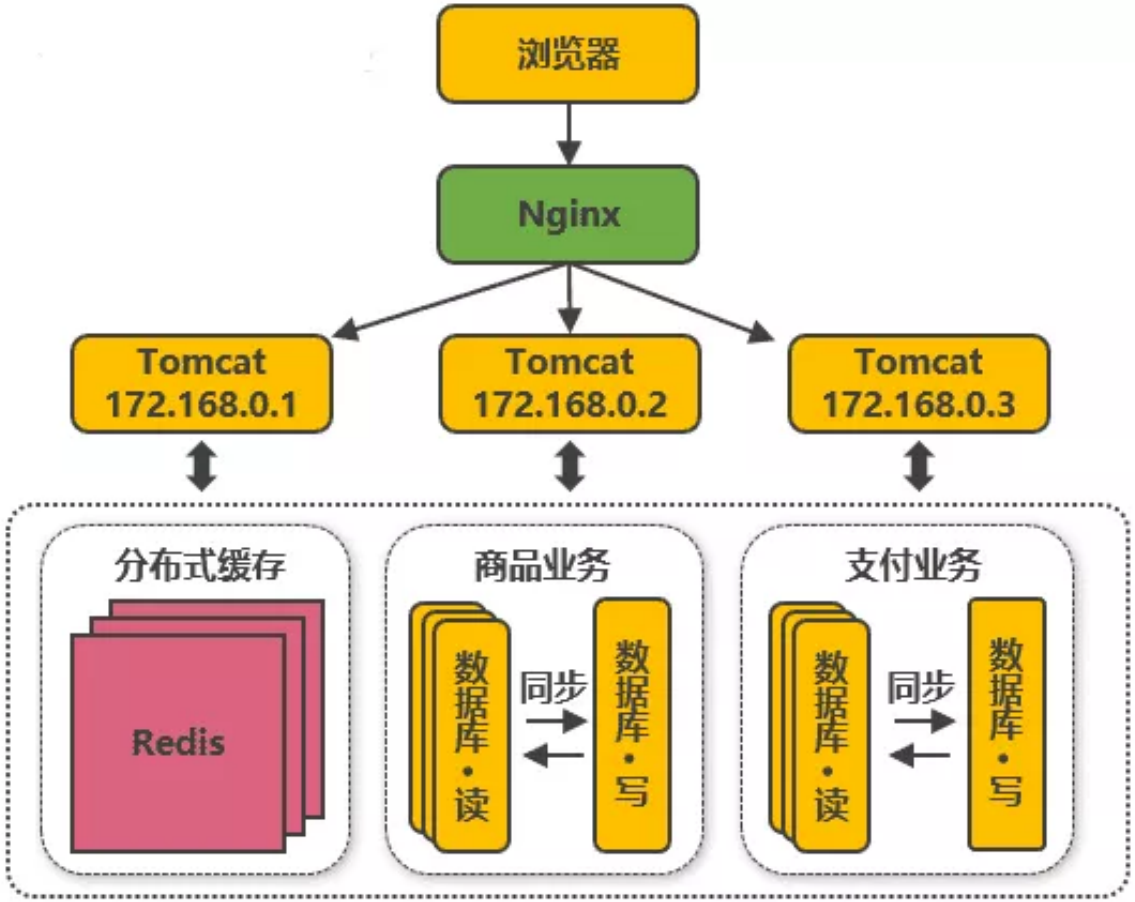


把數據庫劃分為讀庫和寫庫，讀庫可以有多個，通過同步機制把寫庫的數據同步到讀庫，對於需要查詢最新寫入數據場景，可通過在緩存中多寫一份，通過緩存獲得最新數據。

其中涉及的技術包括：Mycat，它是數據庫中間件，可通過它來組織數據庫的分離讀寫和分庫分錶，客戶端通過它來訪問下層數據庫，還會涉及數據同步，數據一致性的問題。

架構瓶頸：業務逐漸變多，不同業務之間的訪問量差距較大，不同業務直接競爭數據庫，相互影響性能。

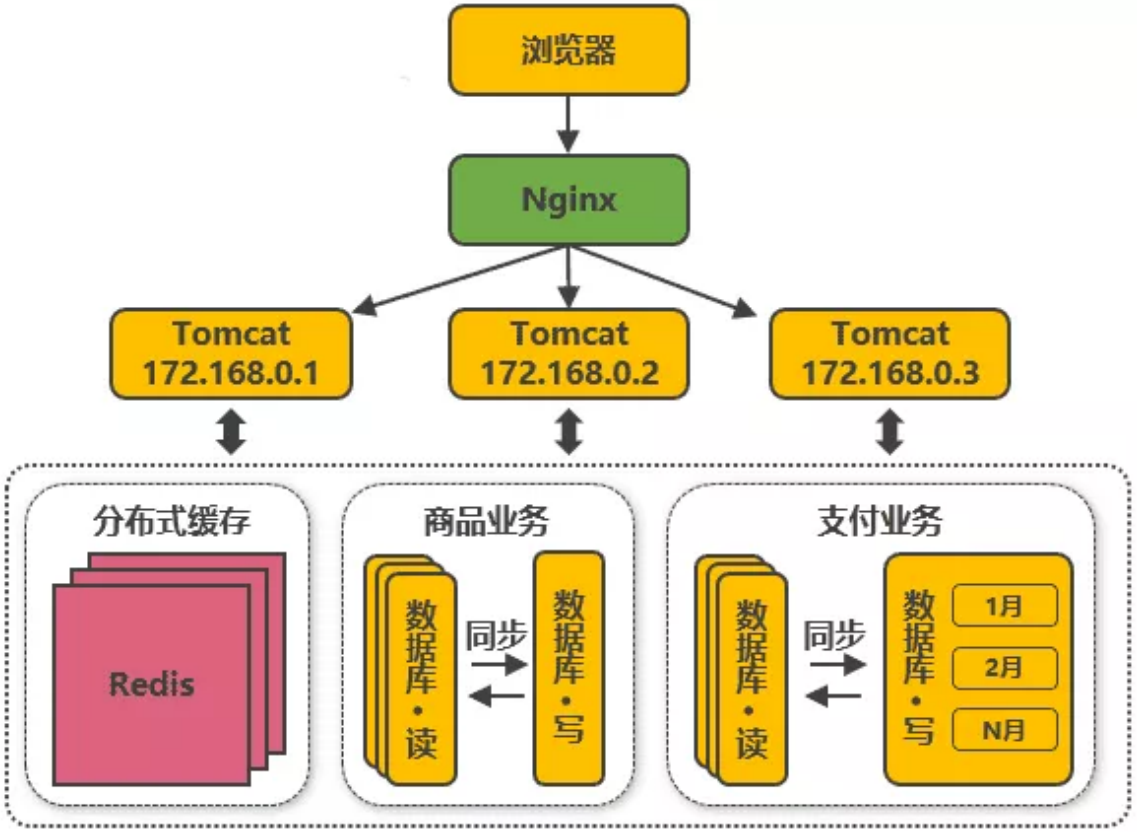
第五次演進：數據庫按業務分庫



把不同業務的數據保存到不同的數據庫中，使業務之間的資源競爭降低，對於訪問量大的業務，可以部署更多的服務器來支撐。這樣同時導致跨業務的表無法直接做關聯分析，需要通過其他途徑來解決，但這不是本文討論的重點，有興趣的可以自行搜索解決方案。

架構瓶頸：隨著用戶數的增長，單機的寫庫會逐漸會達到性能瓶頸。

第六次演進：把大表拆分為小表



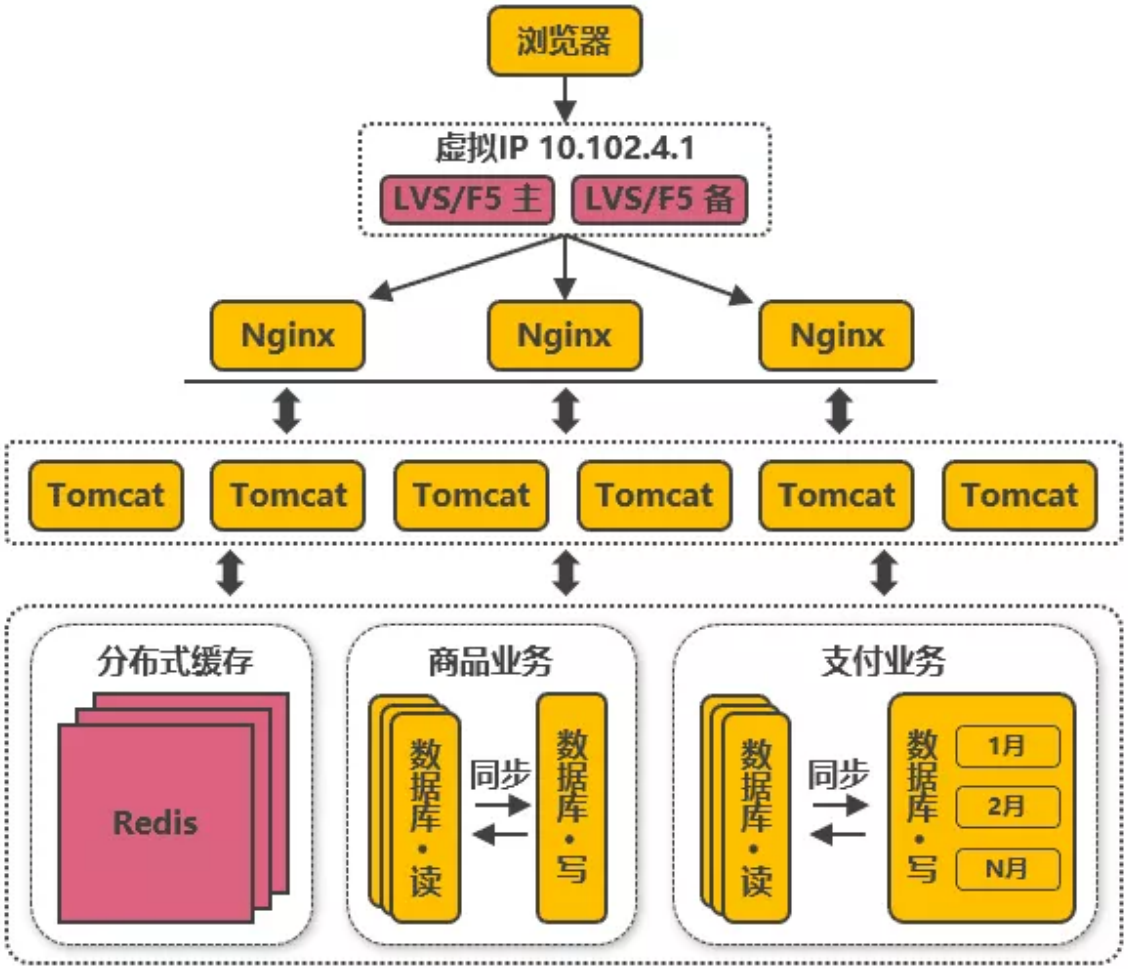
比如針對評論數據，可按照商品ID進行hash，路由到對應的表中存儲；針對支付記錄，可按照小時創建表，每個小時表繼續拆分為小表，使用用戶ID或記錄編號來路由數據。只要實時操作的表數據量足夠小，請求能夠足夠均勻的分發到多台服務器上的小表，那數據庫就能通過水平擴展的方式來提高性能。其中前面提到的Mycat也支持在大表拆分為小表情況下的訪問控制。

這種做法顯著的增加了數據庫運維的難度，對DBA的要求較高。數據庫設計到這種結構時，已經可以稱為分佈式數據庫，但是這只是一個邏輯的數據庫整體，數據庫裡不同的組成部分是由不同的組件單獨來實現的，如分庫分錶的管理和請求分發，由Mycat實現，SQL的解析由單機的數據庫實現，讀寫分離可能由網關和消息隊列來實現，查詢結果的匯總可能由數據庫接口層來實現等等，這種架構其實是MPP（大規模並行處理）架構的一類實現。

目前開源和商用都已經有不少MPP數據庫，開源中比較流行的有Greenplum、TiDB、Postgresql XC、HAWQ等，商用的如南大通用的GBase、睿帆科技的雪球DB、華為的LibrA等等，不同的MPP數據庫的側重點也不一樣，如TiDB更側重於分佈式OLTP場景，Greenplum更側重於分佈式OLAP場景，這些MPP數據庫基本都提供了類似Postgresql、Oracle、MySQL那樣的SQL標準支持能力，能把一個查詢解析為分佈式的執行計劃分發到每台機器上並行執行，最終由數據庫本身匯總數據進行返回，也提供了諸如權限管理、分庫分錶、事務、數據副本等能力，並且大多能夠支持100個節點以上的集群，大大降低了數據庫運維的成本，並且使數據庫也能夠實現水平擴展。推薦：大廠在用的分庫分錶方案，都在這了！

架構瓶頸：數據庫和Tomcat都能夠水平擴展，可支撐的並發大幅提高，隨著用戶數的增長，最終單機的Nginx會成為瓶頸。

第七次演進：使用LVS或F5來使多個Nginx負載均衡



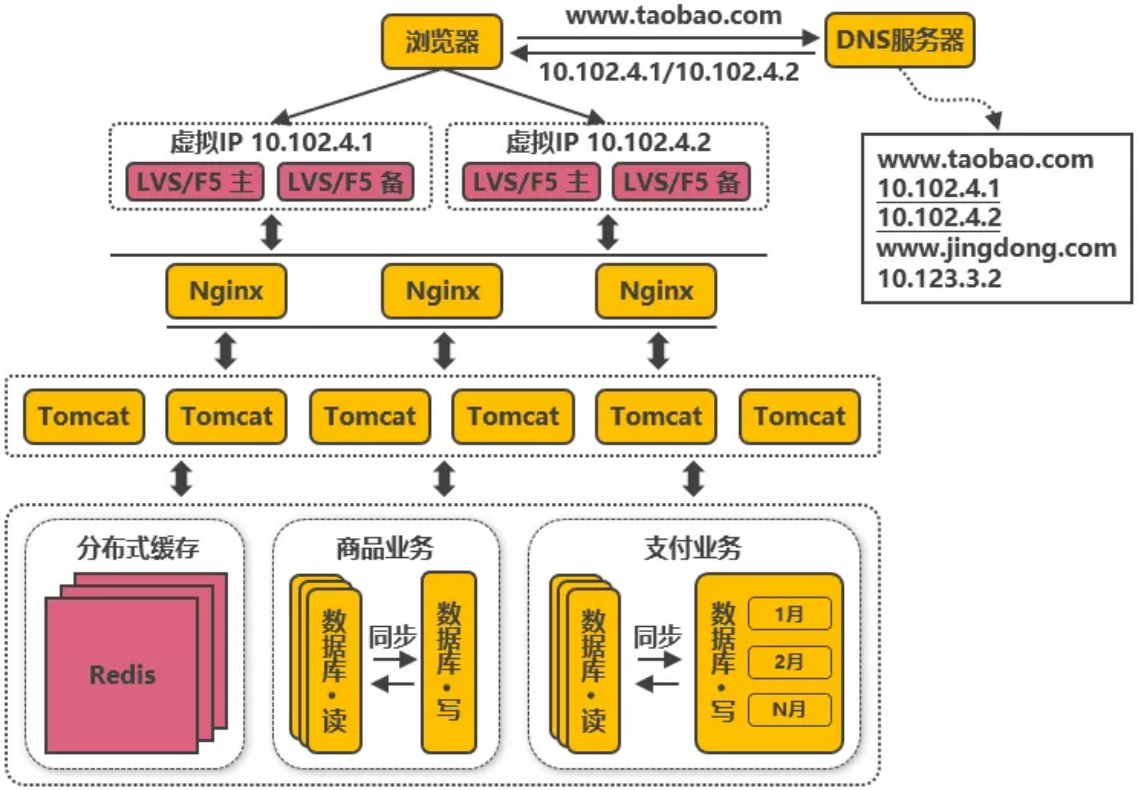
由於瓶頸在Nginx，因此無法通過兩層的Nginx來實現多個Nginx的負載均衡。圖中的LVS和F5是工作在網絡第四層的負載均衡解決方案，其中LVS是軟件，運行在操作系統內核態，可對TCP請求或更高層級的網絡協議進行轉發，因此支持的協議更豐富，並且性能也遠高於Nginx，可假設單機的LVS可支持幾十萬個並發的請求轉發；F5是一種負載均衡硬件，與LVS提供的能力類似，性能比LVS更高，但價格昂貴。由於LVS是單機版的軟件，若LVS所在服務器宕機則會導致整個後端系統都無法訪問，因此需要有備用節點。可使用keepalived軟件模擬出虛擬IP，然後把虛擬IP綁定到多

台LVS服務器上，瀏覽器訪問虛擬IP時，會被路由器重定向到真實的LVS服務器，當主LVS服務器宕機時，keepalived軟件會自動更新路由器中的路由表，把虛擬IP重定向到另外一台正常的LVS服務器，從而達到LVS服務器高可用的效果。

此處需要注意的是，上圖中從Nginx層到Tomcat層這樣畫並不代表全部Nginx都轉發請求到全部的Tomcat，在實際使用時，可能會是幾個Nginx下面接一部分的Tomcat，這些Nginx之間通過keepalived實現高可用，其他的Nginx接另外的Tomcat，這樣可接入的Tomcat數量就能成倍的增加。

架構瓶頸：由於LVS也是單機的，隨著並發數增長到幾十萬時，LVS服務器最終會達到瓶頸，此時用戶數達到千萬甚至上億級別，用戶分佈在不同的地區，與服務器機房距離不同，導致了訪問的延遲會明顯不同。

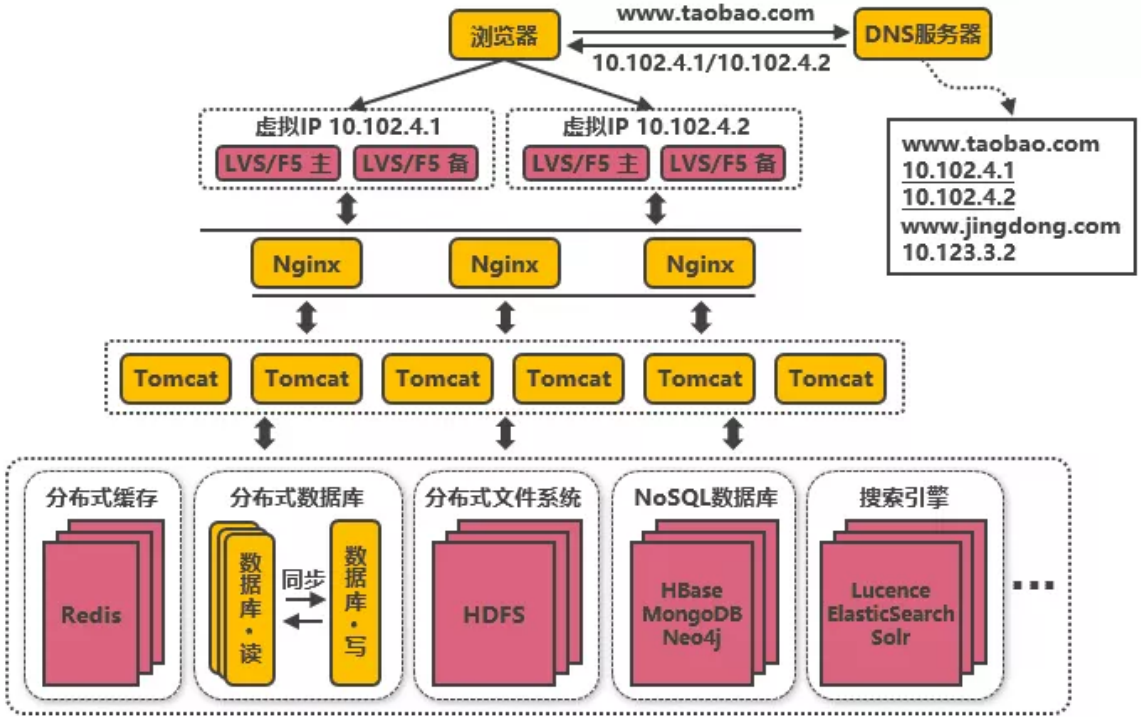
第八次演進：通過DNS輪詢實現機房間的負載均衡



在DNS服務器中可配置一個域名對應多個IP地址，每個IP地址對應到不同的機房裡的虛擬IP。當用戶訪問www.taobao.com時，DNS服務器會使用輪詢策略或其他策略，來選擇某個IP供用戶訪問。此方式能實現機房間的負載均衡，至此，系統可做到機房級別的水平擴展，千萬級到億級的並發量都可通過增加機房來解決，系統入口處的請求並發量不再是問題。

架構瓶頸：隨著數據的豐富程度和業務的發展，檢索、分析等需求越來越豐富，單單依靠數據庫無法解決如此豐富的需求。

第九次演進：引入NoSQL數據庫和搜索引擎等技術

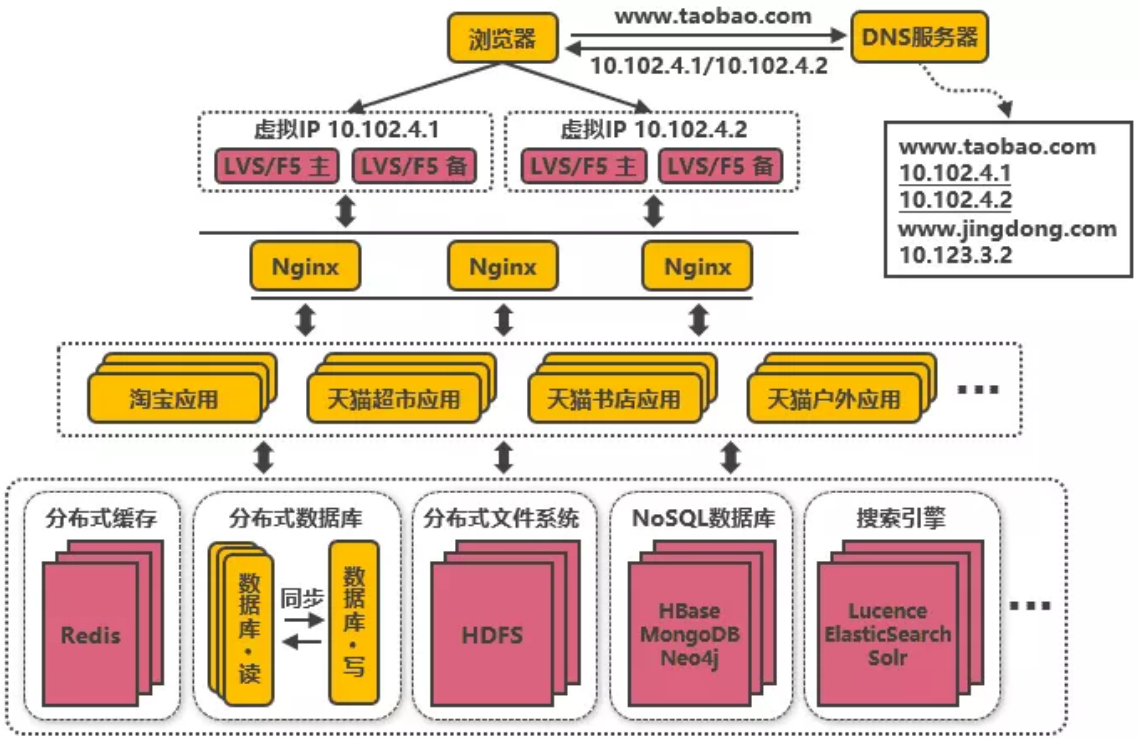


當數據庫中的數據多到一定規模時，數據庫就不適用於復雜的查詢了，往往只能滿足普通查詢的場景。對於統計報表場景，在數據量大時不一定能跑出結果，而且在跑複雜查詢時會導致其他查詢變慢，對於全文檢索、可變數據結構等場景，數據庫天生不適用。因此需要針對特定的場景，引入合適的解決方案。如對於海量文件存儲，可通過分佈式文件系統HDFS解決，對於key value類型的數據，可通過HBase和Redis等方案解決，對於全文檢索場景，可通過搜索引擎如ElasticSearch解決，對於多維分析場景，可通過Kylin或Druid等方案解決。

當然，引入更多組件同時會提高系統的複雜度，不同的組件保存的數據需要同步，需要考慮一致性的問題，需要有更多的運維手段來管理這些組件等。

架構瓶頸：引入更多組件解決了豐富的需求，業務維度能夠極大擴充，隨之而來的是一個應用中包含了太多的業務代碼，業務的升級迭代變得困難。

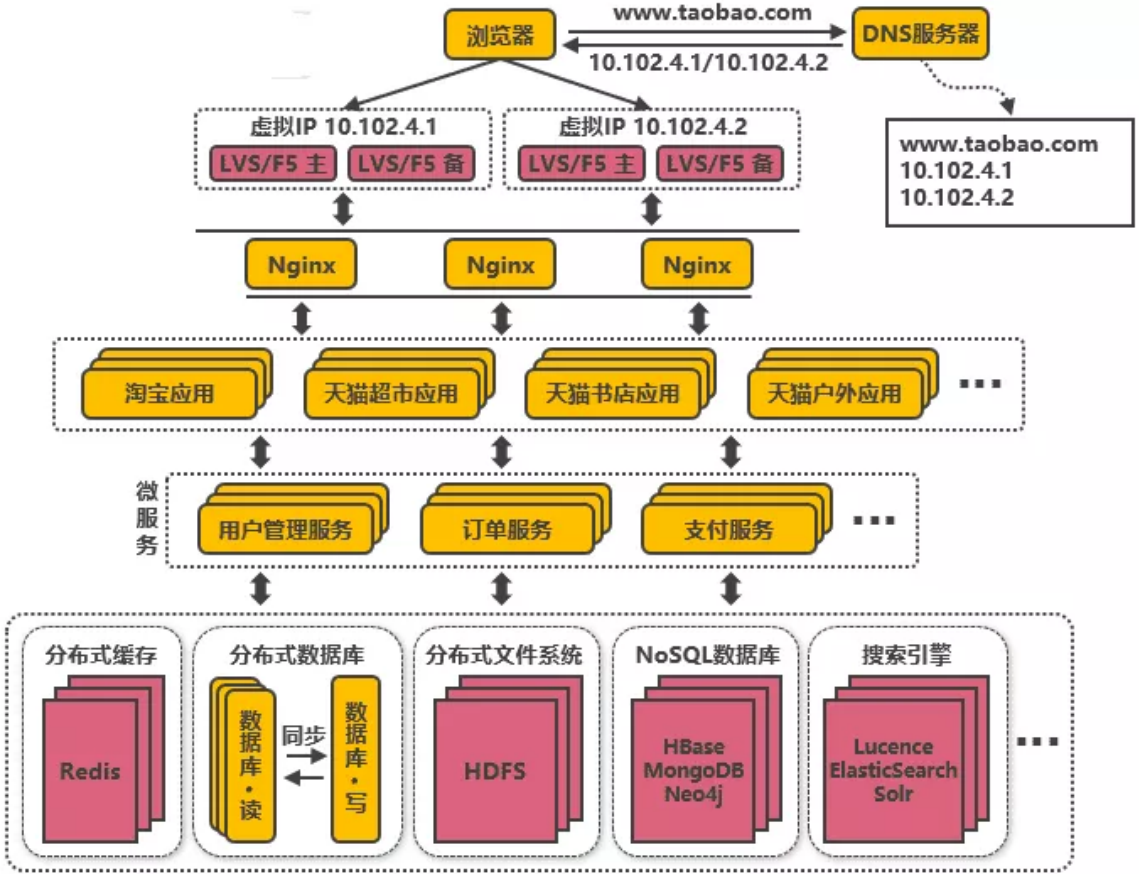
第十次演進：大應用拆分為小應用



按照業務板塊來劃分應用代碼，使單個應用的職責更清晰，相互之間可以做到獨立昇級迭代。這時候應用之間可能會涉及到一些公共配置，可以通過分佈式配置中心Zookeeper來解決。

架構瓶頸：不同應用之間存在共用的模塊，由應用單獨管理會導致相同代碼存在多份，導致公共功能升級時全部應用代碼都要跟著升級。

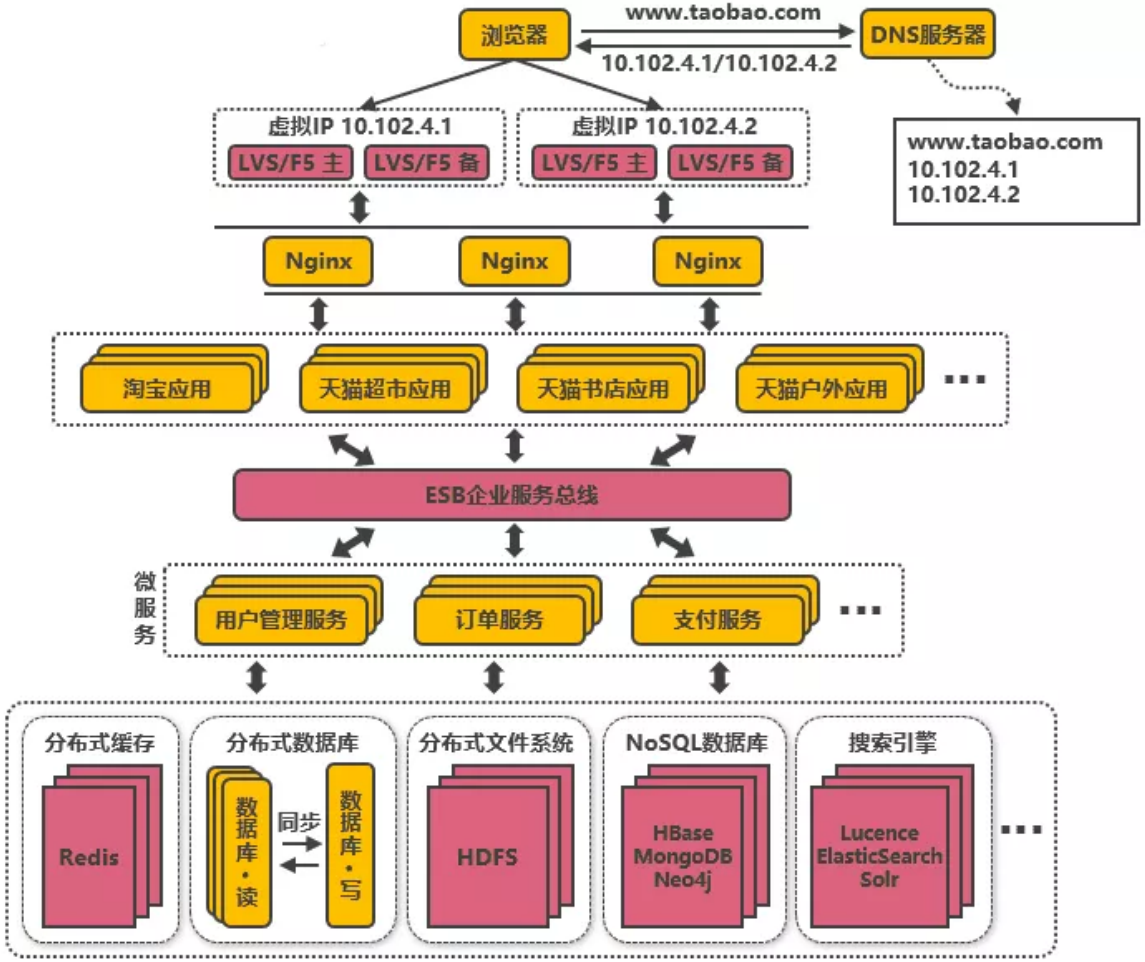
第十一次演進：復用的功能抽離成微服務



如用戶管理、訂單、支付、鑑權等功能在多個應用中都存在，那麼可以把這些功能的代碼單獨抽取出來形成一個單獨的服務來管理，這樣的服務就是所謂的微服務，應用和服務之間通過HTTP、TCP或RPC請求等多種方式來訪問公共服務，每個單獨的服務都可以由單獨的團隊來管理。此外，可以通過Dubbo、SpringCloud等框架實現服務治理、限流、熔斷、降級等功能，提高服務的穩定性和可用性。

架構瓶頸：不同服務的接口訪問方式不同，應用代碼需要適配多種訪問方式才能使用服務，此外，應用訪問服務，服務之間也可能相互訪問，調用鏈將會變得非常複雜，邏輯變得混亂。

第十二次演進：引入企業服務總線ESB屏蔽服務接口的訪問差異



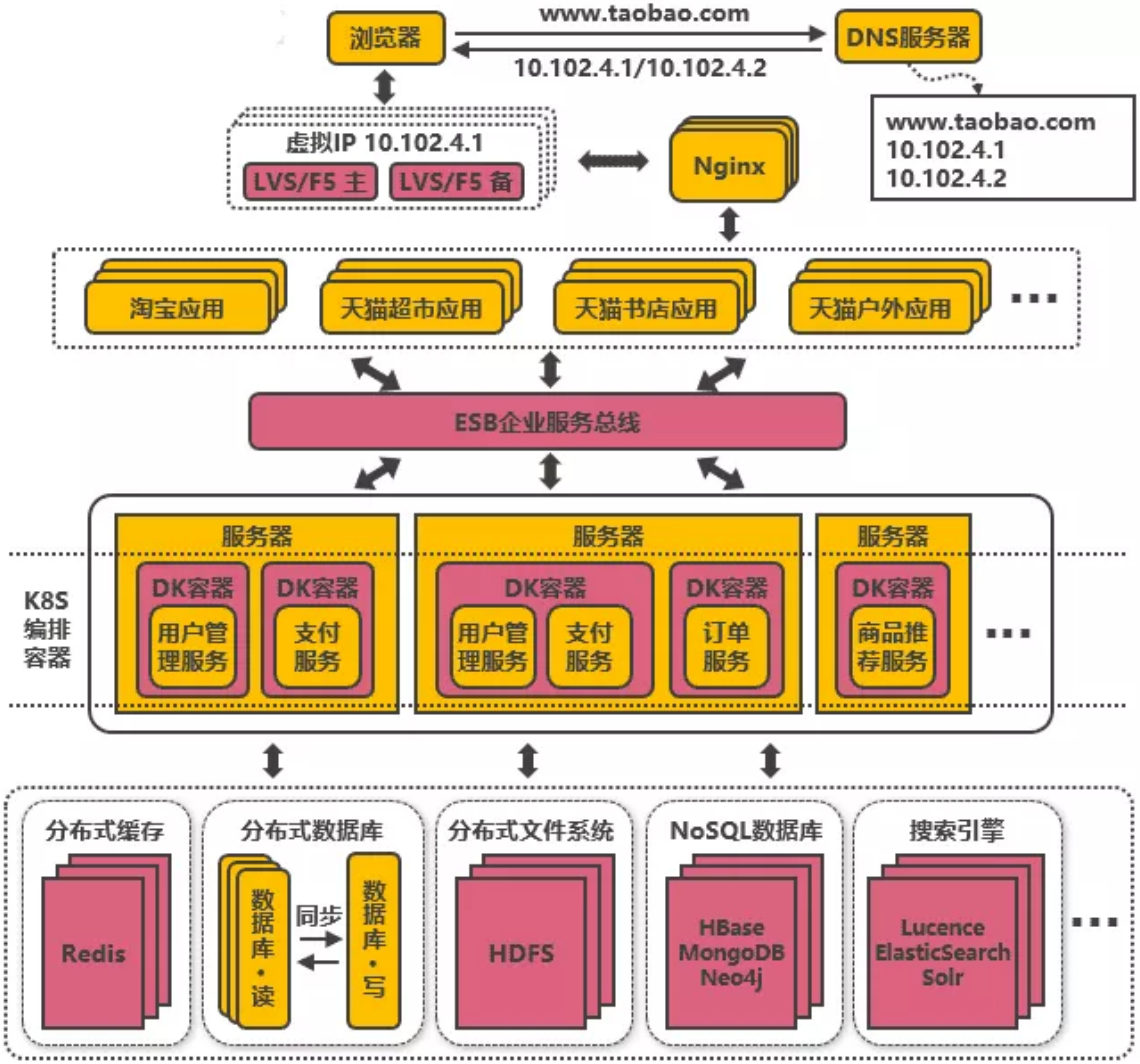
通過ESB統一進行訪問協議轉換，應用統一通過ESB來訪問後端服務，服務與服務之間也通過ESB來相互調用，以此降低系統的耦合程度。

這種單個應用拆分為多個應用，公共服務單獨抽取出來來管理，並使用企業消息總線來解除服務之間耦合問題的架構，就是所謂的SOA（面向服務）架構，這種架構與微服務架構容易混淆，因為表現形式十分相似。

個人理解，微服務架構更多是指把系統裡的公共服務抽取出來單獨運維管理的思想，而SOA架構則是指一種拆分服務並使服務接口訪問變得統一的架構思想，SOA架構中包含了微服務的思想。

架構瓶頸：業務不斷發展，應用和服務都會不斷變多，應用和服務的部署變得複雜，同一台服務器上部署多個服務還要解決運行環境衝突的問題，此外，對於如大促這類需要動態擴縮容的場景，需要水平擴展服務的性能，就需要在新增的服務上準備運行環境，部署服務等，運維將變得十分困難。

第十三次演進：引入容器化技術實現運行環境隔離與動態服務管理

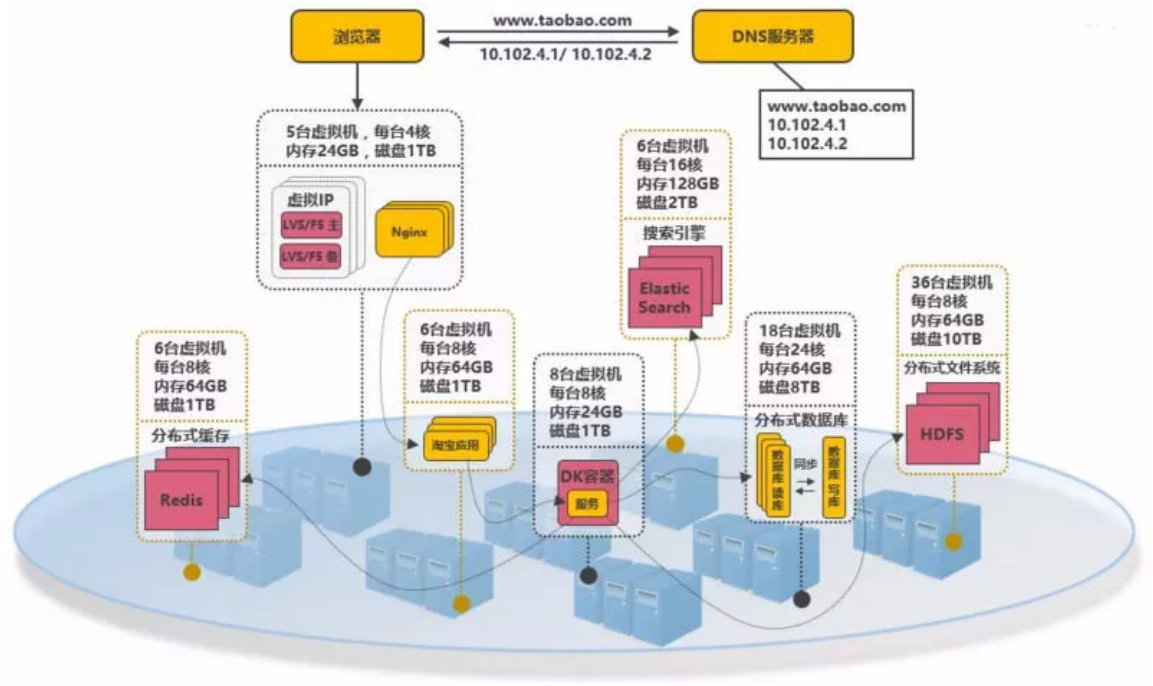


目前最流行的容器化技術是Docker，最流行的容器管理服務是Kubernetes(K8S)，應用/服務可以打包為Docker鏡像，通過K8S來動態分發和部署鏡像。Docker鏡像可理解為一個能運行你的應用/服務的最小的操作系統，裡面放著應用/服務的運行代碼，運行環境根據實際的需要設置好。把整個“操作系統”打包為一個鏡像後，就可以分發到需要部署相關服務的機器上，直接啟動Docker鏡像就可以把服務起起來，使服務的部署和運維變得簡單。

在大促的之前，可以在現有的機器集群上劃分出服務器來啟動Docker鏡像，增強服務的性能，大促過後就可以關閉鏡像，對機器上的其他服務不造成影響（在第18節之前，服務運行在新增機器上需要修改系統配置來適配服務，這會導致機器上其他服務需要的運行環境被破壞）。

架構瓶頸：使用容器化技術後服務動態擴縮容問題得以解決，但是機器還是需要公司自身來管理，在非大促的時候，還是需要閒置著大量的機器資源來應對大促，機器自身成本和運維成本都極高，資源利用率低。

第十四次演進：以雲平台承載系統



系统可部署到公有云上，利用公有云的海量機器資源，解決動態硬件資源的問題，在大促的時間段裡，在雲平台中臨時申請更多的資源，結合Docker和K8S來快速部署服務，在大促結束後釋放資源，真正做到按需付費，資源利用率大大提高，同時大大降低了運維成本。

所谓的云平台，就是把海量机器资源，通过统一的资源管理，抽象为一个资源整体，在之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，用户不需要关系应用内部使用了什么技术，就能够解决需求（如音视频转码服务、邮件服务、个人博客等）。

在云平台中会涉及如下几个概念：

1）IaaS：基础设施即服务。对应于上面所说的机器资源统一为资源整体，可动态申请硬件资源的层面；

2）PaaS：平台即服务。对应于上面所说的提供常用的技术组件方便系统的开发和维护；

3）SaaS：软件即服务。对应于上面所说的提供开发好的应用或服务，按功能或性能要求付费。

至此：以上所提到的从高并发访问问题，到服务的架构和系统实施的层面都有了各自的解决方案。但同时也应该意识到，在上面的介绍中，其实是有意忽略了诸如跨机房数据同步、分布式事务实现等等的实际问题，这些问题以后有机会再拿出来单独讨论。

架构设计经验小结

1）架构的调整是否必须按照上述演变路径进行？

不是的，以上所说的架构演变顺序只是针对某个侧面进行单独的改进，在实际场景中，可能同一时间会有几个问题需要解决，或者可能先达到瓶颈的是另外的方面，这时候就应该按照实际问题实际解决。如在政府类的并发量可能不大，但业务可能很丰富的场景，高并发就不是重点解决的问题，此时优先需要的可能会是丰富需求的解决方案。

2）对于将要实施的系统，架构应该设计到什么程度？

对于单次实施并且性能指标明确的系统，架构设计到能够支持系统的性能指标要求就足够了，但要留有扩展架构的接口以便不备之需。对于不断发展的系统，如电商平台，应设计到能满足下一阶段用户量和性能指标要求的程度，并根据业务的增长不断的迭代升级架构，以支持更高的并发和更丰富的业务。

3）服务端架构和大数据架构有什么区别？

所谓的“大数据”其实是海量数据采集清洗转换、数据存储、数据分析、数据服务等场景解决方案的一个统称，在每一个场景都包含了多种可选的技术，如数据采集有Flume、Sqoop、Kettle等，数据存储有分布式文件系统HDFS、FastDFS，NoSQL数据库HBase、MongoDB等，数据分析有Spark技术栈、机器学习算法等。

总的来说大数据架构就是根据业务的需求，整合各种大数据组件组合而成的架构，一般会提供分布式存储、分布式计算、多维分析、数据仓库、机器学习算法等能力。而服务端架构更多指的是应用组织层面的架构，底层能力往往是由大数据架构来提供。

4）有没有一些架构设计的原则？

a. N+1设计：系统中的每个组件都应做到没有单点故障；

b. 回滚设计：确保系统可以向前兼容，在系统升级时应能有办法回滚版本；

c. 禁用设计：应该提供控制具体功能是否可用的配置，在系统出现故障时能够快速下线功能；

d. 监控设计：在设计阶段就要考虑监控的手段；

e. 多活数据中心设计：若系统需要极高的高可用，应考虑在多地实施数据中心进行多活，至少在一个机房断电的情况下系统依然可用；

f. 采用成熟的技术：刚开发的或开源的技术往往存在很多隐藏的bug，出了问题没有商业支持可能会是一个灾难；

g. 资源隔离设计：应避免单一业务占用全部资源；

h. 架构应能水平扩展：系统只有做到能水平扩展，才能有效避免瓶颈问题；

i. 非核心则购买：非核心功能若需要占用大量的研发资源才能解决，则考虑购买成熟的产品；

j. 使用商用硬件：商用硬件能有效降低硬件故障的机率；

k. 快速迭代：系统应该快速开发小功能模块，尽快上线进行验证，早日发现问题大大降低系统交付的风险；

l. 无状态设计：服务接口应该做成无状态的，当前接口的访问不依赖于接口上次访问的状态。

-END-

如果看到这里，说明你喜欢这篇文章，请转发、点赞。微信搜索「web_resource」，关注后回复「进群」或者扫描下方二维码即可进入无广告交流群。

↓扫描二维码进群↓

推荐阅读

1. 10 个让你笑的合不拢嘴的 GitHub 项目
2. 当我遵循了这 16 条规范写代码
3. 理解 IntelliJ IDEA 的项目配置和 Web 部署
4. Java 开发中常用的 4 种加密方法
5. 團隊開發中Git最佳實踐

學Java，請關注公眾號：Java後端

喜歡文章，點個在看

閱讀原文