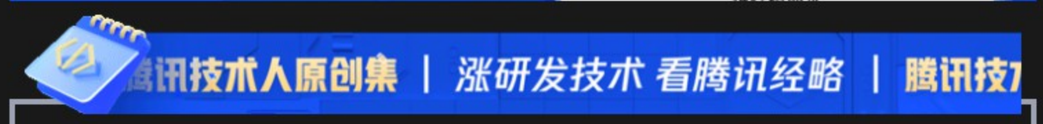
微服务与分布式系统设计看这篇就够了！

原创 黄楠驹 腾讯云开发者 2024年10月22日 08:45 北京



👉目录

后台分布式架构形形色色，特别是微服务和云原生的兴起，诞生了一批批经典的分布式架构，然而在公司内部，或者其他大型互联网企业，都是抛出自己的架构，从接入层，逻辑层，数据层都各有特点，但这些系统设计中到底是出于何种考量，有没有一些参考的脉络呢，本文将从云原生和微服务，有状态服务，无状态服务以及分布式系统等维度探讨这些脉络。

01 分布式系统概论

下面这个定义来自于经典的《Designing Data-Intensive Application》：

一个涉及通过网络进行通信的多台机器的系统被称为分布式系统。需要分布式系统，一般是希望能从分布式系统达到以下的好处：

- **容错/高可用性**：如果您的应用程序需要在一台机器（或多台机器、网络或整个数据中心）宕机时仍然继续工作，您可以使用多台机器来提供冗余。当一台机器失败时，另一台可以接管。
- **可扩展性**：如果您的数据量或计算需求超过单台机器的处理能力，您可以将负载分散到多台机器上。
- **低延迟**：如果您的用户遍布全球，您可能希望在全球各地设置服务器，以便每个用户都可以从地理位置靠近他们的数据中心获得服务。这避免了用户必须等待网络包绕地球半圈来响应他们的请求。
- **资源弹性**：如果您的应用程序在某些时候忙碌而在其他时候空闲，云部署可以根据需求扩展或缩减，因此您只需为您实际使用的资源付费。这在单台机器上更难实现，因为它需要预先配置好以应对最大负载，即使在很少使用时也是如此。
- **法律合规**：一些国家有数据居留法律，要求在其管辖区内的人的数据必须在该国地理范围内存储和处理 。这些规则的范围各不相同——例如，在某些情况下，它仅适用于医疗或财务数据，而其他情况则更广泛。因此，一个在几个这样的司法管辖区有用户的服务将不得

不将其数据分布在几个位置的服务器上。

分布式系统虽然能带来这些好处，但是分布式系统也存在很多挑战。通过网络传输的每个请求和 API 调用都需要处理可能发生的故障：网络可能中断，服务可能过载或崩溃，请求超时。

但 DDIA 定义主要是基于有数据有状态来讨论分布式，但是从现实的实践中，分布式系统存在有状态和无状态两种：

类型	相关描述
有状态	特点： • 有状态服务会在自身保存一些数据，因此先后的请求是有关联的。 • 处理一个请求可能需要依赖之前请求的结果或上下文信息，这些信息被保存在服务的状态中。 • 由于有状态服务需要维护状态的一致性，因此在扩展或部署时需要考虑状态迁移和同步的问题。 • 有状态服务通常用于需要维护用户会话、事务处理或需要保持数据一致性的场景。 相关理论： • CAP理论，CAP理论是指在分布式系统设计中，一致性（Consistency）、可用性（Availability）和分区容错性（Partition Tolerance）这三个特性无法同时满足。一致性指的是在分布式系统中的所有数据备份在同一时刻是否同样的值；可用性指的是每个请求不管成功或者失败都有响应；分区容错性指的是系统中任意信息的丢失或失败不会影响系统的继续运作。 • BASE理论：BASE理论是对CAP定理的一种实用化延伸，强调在分布式系统中适当放宽对强一致性的要求，以换取更高的可用性和系统性能。其核心思想是：我们无法做到强一致，但每个应用可以根据自身的业务特点，采用适当的方式来使系统达到最终一致性。
无状态	特点： • 无状态服务在处理请求时不依赖其他请求，每个请求都是独立的。 • 处理一个请求所需的全部信息要么包含在请求本身中，要么可以从外部资源（如数据库）中获取。 • 服务器本身不存储任何与请求相关的状态信息，因此不需要在请求之间保持状态的一致性。 • 由于无状态服务的独立性，它们通常更容易扩展和部署，因为不需要考虑状态迁移或同步的问题。 特别的，是各种并行计算相关框架，比如： • MapReduce算法 • OpenMP • MPI框架

为什么我们要重点说有状态的分布式架构？

目前大部分业务应用场景（特别是 to c 业务）需要存储用户的数据以及业务数据，本质还是数据密集型系统，也就是有状态的服务，那么**如何设计好一个有状态的分布式架构，是大部分业务都会面临的共同问题，也是本文的重点。**

这里给出设计有状态分布式架构，需要的一些考虑：

- 数据可靠：数据写入可靠，并且多个副本间最终一致性。
- 高可用：可以实现物理故障的容灾（粒度可能包括机器，机架，同城，跨城）。
- 更好的用户体验：尽量让单个请求耗时减少，从用户到业务机器，非必要不跨城，如果跨城，一次请求至多一次跨城。
- 高并发：支持高性能的读写操作，读写要求都超过单机性能。
- 降低运维成本：支持水平扩缩容，并且尽可能利用机器资源。

— 02 —

实现分布式系统的模型

下面的图都涉及到两个概念，这两个概念很多业务分层的概念，本文用这个分层来作为例子讨论：

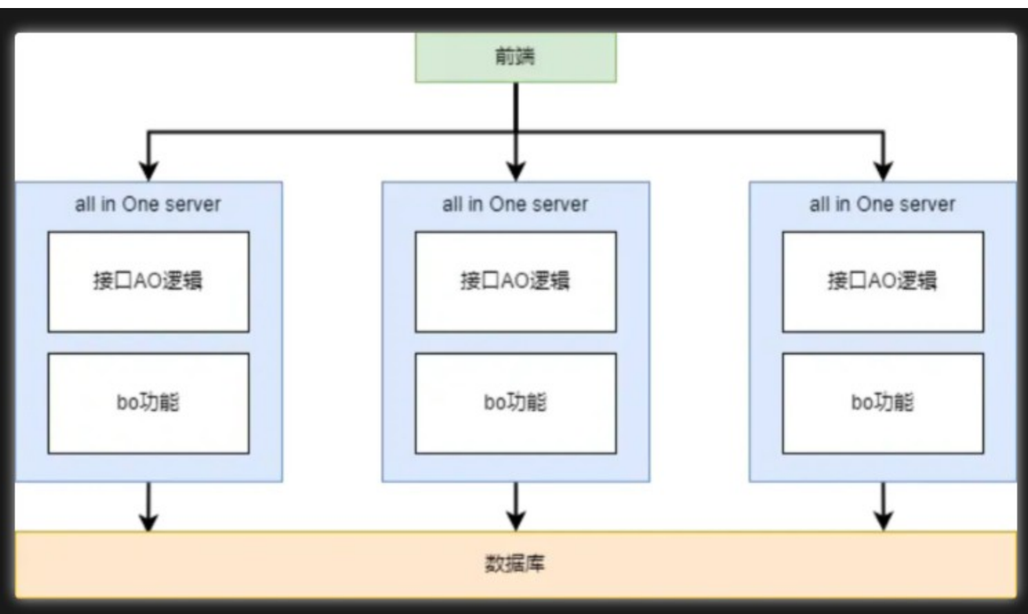
AO：封装了应用程序的业务逻辑和处理流程。AO 主要负责处理用户请求，调用相关的原子服务来完成特定的任务。它通常位于微服务的顶层，与其他对象进行交互，协调不同的功能模块。

BO：微服务中相关的原子服务，负责业务原子化的服务，通过被各种 AO 服务调用。功能可以是某个特定的业务原子功能，也可以是和数据打交道的原子服务。

实现有状态的分布式系统，通常有以下三种：

2.1 单体应用

单体架构是一种传统的架构方式，应用程序作为一个整体进行开发、测试和部署。



优点

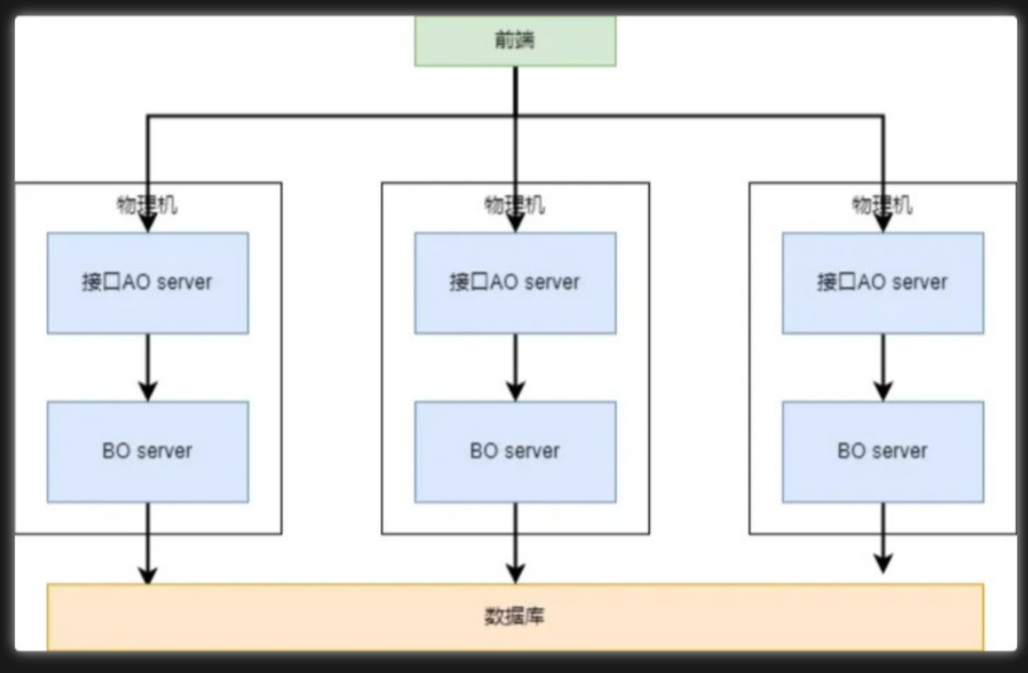
- **简单性**：相对于微服务架构，单体架构的开发、测试和部署更为简单。整个服务可以简单扩展。
- **性能较好**：由于所有功能都在同一个进程中运行，通常具有较好的性能和响应能力。
- **易于维护**：所有的代码和功能都在同一个代码库中，使得维护和修改相对容易。

面临的问题

- **系统复杂度高**：随着功能的增加，代码库变得庞大和复杂，导致开发人员难以理解整个系统，进而影响代码的质量和维护性。
- **开发速度慢**：由于需要编译、构建和测试整个项目，每次更改代码都会消耗大量时间，增加了开发成本。
- **难以扩展**：单体应用难以根据不同模块的需求进行针对性的扩展，往往需要整体扩展，导致资源利用效率低下。
- **难以维护**：模块之间的耦合度较高，修改一个模块的需求往往会带来连锁反应，影响其他模块的稳定性。
- **难以采用新技术**：项目是一个庞大的整体，使得应用新技术的成本很高，因为必须对整个项目进行重构，这通常是不可能的。
- **开发速度慢**：应用太大，每启动一次都需要很长时间，因此从编辑到构建、运行再到测试这个周期花费的时间越来越长。
- **部署困难**：代码部署的周期很长，而且容易出问题。程序更改部署到生产环境的时间变得更长，代码库复杂，以至于一个更改可能引起的影响是未知的。
- **系统故障隔离差**：应用程序缺乏故障隔离，因为所有模块都运行在同一个进程当中，任何部分的故障都可能影响整个系统的稳定性，导致宕机。

2.2 SOA 架构

SOA 架构更多是关注于改变IT服务在企业范围内的工作方式，SOA（面向服务的架构）定义了一种可通过服务接口复用软件组件并实现其互操作的方法。简单举个例子，使用 SOAP（简单对象访问协议）/HTTP 或 Restful HTTP (JSON/HTTP) 等标准网络协议来公开服务算是 SOA 的一种。



优点

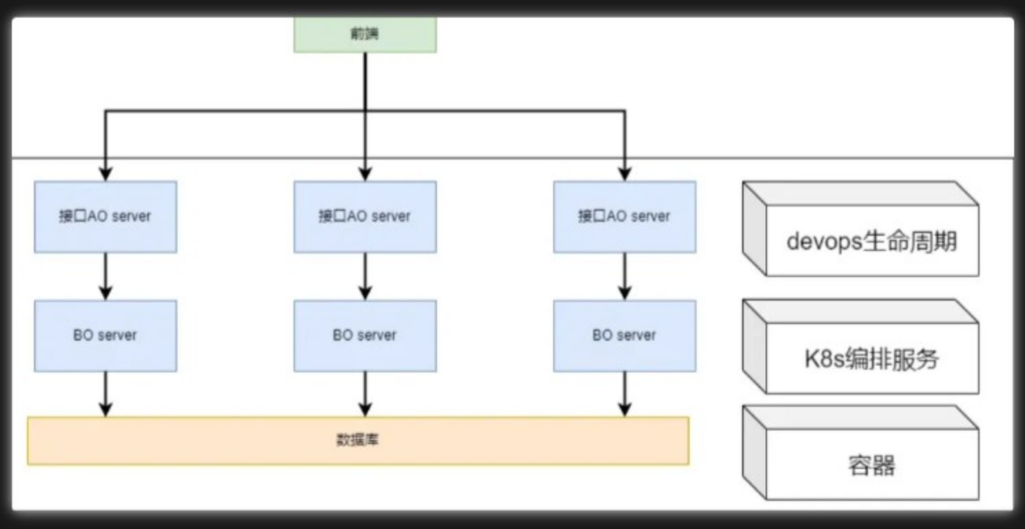
- **可扩展性和灵活性**：SOA 架构将系统拆分成独立的服务，可以按需组合和重组这些服务，从而实现系统的快速扩展和灵活部署。
- **提高系统的可重用性**：每个服务都是独立的功能单元，可以在不同的系统中复用，提高了系统的开发效率和维护成本。
- **降低系统的耦合性**：SOA 架构通过服务之间的松耦合关系，降低了服务之间的依赖性，有利于系统的模块化和维护。
- **提高系统的稳定性和可靠性**：SOA 架构采用了服务注册与发现机制、负载均衡、故障恢复等机制，提高了系统的稳定性和可靠性。

面临的问题

- **系统复杂度高**：SOA 架构中涉及多个服务之间的协作和通信，系统的复杂度较高，开发、测试和维护成本相对较高。
- **性能问题**：由于服务之间的通信需要通过网络进行，可能存在网络延迟和性能损失，对系统的性能造成影响。
- **安全性难以保障**：SOA 架构中涉及多个服务之间的通信，需要对数据传输进行加密和安全控制，保障系统的安全性比较困难。
- **部署和运维难度大**：SOA 架构中涉及多个服务的部署和管理，需要专门的运维团队进行管理，增加了系统的复杂性和运维成本。

2.3 微服务

微服务架构是一种云原生架构常用的实现方式，在这种方法中，单个应用程序由很多松散耦合并能够独立部署的更小组件或服务组成。一般认为是 SOA 架构的一种变体，一般也会使用 SOA 的原则来设计接口，但微服务更强调基于云原生，独立部署，和 devops，持续交付是一脉相承的。



优点

除了类似于 SOA 的优点，还有以下优点

- **独立性**：每个服务可以独立部署和更新，提高了系统的灵活性和可靠性。
- **可扩展性**：根据需求，可以独立扩展单个服务，而不是整个应用程序。
- **容错性**：单个服务的故障不会影响其他服务，提高了系统的稳定性。

对比 SOA，主要是在部署和运维难度上提升了，但是又引入了以下一些需要解决的问题：

1. **必须有接入层**：如上图，微服务化后，必然存在用户需要直接链接后端服务，那么这个时候就需要网关来解耦这块，也就是上面接入层讨论的好处。
2. **服务容错**：多个微服务部署在云上，不同母机，会带来通讯的复杂性，网络问题会成为常态，那么如何容灾，容错，降级，也是需要考虑的。
3. **服务发现**：当服务 A 发布或者扩缩容时，依赖服务 A 的服务 X 如何在保持运行的前提下自动感知到服务 A 的变化。这里需要引入第三方服务注册中心来实现服务的可发现性。比如北极星，stark，以及如何和容器，云原生结合。
4. **服务部署**：服务变成微服务之后，部署是分散，部署是独立的，就需要有一个可靠快速的部署，扩缩容方案，也包括 ci/cd，全链路、实时和多维度的可观测性等，如 tke，智妍等，k8s 就是解决这种问题的。
5. **数据存储隔离**：数据存储隔离(Data Storage Segregation, DSS) 原则，即数据是微服务的私有资产，必须通过当前微服务提供的 API 来访问数据，避免数据层产生耦合。对于有状态的微服务而言，通常使用计算与存储分类的方式，将数据下层到分布式存储方案中，从而一定程度上实现服务无状态化。
6. **服务间调用**：服务 A 采用什么方式才可以调用服务 X，由于服务自治的约束 ，服务之间的调用需要采用开发语言无关的远程调用协议。现在业界大部分的微服务架构通常采用基于 IDL （Interactive Data Language， 交互式数据语言）的二进制协议进行交互，如 pb。

通过上面讨论，**目前微服务来实现分布式系统是比较符合云原生的趋势，所以下面的讨论主要是基于微服务化的设计。**

下面针对微服务设计需要解决的问题，一起探讨下上面微服务需要解决的1-5个方面在系统该如何实现（第6点方案比较明确，这里不展开讲述）：

03

接入层解决了什么问题？

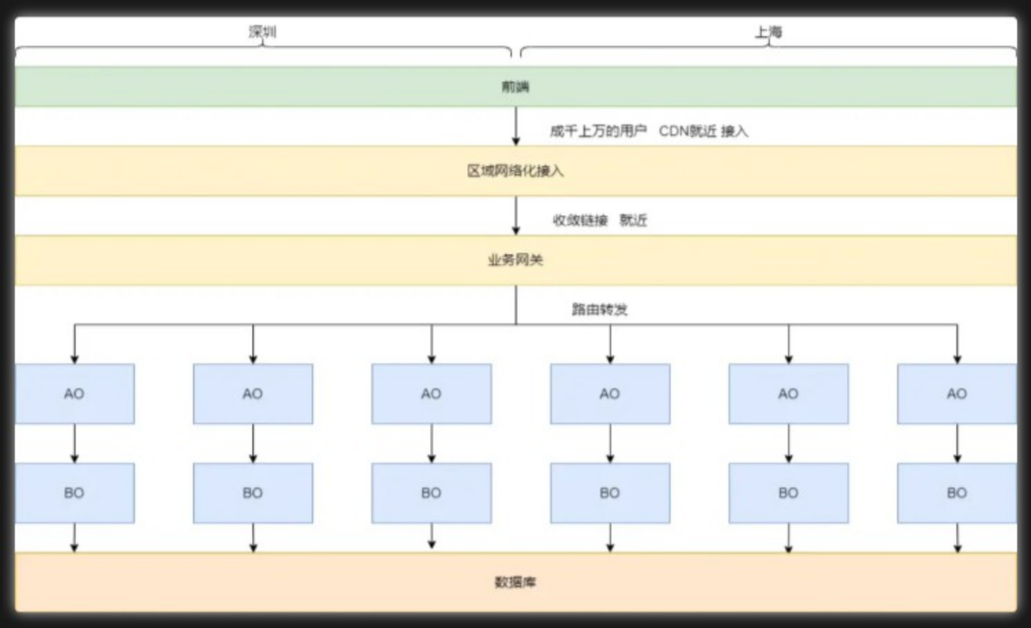
从上面单体应用架构，微服务架构，可以看出明显的差别，就是单体应用架构每个服务都可以提供完整的服务，那么用户和后台链接数可以通过服务的无限扩容得到满足，但是微服务不行，微服务的每个对外的服务接口都得和用户建立链接，就会存在两个最主要的问题：

1. **链接爆炸**，有多少对外的微服务就多几倍的链接
2. **服务和用户端耦合严重**，用户端得知道哪个请求发往哪个服务，缺乏统一路由。

那么这个时候引入一个中间层隔离用户和后端服务是非常必要，这个中间层负责对接用户的链接，然后把请求往业务服务上发。同时考虑服务的用户有地域性，可以把中间层分成两个部分：

1. **区域化网络接入层**：负责区域化网络接入，跟用户地域就近。
2. **业务网关**：负责服务透明代理，命令字的转发等。

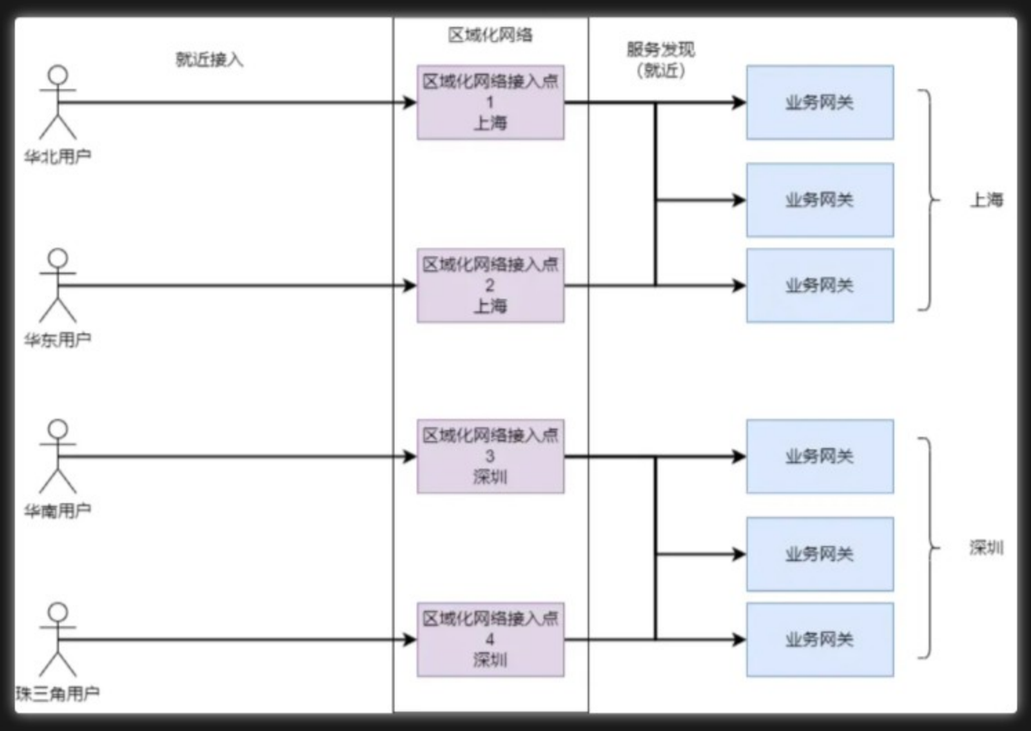
这时微服务架构就如下：



3.1 区域化网络接入层

这一层需要解耦的重点是把用户的实际地域 和 业务的逻辑RS 对应起来，同时避免每个业务都去建设类似的架构，同时解决用户体验，让用户尽量访问离用户最近的业务机器，减少耗时。

所以用户到区域化网络层应该是地域就近，区域化网络层到业务网关也是就近，深圳对应深圳，上海对应上海，并且考虑机房间就近。



3.2 业务网关

从上面分析可以看到，业务网关的作用最主要是透明服务代理（命令字转发），但业务网关还能提供如下作用：

- **透明服务代理**：提供统一服务入口，让微服务对前台透明，一般是通过命令字和后端 ao 接口对应起来。
- **控制访问**：收敛登录态校验，转发逻辑，染色，业务头部统一规整等功能，
- **用户流量接入和隔离，流量转发，保护后端**：如果是长连接服务，还可以收敛业务 ao 的链接，用户只需要和接入层链接，就可以使用所有服务，而不是每个用户都和 ao 链接，会造成 ao 服务的负载比较高。
- **负载均衡**：可以把用户的流量均衡（某些场景可能不是完全均衡，但也有分流的作用）请求 ao。
- **限流降级**：针对命令字会有限流，允许网关在极端情况下，进行随机请求丢弃，以避免整体服务的雪崩。
- **就近接入，尽早跨城**：尽可能先计算好用户的访问路径，比如读请求，可以直接就近访问下游 ao，bo，写请求需要知道用户数据的写点，可以尽早找到写点存在的集群，以满足至多一次跨城的需求。
    - 如果没有在一开始计算出来，在写点是跨城并且一次请求需要多次写的情况下，ao 需要调用多次 bo，bo 再去跨城的，就可能存在多次跨城。

业务网关的主要构成有：具有代理用户的接入服务 + 具有路由的转发服务。

路由实现又包含三个部分，路由存储、路由计算、路由容灾。

04

微服务的容错

4.1 条带化

为啥说微服务的服务容错能力，要到这个概念？

微服务在部署上，特别是在云上，一种服务进行多份同构部署，以提供容灾的可能性，如果多种微服务的部署机房是无序的，那么在故障出现时，就很容易出现服务的调用链路，部分正常，部分故障，就算故障服务被剔除系统外，链路也是来回穿梭在多个机房，耗时和故障影响都不可控。(考量一些服务半死不活，也会形成二次打击)。

所以一个比较简洁的方案是一个完整的服务集群部署在同一个物理单元，在物理单元的粒度上进行服务的容错，同时在同一个物理单元内，微服务内的时耗也是最低的。

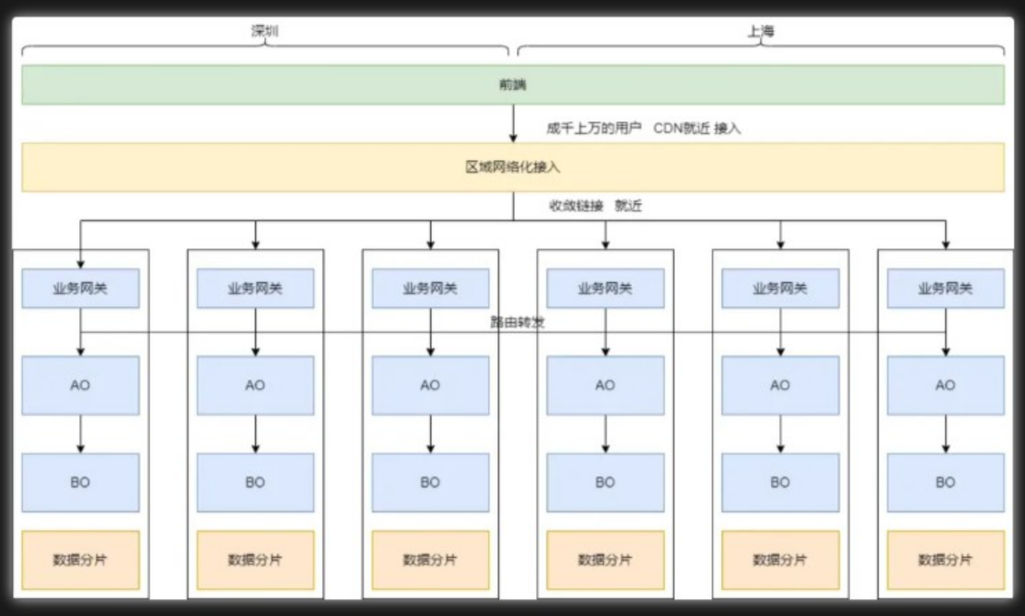
把一个完整服务集群部署在同一个物理单元（或者多个物理单元构成的逻辑单元），并且隔离流量的做法叫做条带化，其中每一个条带化的单元也叫做 set，也有人称之为单元化和逻辑单元，下文主要以条带化和 set 来表述。

使用条带化的好处：

- **多 AZ 容灾**：通过分布在不同可用区的服务器提高系统的容灾能力。提供 IDC（AZ 可用区）级别的物理容灾能力。
- **提升用户请求访问速度**：通过优化网络结构和部署策略，加快用户请求的处理速度。同 IDC（AZ 可用区）间微服务调用，耗时少。
- **多活架构**：通过在 IDC（AZ 可用区）之间分配流量负载，提高系统处理能力和资源利用率。
- **故障影响控制**：系统故障影响可以控制在故障的物理粒度范围内，确保服务的高可用性。
  - 一般还要求流量隔离，这样才能避免流量互相穿透在故障下带来的故障扩散效应。
  - 容灾是基于 IDC（AZ 可用区），而不是直接管理微服务，大大减少容灾维护的成本，同时可以提供容灾切换的成功率。

那么是不是自上而下都是条带化（包括逻辑层，数据层），就可以保证业务整体都有容灾能力呢？我们来对比下。

自上而下全部条带化



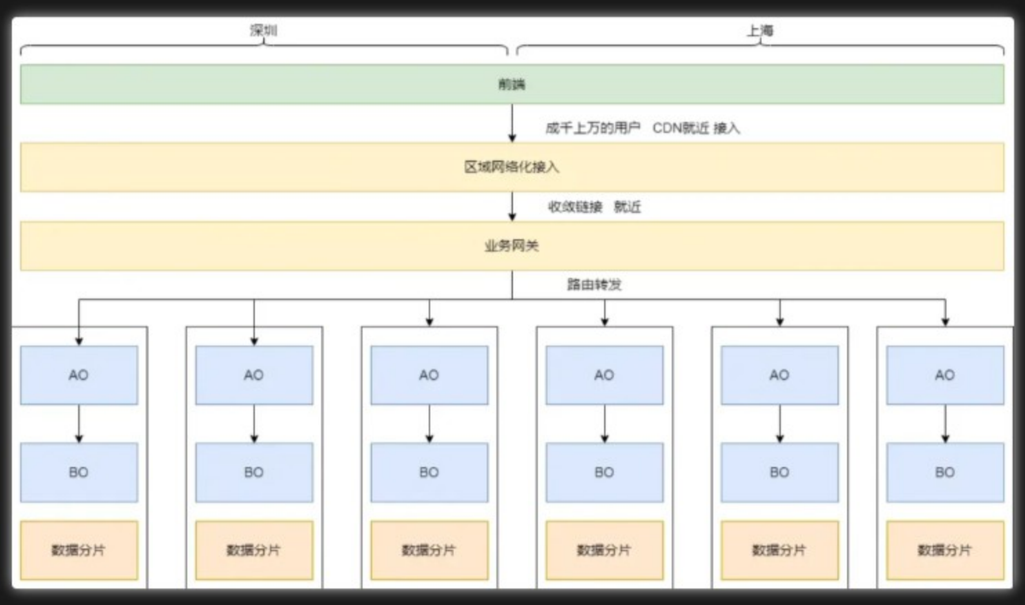
这个方案是把业务关心的内容都包括在条带化容灾里面，但从上面可以直接得看出因为用户存储数据不一定是在就近接入的业务网关的那个 set，所以业务网关实际也是有可能回路由到其他服务，这里其实就破坏了条带化。**这种条带化没有达到流量隔离的目的。比如某个 set 的 ao 耗时变慢了，所有 set 的业务网关会收到影响，这个 set 所在机器也有可能受影响，进而导致这个 set 的 ao 和 bo 也受影响。故障会扩散。**

除此之外，这种方案还有以下缺点：

- **网关难以复用多个业务**：业务网关的水平扩展与逻辑层耦合，业务网关只能 for 单个业务架构，达不到复用。
- **容灾切换复杂**：需要区域网络接入层和业务网关的路由协同处理容灾切换。如果是业务网关以下发生故障，发生切换，需要把业务网关的路由转发踢掉故障 set，区域网络接入层也要踢掉故障 set。
- **数据层耦合限制 ao 和 bo 扩容**：为了保证数据一致性，就需要牺牲整个 set 的可用性，这在一定程度会造成 ao 和 bo 的不可用。同时数据层的数据分片迁移难度也会限制扩容 set，从而影响 ao 和 bo 的扩容。

    - 更进一步说，无状态的微服务架构，自上而下都是条带化是可行，但是在有状态的微服务架构里面，无状态服务可以条带化，但是数据存储本身如果条带化，就会存在 CAP 理论中一致性和可用性的矛盾。
- **业务网关需要理解数据分片情况**：为了保证至多一次跨城，如果请求存在多次调用某个 bo 访问数据库，就得在业务网关直接知道接口需要的数据在哪个 set 的路由能力。

接入层以下条带化

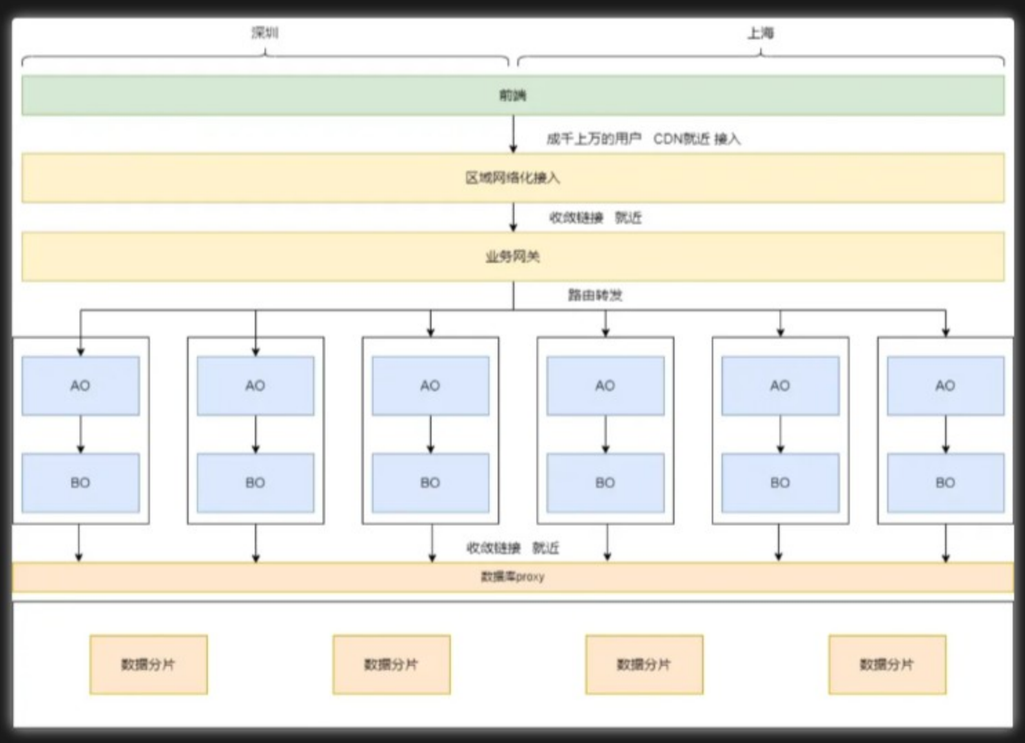


这个方案是把逻辑层和数据层放在 set 里面。

这种方案的缺点：

- **数据层耦合限制 ao 和 bo 扩容**：为了保证数据一致性，就需要牺牲整个 set 的可用性，这在一定程度会造成 ao 和 bo 的不可用。同时数据层的数据分片迁移难度也会限制扩容 set，从而影响 ao 和 bo 的扩容。
    - 更进一步说，无状态的微服务架构，自上而下都是条带化是可行，但是在有状态的微服务架构里面，无状态服务可以条带化，但是数据存储本身如果条带化，就会存在 CAP 理论中一致性和可用性的矛盾。
- **业务网关需要理解数据分片情况**：为了保证至多一次跨城，如果请求存在多次调用某个 bo 访问数据库，就得在业务网关直接知道接口需要的数据在哪个 set 的路由能力
- **业务网关只能跨城切换**：这个严格来说不算缺点，因为业务网关是一个无状态的服务，可以水平扩容，甚至可以和逻辑层解耦部署，只考虑就近，多地部署即可，如果遇到机房故障，该机房机器剔除之后，还能正常地服务，也就不需要跨城切换了。

仅逻辑层条带化



这种方案，便是逻辑层 看作是无状态的微服务集群，把有状态的界限尽量限定在数据层，避免耦合逻辑层比较多的无状态服务。在设计微服务时，应当遵守无状态服务设计原则，与底层基础设施解耦，从而实现在不同容器之间自由调度。对于有状态的微服务而言，通常使用计算与存储分类的方式，将数据下层到分布式存储方案中，从而一定程度上实现服务无状态化。

这样做的可以解决上面两种方案 **数据层耦合限制 ao 和 bo 扩容 的缺点，但是为了保证尽早跨城，尽量减少跨城，业务网关路由可以和数据库 proxy 的路由具有一定 联动关系。尽可能在业务网关路由时通过用户地域路由（在用户地域存储数据也是最优方案）的方式减少跨城。**

具体的联动方式说明：

- 用户数据有位置变化或者数据扩缩容的时候，如果某个接口时强依赖这个数据，就需要知道这个变化，针对这类型的路由，路由项得变更。
- 当数据层需要容灾的时候，也需要根据数据库模型，进行相应的容灾路由变更，如果数据是全局可以不变，如果只有一地部署，需要根据业务场景和数据模型是决定需不需要容灾进行主写点城市切换，也可以通过改变路由来实现。这部分在数据层设计详细讨论。

4.1.1 条带化粒度选型

既然微服务的容灾单元是 set，解决的主要是物理级别的故障，那么 set 条带化粒度的选型有哪些考量，下面给个表格参考：

条带化粒度	业务考量	必要的条件	备注
城市	• 除非因为投入产出成效低等不考量容灾，一般都至少要求城市级别	• 服务发现支持城市级别	一般能满足风火水电隔离，还也可能存在就近城市群有基础设施共用，所以一般选择距离较远的两地，比如深圳，上海
IDC（AZ可用区）	• 业务是否有多IDC的成本，是否有可以物理隔离 • 的IDC预算IDC故障时，进行跨城切换对业务耗时，可用性影响比较大	• 服务发现支持IDC级别 • 有物理隔离的IDC，风火水电隔离	**推荐**
机架（机柜）	• 业务是否有多机架的成本，是否有可以物理隔离 • 的机架预算机架故障时，进行IDC切换对业务耗时，可用性影响比较大	• 服务发现支持机架级别 • 有物理隔离的机架，最好是风火水电都是隔离的	一般一个IDC的机架很多设施都是共用的。如量化交易，进行主机托管和交易所就近的话，如果出现机架的故障，就切IDC，会让交易变慢，从而业务受损，所以能切换机架是更好的。
机器（OMP）	• 业务是否属于耗时敏感，有要求强一致性，跨机器访问也不太能忍 • 受业务是否自研了一些配套部署措施来匹配，比如极致条带化	• 服务发现支持严格机器级别（或者是只允许本机互调）	一般不建议，因为上云之后就更难保证一台机器有所有微服务功能，能完整构成一套服 务但是类似微信支付这种高并发又需要强一致的业务，可以考虑

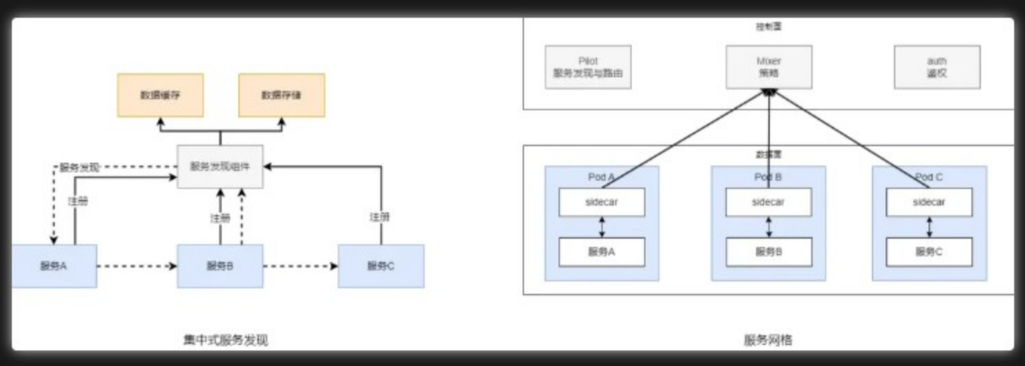
05

服务发现

微业务服务的互相调用，主要是业务网关发现 set，set 内微服务的互相发现，这两个都可以通过服务发现来做。

为了保证条带化（物理隔离，流量隔离），以及 set 内的完整性，服务发现时，应该尽量**在条带化粒度内服务可以互相发现，条带化粒度外服务不能互相发现（容灾降级应该由 set 间的容灾策略决定），我们选择的服务发现组件一定要支持的一个能力。**

在云原生的场景，一般存在两种服务发现实现方案，集中式服务发现，以及服务网格。



集中式服务发现与服务网格**的主要区别在于它们解决的问题范围和实现方式。**

集中式服务发现是一种服务注册和发现机制，通过一个中心化的服务注册表来管理所有服务的IP地址和相关信息。每当服务上线或下线时，它会向注册表注册或注销自己，其他服务在需要时查询注册表以获取其他服务的地址。这种方式简化了服务之间的通信，但需要维护一个中心化的服务注册表，可能成为单点故障，并且需要处理网络延迟和单点故障问题。

服务网格则是一种专用的基础设施层，用于管理分布式应用程序中各个微服务之间的通信。它通过部署在应用服务旁边的代理（称为 sidecar）来处理服务之间的通信，提供服务发现、负载均衡、流量路由、身份验证和可观测性等功能。服务网格将网络通信的复杂性抽象化，使开发人员能够专注于应用程序逻辑，而不必处理底层的网络代码。它通过分布式的方式提供服务，避免了单点故障，并且提供了更高的灵活性和可扩展性。

总结来说，集中式服务发现通过中心化的方式简化服务之间的通信，但可能面临单点故障和网络延迟的问题；而服务网格通过分布式代理网络提供更高级的通信管理功能，具有更高的灵活性和可扩展性。

虽然有可能因为集中式服务发现故障不可用，但可以通过本地 SDK 接入提供本地缓存和异步更新的功能，也能一定程度上缓解单点依赖。相对来说如果服务网格的控制面故障了，其实也在短时间内没办法更新容器的网络策略。

06

扩容

如上所说微服务的部署是一个难题，微服务化后，服务数量都会比原来单服务数量多了十几倍，那么自动化部署，自动化扩容就成为微服务化一个必要项。所幸的是伴随着云原生的发展，容器编排系统也在飞速发展，其中 kubernetes 是代表性的容器化部署编排系统。

那么在支持条带化上，kubernetes 应该需要支持**set 内机器横向扩容，set 间也能横向扩容。**

要达到这个目的，一般需要几方面的支持：

- **部署平台的支持**：基于 kubernetes 就可以进行容器的扩缩容调度，在实操上可以按照母机提前规划资源集群，每个 set 用隔离的资源集群，所以每个 set 内可以根据资源扩缩容，也能扩容 set 数，可扩容的 set 数也是部门支持的资源集群。
- **网关动态路由支持**：set 间的扩容，会需要网关支持，可以让网关的路由识别到有set进行上下线，调整动态路由策略。
- **容灾支持**：容灾识别处理器（可通过接口错误码来做容灾识别）也需要识别到 set 进行上下线，才能识别 set 是不是有故障。
- **权限支持**：原则上，容器是无状态的，可以简单进行扩缩容，原则上不要依赖比如 ip 授权，set 授权。就算需要授权，也应该是变成扩缩容的中的一个自动步骤。

— 07 —

数据存储

微服务还有一个需要解决的问题，就是数据服务隔离的问题，数据存储是系统有状态的来源，可以分为以下两种情况讨论：

1. 如果是数据存储是全局类的（写点只有一个），那么对于写，上层任何set都得接入，必然存在跨城，容灾时只能依赖数据存储自己的主从切换。
2. 如果数据存储是分片的，一般都希望上层 set 和底层写点尽量靠近，上文也提供，如果把数据分片和 set 绑定，是能最大程度靠近，但缺点是**数据分片本身会限制 set 的扩容，也需要上层需要知道底层数据分片情况。**

考虑数据分片的因素主要是包括以下考虑之一：

1. 不能允许跨城的写耗时。
2. 单机存储或者写 io 不能满足要求。
3. 希望通过数据隔离，减少故障容灾的影响，满足合规要求等。

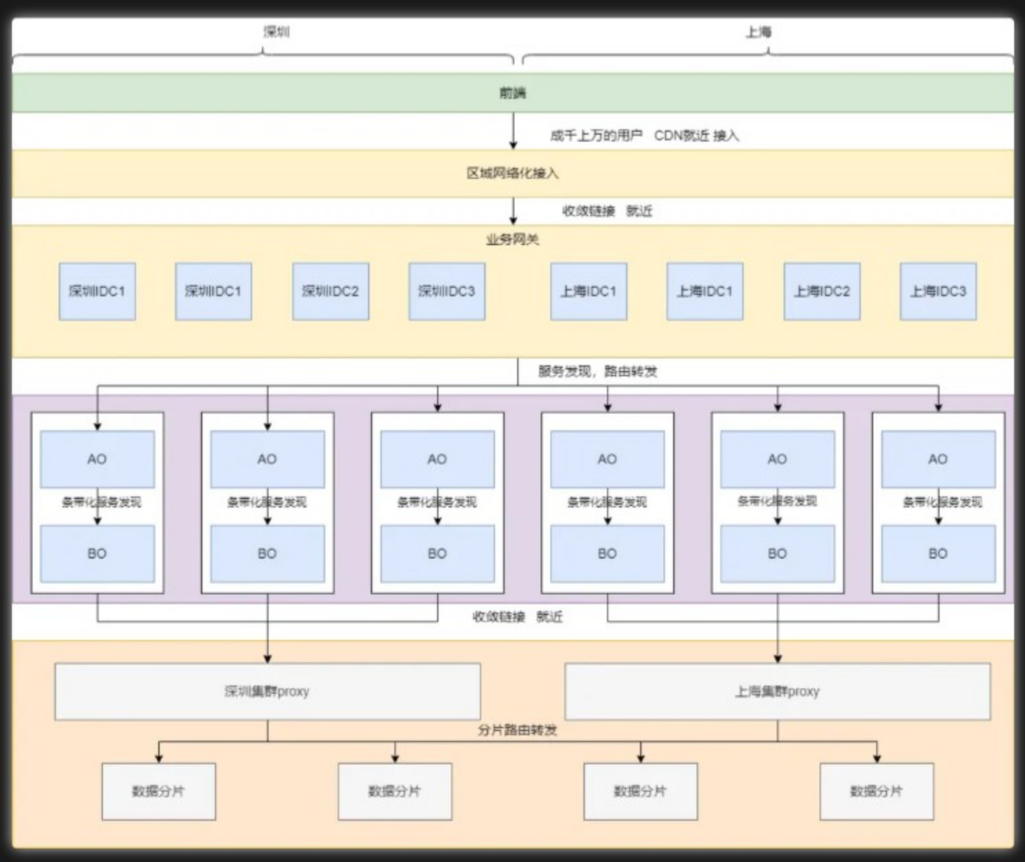
这里提供一种指导：

1. 数据分片和 set 不绑定，通过一层数据 proxy 解耦和 set 的关系。
2. 数据存储采用分布式存储系统，逻辑服务访问数据层通过就近访问（一般数据 proxy 这一层，分布式存储系统都会提供）。
3. 如果有严格地域就近要求，底层存储系统可以联动接入的路由来做尽可能的就近。把数据分片和部署关系通过路由数据来解耦，而非分散在逻辑服务中，让逻辑服务知道底层存储部署。

— 08 —

总结

经过上面讨论，一种有状态的分布式系统设计的整体架构图可以如下：



8.1 各层的容灾分析

以下都是基于机器足够的情况，性能机器预留就按照每个业务进行评估。

区域网络化接入层

1. **单机故障**：直接剔除出 CDN。
2. **机房故障**：直接把机房的机器都剔除出 CDN。
3. **城市故障**：把 CDN 暂时指向另一个城市

业务网关

1. **网关单机故障**：区域网络接入层剔除这些机器，可以通过监控错误自动剔除
2. **网关机房故障**：区域网络接入层剔除故障机房，可以通过监控错误自动剔除
3. **网关城市故障**：区域网络接入层把流量导到正常的城市，剔除掉故障城市的流量，这种情况下故障期间，会存在二分之一流量增加一次跨城耗时，理论上也可以实现自动剔除，但这个操作比较重，也需要进行慎重评估，建议是手动切。

逻辑层

- **网关单机故障**：业务网关路由剔除掉这些机器，可以通过服务发现组件剔除这些机器，可以通过监控错误自动剔除。
- **网关机房故障**：业务网关路由把故障 set 的流量均衡切到其他正常的 set，可以通过监控错误自动剔除。
- **网关城市故障**：业务网关路由把故障城市的 set 均衡切到正常城市的 set，理论上也可以实现自动剔除，但这个操作比较重，也需要进行慎重评估，建议是手动切。

数据层的容灾切换，一方面可以依赖逻辑层访问 db 故障后，返回给业务网关的错误码来实现从逻辑层开始切换，一方面可以依赖分布式存储系统自己的切换能力。

降级策略

业务网关也有一些其他应对故障的降级方案，如果在整体系统故障的情况下，可以这么处理降级：

- **降级时保证高优请求**：允许网关优先把资源提供给 P0 重要的命令字，其他接口做拒绝处理
- **防止雪崩**：允许网关在极端情况下，进行随机请求丢弃，以避免整体服务的雪崩。

-End-

原创作者 | 黄楠驹

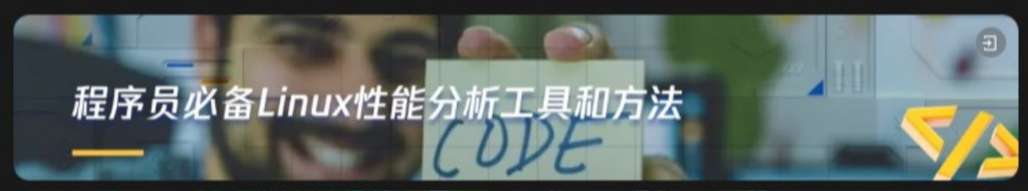
感谢你读到这里，不如关注一下？👇

微服务架构和分布式架构有哪些区别？欢迎评论分享。我们将选取**点赞本文并且留言评论**的一位读者，送出腾讯云开发者定制发财按键1个（见下图）。10月29日中午12点开奖。

欢迎加入腾讯云开发者社群，享前沿资讯、大咖干货，找兴趣搭子，交同城好友，更有鹅厂招聘机会、限量周边好礼等你来~

（长按图片立即扫码）

「 精品 · 知识推荐 」



赛博玄学，一键三连少一个Bug!

为好文章 点 收藏 点亮

腾讯技术人原创集 234

腾讯技术人原创集 · 目录 ≡

〈上一篇
一文搞懂第三方支付系统架构设计

下一篇〉
程序员的核心竞争力都藏在这个书单里了丨1024书单