详解主流分布式架构选型与高可用设计

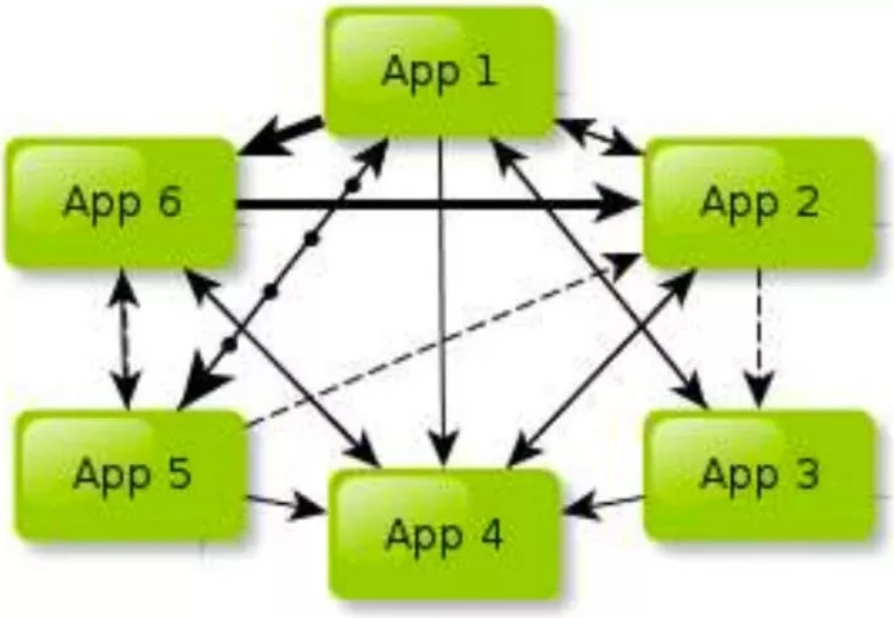
阿豪说 DBAplus社群 今天

本文主要分享目前主流的分布式架构、分布式架构中常见理论以及如何才能设计出高可用的分布式架构。

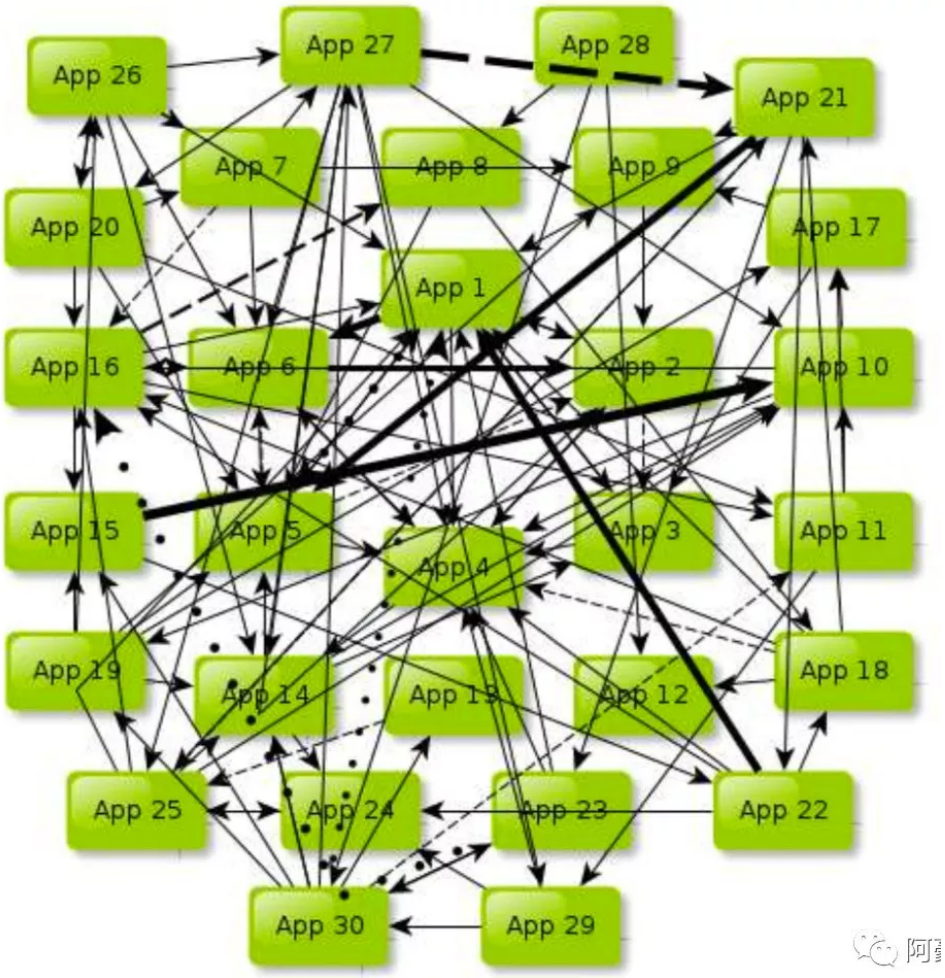
分布式架构中，SOA和微服务架构是最常见两种分布式架构，而且目前服务网格的概念也越来越火了，那本文就先从这些常见架构开始说起。

一、SOA架构解析

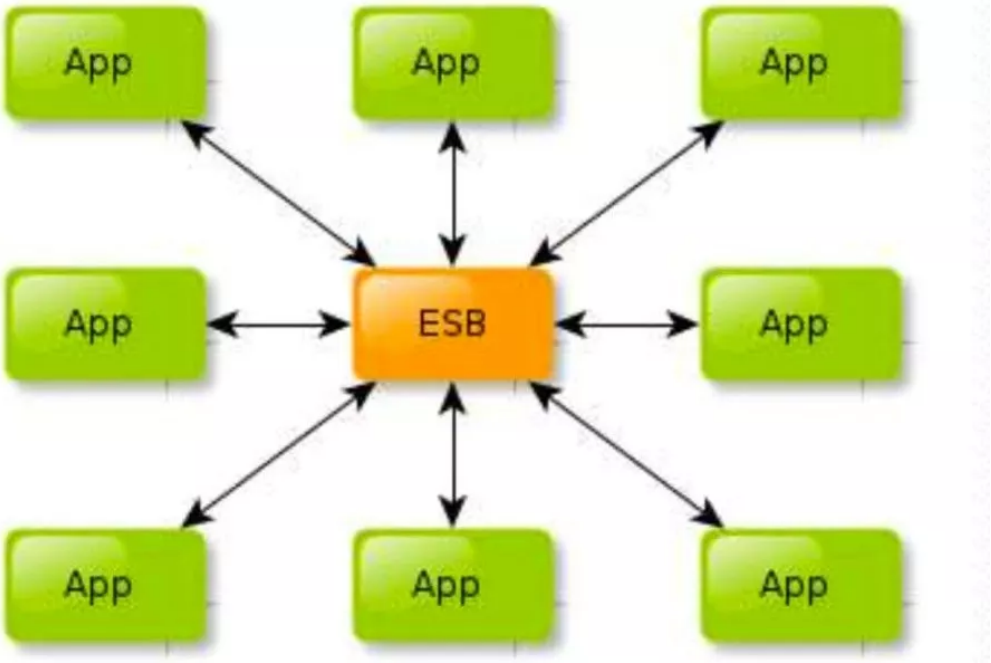
SOA（Service Oriented Architecture）的中文释义为“面向服务的架构”，是一种设计理念，其中包含多个服务，服务之间通过相互依赖最终提供一系列完整的功能。各个服务通常以独立的形式部署运行，服务之间通过网络进行调用。架构图如下：



跟SOA相提并论的还有一个ESB（企业服务总线），简单来说ESB就是一根管道，用来连接各个服务节点。ESB的存在是为了集成基于不同协议的不同服务，ESB做了消息的转化、解释以及路由的工作，以此来让不同的服务互联互通；随着我们业务的越来越复杂，会发现服务越来越多，SOA架构下，它们的调用关系会变成如下形式：



很显然，这样不是我们所想要的，那这时候如果我们引入ESB的概念，项目调用就又会很清晰，如下：



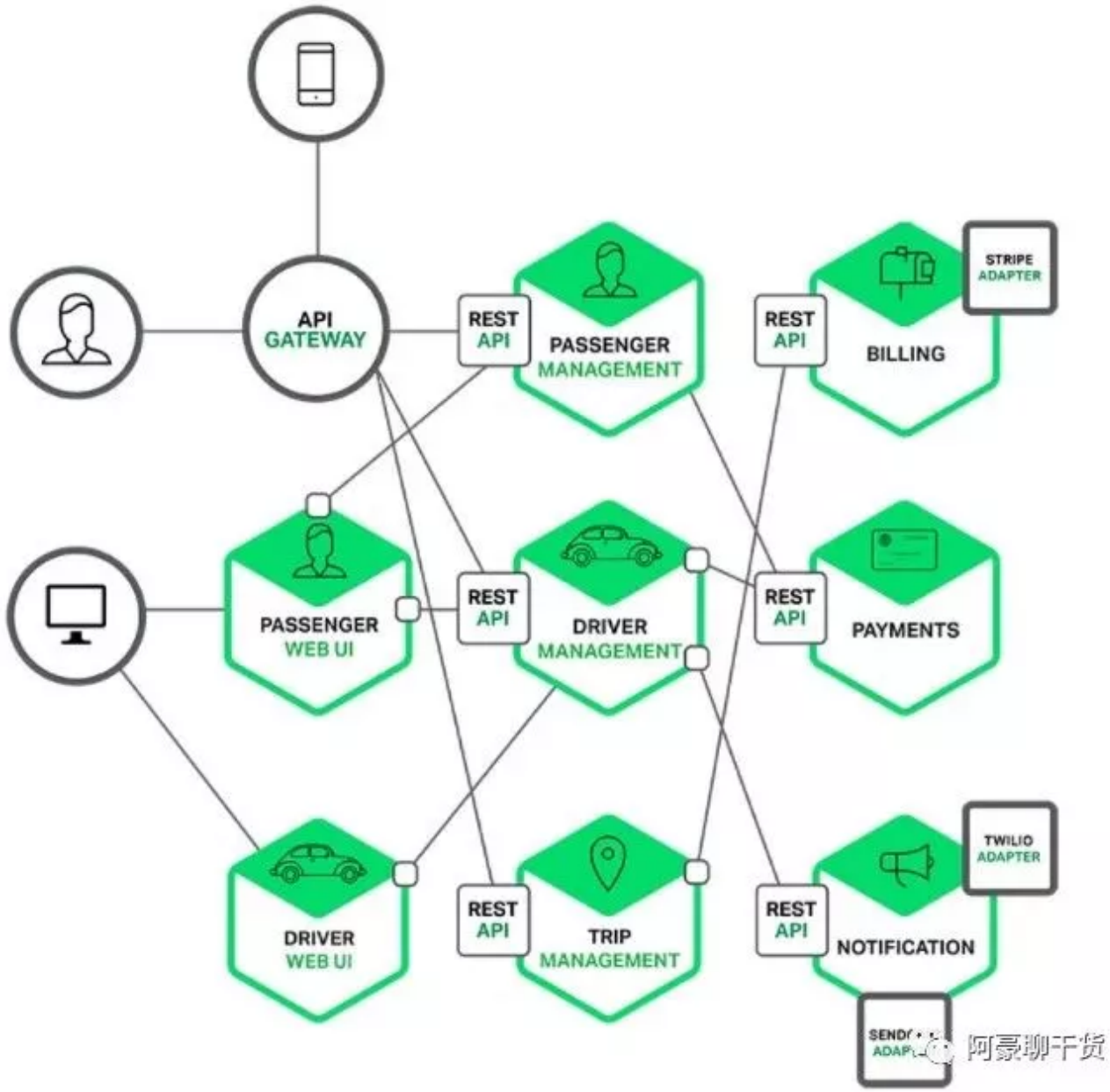
SOA所要解决的核心问题

- 系统间的集成：我们站在系统的角度来看，首先要解决各个系统间的通信问题，目的是将原先系统间散乱、无规划的网状结构，梳理成规整、可治理的星形结构。这一步的实现往往需要引入一些概念和规范，比如ESB、以及技术规范、服务管理规范，这一步解决的核心问题是**有序**。
- 系统的服务化：我们站在功能的角度，需要把业务逻辑抽象成可复用、可组装的服务，从而通过服务的编排实现业务的快速再生，目的是要把原先固有的业务功能抽象设计为通用的业务服务、实现业务逻辑的快速复用，这步要解决的核心问题是**复用**。
- 业务的服务化：我们站在企业的角度，要把企业职能抽象成可复用、可组装的服务，就要把原先职能化的企业架构转变为服务化的企业架构，以便进一步提升企业的对外服务的能力。前面两步都是从技术层面来解决系统调用、系统功能复用的问题。而本步骤则是以业务驱动把一个业务单元封装成一项服务，要解决的核心问题是**高效**。

二、微服务架构解析

微服务架构和SOA架构非常类似，微服务只是的SOA升华，只不过微服务架构强调的是“业务需要彻底的组件化及服务化”，原单个业务系统会被拆分为多个可以独立开发、设计、部署运行的小应用。这些小应用间通过服务化完成交互和集成。组件表示的就是一个可以独立更换和升级的单元，就像PC中的CPU、内存、显卡、硬盘一样，独立且可以更换升级而不影响其他单元。

若我们把PC中的各个组件以服务的方式构建，那么这台PC只需要维护主板（可以理解为ESB）和一些必要的外部设备就可以。CPU、内存、硬盘等都是以组件方式提供服务，例如PC需要调用CPU做计算处理，只需知道CPU这个组件的地址就可以了。



微服务的特征：

- 通过服务实现组件化
- 按业务能力来划分服务和开发团队
- 去中心化
- 基础设施自动化（DevOps、自动化部署）

三、SOA和微服务架构的差别

微服务不再强调传统SOA架构里面比较重的ESB企业服务总线，同时以SOA的思想进入到单个业务系统内部实现真正的组件化。

Docker容器技术的出现，为微服务提供了非常便利的条件，比如更小的部署单元，每个服务可以通过类似Spring Boot或者Node等技术独立运行。

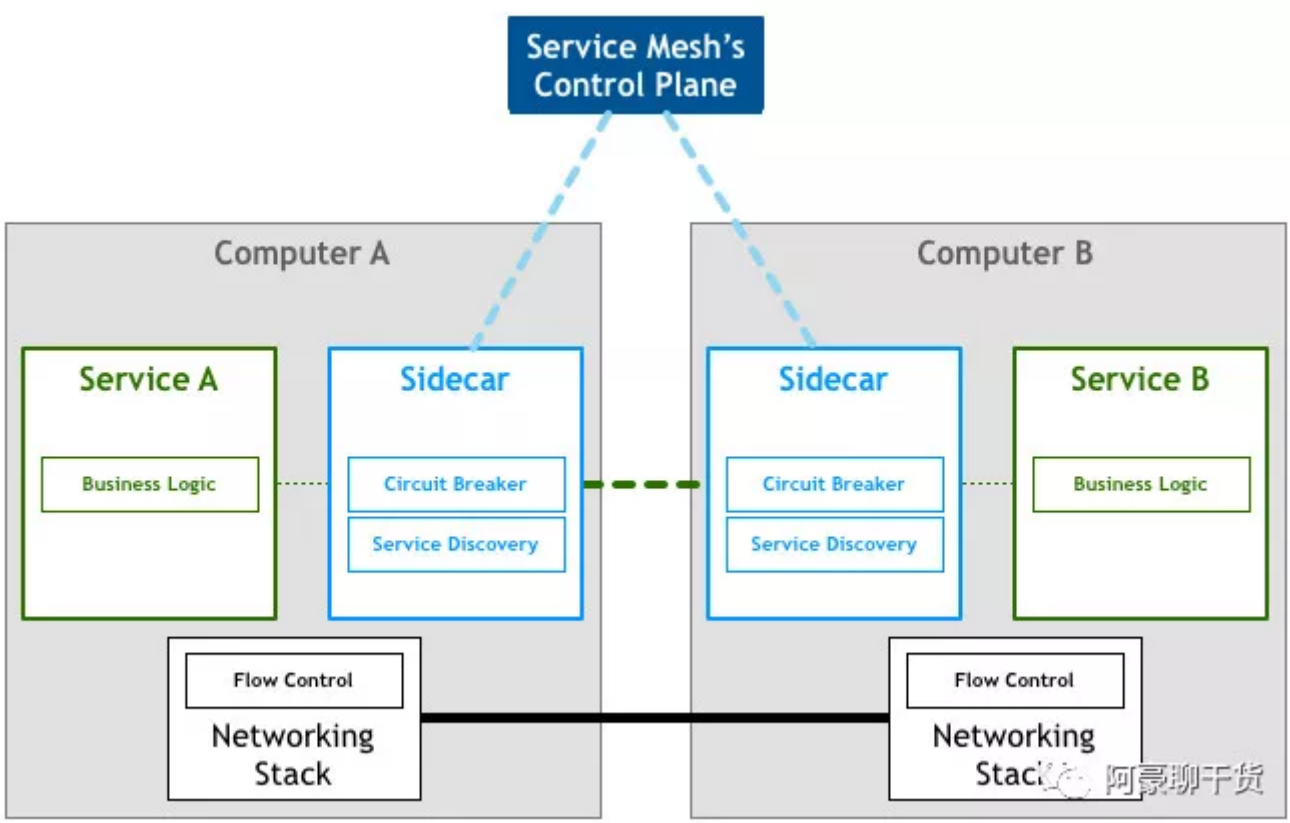
还有一个点大家应该可以分析出来，SOA注重的是系统集成，而微服务关注的是完全分离。

四、服务网格架构解析

17 年年底，非侵入式的 Service Mesh 技术慢慢走向了成熟。Service Mesh ，中文释义“服务网格”，作为服务间通信的基础设施层在系统中存在。

如果要用一句话来解释什么叫 Service Mesh，我们可以将它比作是应用程序或者说微服务间的TCP/IP，负责服务间的网络调用、熔断、限流和监控。

我们都知道在编写应用程序时程序猿一般都无须关心TCP/IP这一层（比如提供HTTP协议的Restful应用），同样如果使用服务网格我们也就不需要关心服务间的那些原来是由应用程序或者其他框架实现的事情（熔断、限流、监控等），现在只要交给Service Mesh就可以了。服务网格架构图如下：



目前流行的Service Mesh开源软件有Linkerd、Envoy和Istio，而最近Buoyant（开源Linkerd的公司）又发布了基于Kubernetes的Service Mesh开源项目Conduit。

关于微服务和服务网格的区别，我这样理解：微服务更注重服务之间的生态，专注于服务治理等方面，而服务网格更专注于服务之间的通信，以及和DevOps更好的结合等。

服务网格的特征：

- 应用程序间通讯的中间层
- 轻量级网络代理
- 应用程序无感知
- 解耦应用程序的重试/超时、监控、追踪和服务发现

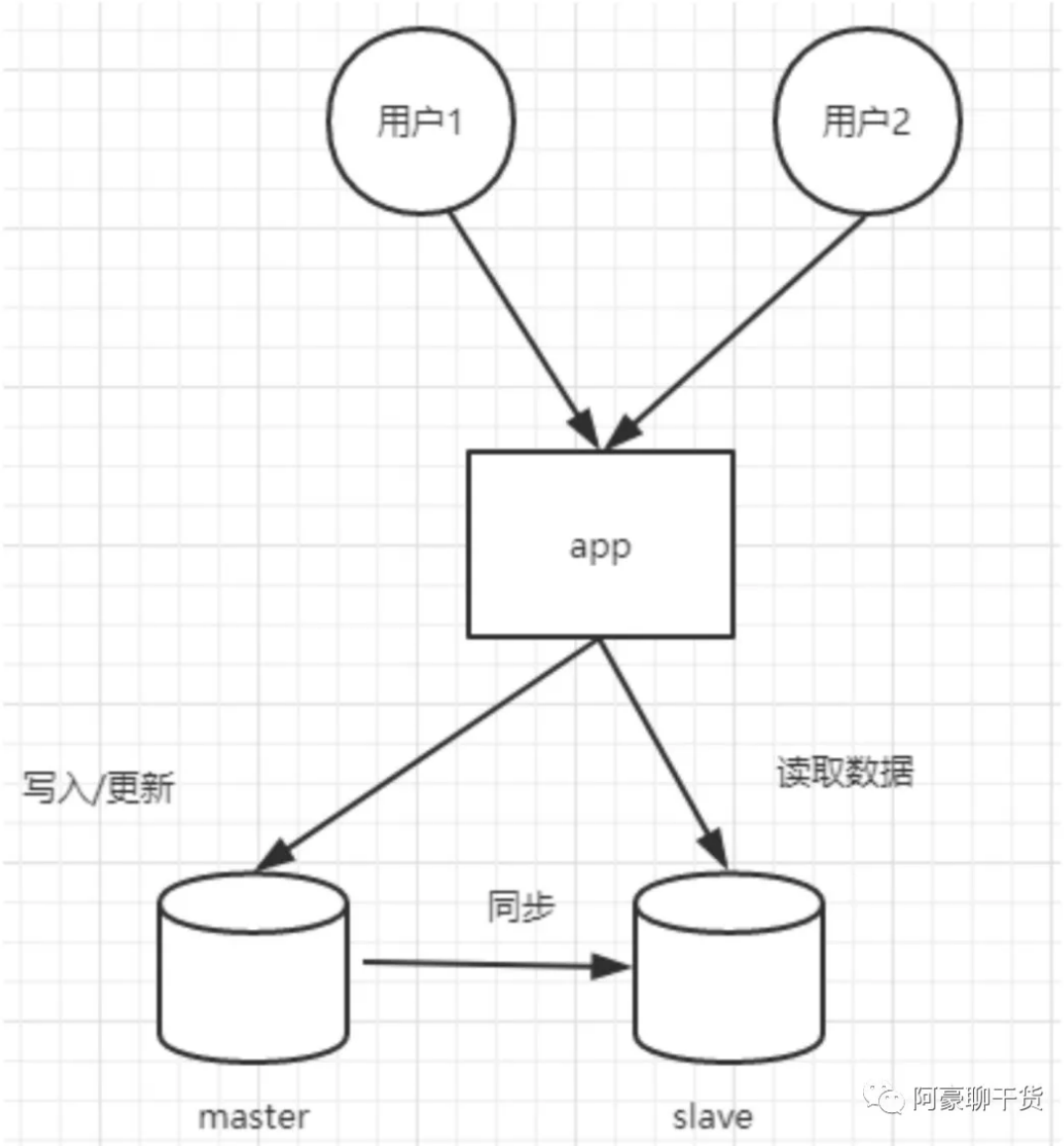
五、分布式架构的基本理论

在说CAP、BASE理论之前，我们先要了解下分布式一致性的问题。实际上对于不同业务的服务，我们对数据一致性的要求是不一样的，如12306，它要求数据的严格一致性，不能把票卖给用户以后却发现没有座位了；再比如银行转账，我们通过银行转账的时候，一般都会收到一个提示：转账申请将会在 24 小时内到账。实际上这个场景满足的是最终钱只要转账成功了即可，同时如果钱没汇出去还要保证资金不丢失。所以说，用户在使用不同的服务的时候对数据一致性的要求是不一样的。

关于分布式一致性问题

分布式系统中要解决的一个非常重要的问题就是数据的复制。在我们的日常开发经验中，相信大多数开发人员都遇过这样的问题：在做数据库读写分离的场景中，假设客户端A将系统中的一个值V由V1变更为V2，但客户端B无法立即读取到V的最新值，需要在一段时间之后才能读取到。

这再正常不过了，因为数据库复制之间是存在延时的。



所谓分布式一致性的问题，就是指在分布式环境中引入数据复制机制后，不同数据节点之间可能会出现的、且无法依靠计算机应用程序自身解决的数据不一致的情况。

简单来说，数据一致性就是指在对一个副本数据进行变更的时候，必须确保也能够更新其它的副本，否则不同副本之间的数据将出现不一致。

那么如何去解决这个问题呢？

按照正常的思路，我们可能会想到既然是网络延迟导致的问题，那么我们就把同步动作进行阻塞，用户2在查询的时候必须要等数据同步完成以后再来做。但这个方案会非常影响性能。如果同步的数据比较多或比较频繁，那么阻塞操作可能会导致整个新系统不可用。故我们没有办法找到一种既能够满足数据一致性、又不影响系统性能的方案，所以就诞生了一个一致性的级别：

- 强一致性：这种一致性级别是最符合用户直觉的，它要求系统写入的是什么，读出来的也要是什么，用户体验好，但实现起来往往对系统的性能影响较大；

- 弱一致性：这种一致性级别约束了系统在写入成功后，不保证立即可以读到写入的值，也不保证多久之后数据能够达到一致，但会尽可能地保证到某个时间级别（如秒级别）后，数据能够达到一致状态；
- 最终一致性：最终一致性其实是弱一致性的一个特例，系统会保证在一定时间内，能够达到数据一致的状态。这里之所以将最终一致性单独提出来，是因为它是弱一致性中非常推崇的一种一致性模型，也是业界在大型分布式系统的数据一致性上用的比较多的一致性模型。

CAP 理论

它是一个经典的分布式系统理论。CAP 理论告诉我们：一个分布式系统不可能同时满足一致性（C：Consistency）、可用性（A：Availability）及分区容错性（P：Partition tolerance）这三个基本要求，最多只能同时满足其中两项。

CAP理论在互联网界有着广泛的知名度，也被称为“帽子理论”，它是由Eric Brewer教授在2000年举行的ACM研讨会提出的一个著名猜想：一致性（Consistency）、可用性（Availability）、分区容错 （Partition-tolerance）三者无法在分布式系统中被同时满足，并且最多只能满足两个。

- 一致性：所有节点上的数据时刻保持同步
- 可用性：每个请求都能接收一个响应，无论响应成功或失败
- 分区容错：系统应该持续提供服务，即使系统内部（某个节点分区）有消息丢失。比如交换机失败、网址网络被分成几个子网，形成脑裂、服务器发生网络延迟或死机，导致某些Server与集群中的其他机器失去联系。

分区是导致分布式系统可靠性问题的固有特性，从本质上来看，CAP理论的准确描述不应该是从3个特性中选取两个，所以我们只能被迫适应，根本没有选择权。CAP并不是一个普适性原理和指导思想，它仅适用于原子读写的NoSQL场景中，并不适用于数据库系统。

BASE 理论

从前面的分析中我们知道：在分布式（数据库分片或分库存在的多个实例上）前提下，CAP理论并不适合数据库事务，因为更新一些错误的数据而导致的失败，无论使用什么高可用方案都是徒劳的，因为数据发生了无法修正的错误。

此外，XA事务虽然保证了数据库在分布式系统下的 ACID （原子性、一致性、隔离性、持久性）特性，但同时也带来了一 些性能方面的代价，对于并发和响应时间要求都比较高的电商平台来说，是很难接受的。

eBay尝试了另外一条完全不同的路，放宽了数据库事务的ACID要求，提出了一套名为BASE的新准则。BASE全称为Basically Available，Soft-state，Eventually Consistent。系统基本可用、软状态、数据最终一致性。相对于CAP来说，它大大降低了我们对系统的要求。

Basically Available(基本可用)

表示在分布式系统出现不可预知的故障时，允许瞬时部分可用性：

- 比如我们在淘宝上搜索商品，正常情况下是在0.5s内返回查询结果，但是由于后端的系统故障导致查询响应时间变成了2s；
- 再比如数据库采用分片模式，100W个用户数据分在5个数据库实例上，如果破坏了一个实例，那么可用性还有80%，也就是80%的用户都可以登录，系统仍然可用；
- 电商大促时，为了应对访问量激增，部分用户可能会被引导到降级页面，服务层也可能只提供降级服务。这就是损失部分可用性的体现。

Soft-state(软状态)

表示系统中的数据存在中间状态，并且这个中间状态的存在不会影响系统的整体可用性，也就是表示系统允许在不同节点的数据副本之间进行数据同步过程中存在延时；比如订单状态，有一个待支付、支付中、 支付成功、支付失败，那么支付中就是一个中间状态，这个中间状态在支付成功以后，在支付表中的状态同步给订单状态之前，中间会存在一个时间内的不一致。

Eventually consistent(数据的最终一致性)

表示的是所有数据副本在一段时间的同步后最终都能达到一个一致的状态，因此最终一致性的本质是要保证数据最终达到一致，而不需要实时保证系统数据的强一致。

BASE理论的核心思想是：即使无法做到强一致性，但每个应用都可以根据自身业务特点，采用适当的方式来使系统达到最终一致性。

六、分布式架构下的高可用设计

避免单点故障

- 负载均衡技术（Failover、选址、硬件负载、软件负载、去中心化的软件负载（Gossip（Redis- Cluster））））
- 热备（Linux HA）
- 多机房（同城灾备、异地灾备）

应用的高可用性

- 故障监控（系统监控（CPU、内存）/链路监控/日志监控）自动预警

- 应用的容错设计、（服务降级、限流）自我保护能力
- 数据量（数据分片、读写分离）

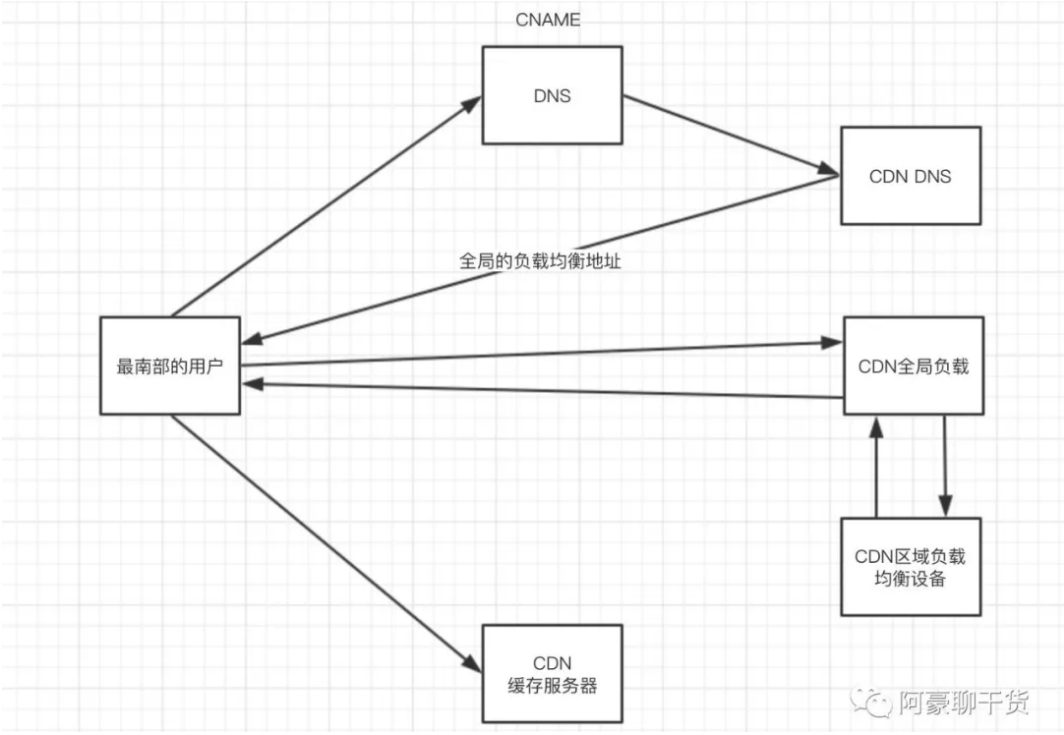
分布式架构下的可伸缩设计

- 垂直伸缩
- 提升硬件能力
- 水平伸缩
- 增加服务器

加速静态内容访问速度的CDN

CDN全称是Content Delivery Network，中文释义是内容分发网络。CDN的作用是把用户需要的内容分发到离用户最近的地方进行响应，这样用户能够快速获取所需要的内容。CDN本质上就是一种网络缓存技术，能够把一些相对稳定的资源放到距离最终用户较近的地方，一方面可以节省整个广域网的带宽消耗，另外一方面也可以提升用户的访问速度、改善用户体验。

现实系统中我们一般会把静态的文件（图片、脚本、静态页面等）放到CDN中。



- 当用户访问网站页面上的内容URL，经过本地DNS系统解析，DNS系统最终会将域名的解析权交给CNAME指向的CDN专用DNS服务器；
- CDN的DNS服务器将CDN的全局负载均衡设备IP地址返回用户；
- 用户向CDN的全局负载均衡设备发起内容URL访问请求；

- CDN全局负载均衡设备根据用户IP地址，以及用户请求的内容URL， 选择一台用户所属区域的区域负载均衡设备，告诉用户向这台设备发起请求；
- 区域负载均衡设备会为用户选择一台合适的缓存服务器提供服务。
  选择的依据包括：根据用户IP地址，判断哪一台服务器距离用户最近。根据用户所请求的URL中携带的内容名称，判断哪一台服务器上有用户所需内容；查询各个服务器当前的负载情况，判断哪一台服务器上有服务能力。基于以上条件的综合分析之后，区域负载均衡设备会向全局负载均衡设备返回一台缓存服务器的 IP 地址。
- 局负载均衡设备把服务器的IP地址返回给用户。
  用户向缓存服务器发起请求，缓存服务器响应用户请求，将用户所需内容返回到用户终端。如果这台缓存服务器上并没有用户想要的内容，而区域均衡设备依然将它分配给了用户，那么这台服务器就要向它的上一级缓存服务器请求内容，直到追溯到包含该内容的源服务器并将内容拉到本地。

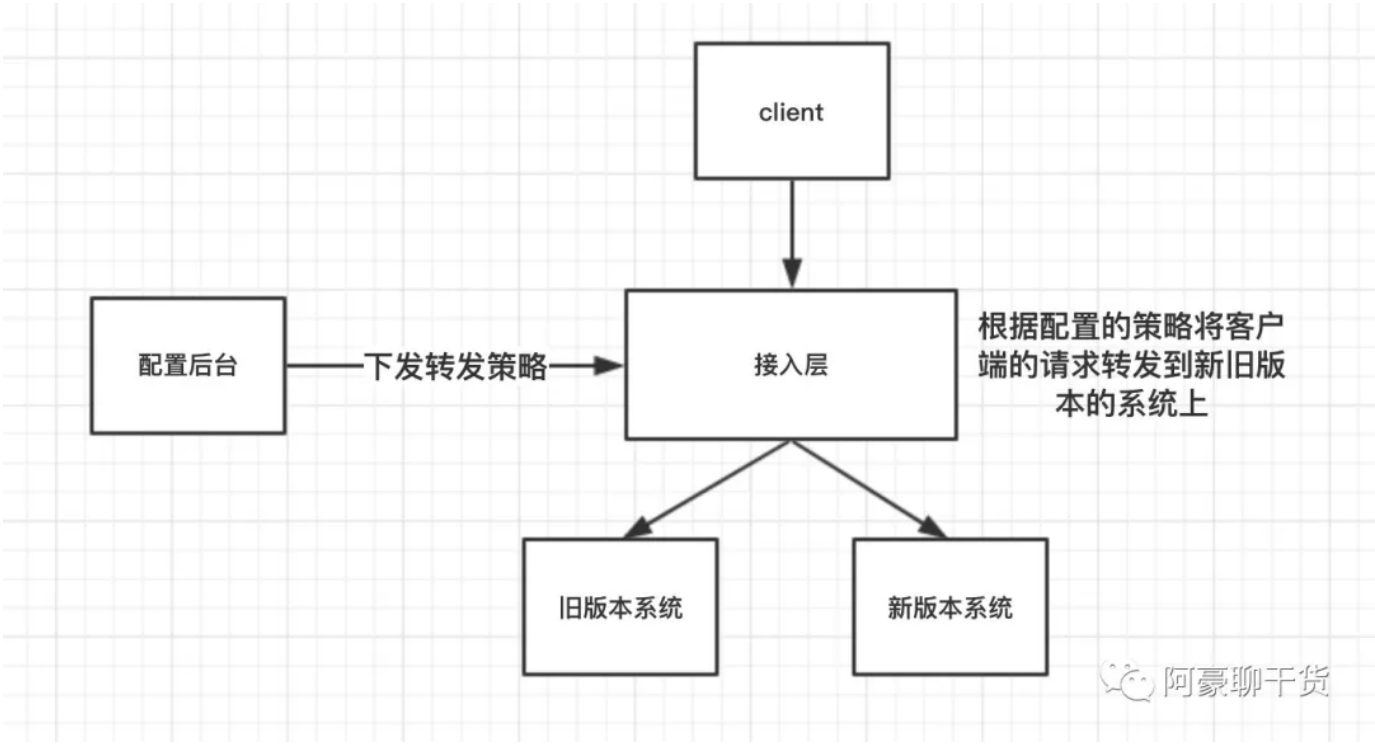
什么情况下用CDN？

最适合的是那些不会经常变化的内容，比如图片，js文件，CSS文件，图片文件包括程序模板中CSS文件中用到的背景图片，还有就是作为网站内容组成部分的那些图片等等。

灰度发布

我们的应用即使经过了测试部门的测试，也仍然很难全面覆盖用户的使用场景，为了保证万无一失，我们在进行发布的时候一般都会采用灰度发布，也就是会对新应用进行分批发布，逐步扩大新应用在整个及集群中的比例直到最后全部完成。灰度发布是说针对新应用在用户体验方面完全无感知。

灰度发布系统的作用在于，可以根据自己的配置，来将用户的流量导到新上线的系统上，来快速验证新的功能，而一旦出问题，也可以马上的回滚发布，简单的说，就是一套A/BTest系统。



七、总结

通过本文，我们就对主流的SOA架构、微服务架构、服务网格架构做了解析，然后知道了分布式架构中的几个基本理论，还分析了如何设计出高可用的分布式架构。不知大家是否有所收获呢？有意见或建议的朋友欢迎留言与我讨论。

作者：阿豪
来源：阿豪聊干货订阅号（ID：hafiz_talk）
DBAplus社群欢迎广大技术人员投稿，投稿邮箱：editor@dbaplus.cn

文章转载自公众号

阿豪聊干货