阿里云

专有云企业版

蚂蚁单元化解决方案白皮书

文档版本：20200402

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云文档中所有内容，包括但不限于图片、架构设计、页面布局、文字描述，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

目录

1.单元化简介

1.1. 什么是单元化

异地多活是分布式系统的一种高可用部署架构，LDC 单元化架构是蚂蚁金服实现异地多活的解决方案和实现路径，经过了蚂蚁金服自身严苛的金融场景验证和多年的业务锤炼。

LDC，全称 Logical Data Center（逻辑数据中心），是对 IDC（Internet Data Center，互联网数据中心）的一种逻辑划分。

所谓 “单元”，是指一个能完成所有业务操作的自包含集合，在这个集合中包含了所有业务所需的所有服务，以及分配给这个单元的数据。单元化架构就是把单元作为部署的基本单位，在全站所有机房中部署数个单元，每个机房里的单元数目不定，任意一个单元都部署了系统所需的所有应用，数据则是全量数据按照某种维度划分后的一部分。

单元化架构希望在分布式架构应用拆分和数据拆分的基础上解决以下问题：

- 异地场景下的访问延时问题，实现异地多活。
- 单机房数据库连接限制问题，突破物理限制。
- 容量预估和扩容复杂问题，按单元预估和扩容。
- 发布变更灰度问题，支持灰度发布。

单元化架构的核心原则是单元化流量封闭，包含以下几点：

- 核心业务是可分片的。
- 核心业务的分片是均衡的，如支付宝以用户 ID 作为分片维度。
- 核心业务尽量自包含，调用尽量封闭。
- 整个系统要面向逻辑分区设计，而不是物理部署。

单点瓶颈

任何一个互联网系统，不论是支付宝、淘宝，还是 Google、Facebook，当发展到一定规模时，都会不可避免地触及到单点瓶颈。这里所说的“单点”，在系统的不同发展阶段表现不同。

在系统发展初期，服务器和应用单点最先成为瓶颈，解决的方法也很简单，加机器、拆应用；紧接着数据库单点，解决起来就开始不那么容易了，典型的做法是先垂直拆分，再水平拆分，在这个过程中需要解决多数据源、数据分片、透明访问等问题。

当应用越来越多、服务器越来越多、数据库越来越多，单个机房的容量开始捉襟见肘，装不下这么多服务器。而且业务量的增长也让系统单机房运行的风险激增，一旦发生机房断电或是其他灾害导致机房故障，就会让整个系统完全瘫痪。机房不能放在一个篮子里，必须让系统在两个或更多个 IDC 里跑起来。

多机房部署通常有以下两种模式：

- 垂直模式：把全站应用、数据库等划分为几个部分，分别放在不同的机房中，完成一个业务可能需要不同机房中的不同应用提供服务。这就相当于在逻辑上把一个物理机房放大了，机房容量不足的问题得以解决。

- 水平模式：每个机房中部署的应用都是相同的，每个机房都有完成全站所有业务的能力，在运行时每个机房都只承担整站的一部分业务流量。

在具体实现上，垂直模式更容易一些，能够突破机房容量瓶颈，但是不具备容灾能力，而容灾，是一个发展到一定规模的互联网系统的重要诉求，更是金融场景和金融业务必不可少的基础能力，因此在具体实践中，大多数大型系统都采用多机房水平扩展的模式。

上面所说的容灾问题，对于一个有着亿级用户的系统，或一个数据系统来说尤为重要，仅仅是机房级容灾还不够，必须要考虑部署地容灾，也就是说不能把所有机房部署在地理上临近的地区，以防发生地震、海啸、核爆等剧烈灾害而导致系统被毁灭性破坏。这类系统的典型代表是银行、第三方支付等金融类系统，比如银行就对机房部署有着经典的“两地三中心”要求，于是单部署地开始成为制约业务发展的瓶颈。

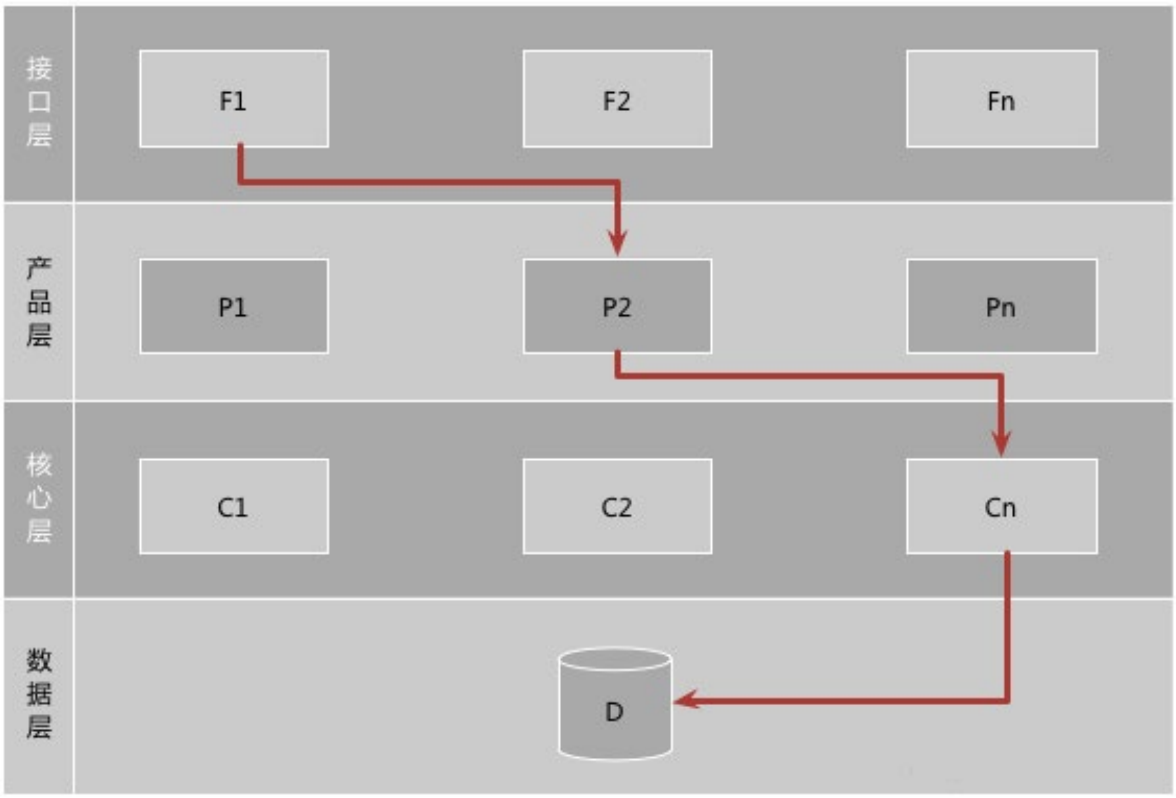
能把一个系统的一部分部署到距离较远的另外一个地区(城市)，是一个大型互联网系统走向成熟的标志。多地部署本质上和多机房部署是一样的，面临的问题也几乎一样，除了距离问题。表象上是距离，下面隐藏着的是延时，更远的距离意味着更长的延时，个位数毫秒的延时不会给系统带来什么麻烦，但一旦这个数值变成几十毫秒，量变就引发了质变，很多业务开始不能忍受和忽略延时带来的影响。

单元化

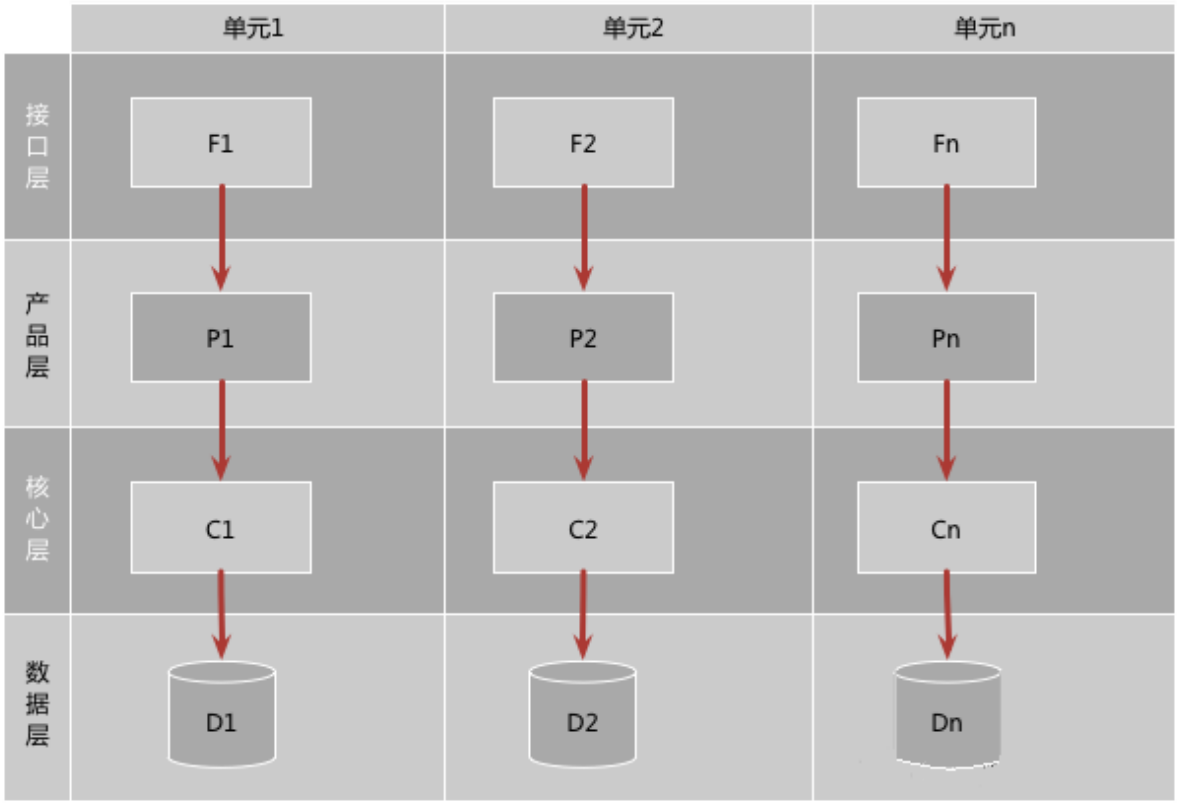
多地多机房部署，是互联网系统的必然发展方向，一个系统要走到这一步，也就必然要解决上面提到的问题：流量调配、数据拆分、延时等。业界有很多技术方案可以用来解决这些问题，而承载这些方案的，是一个部署架构。尽管可采用的部署架构不止一个，但不论是纯理论研究，还是一些先行系统的架构实践，都把“单元化部署”推崇为最佳方案。

所谓单元（对应单元化应用服务产品层的部署单元），是指一个能完成所有业务操作的自包含集合，在这个集合中包含了所有业务所需的所有服务，以及分配给这个单元的数据。单元化架构就是把单元作为部署的基本单位，在全站所有机房中部署数个单元，每个机房里的单元数目不定，任意一个单元都部署了系统所需的所有应用，数据则是全量数据按照某种维度划分后的一部分。

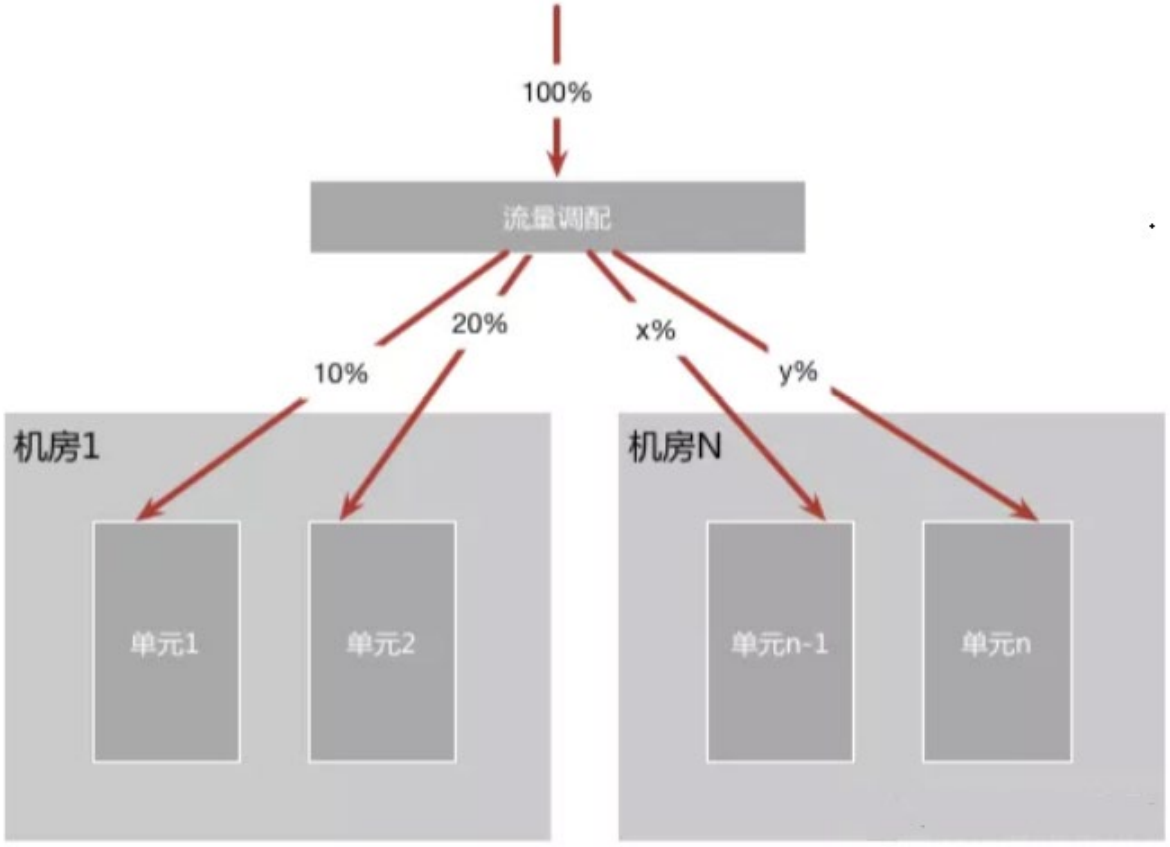
传统意义上的 SOA 化（服务化）架构，服务是分层的，每层的节点数量不尽相同，上层调用下层时，随机选择节点。



单元化架构下，服务仍然是分层的，不同的是每一层中的任意一个节点都属于且仅属于某一个单元，上层调用下层时，仅会选择本单元内的节点。



一个单元，是一个五脏俱全的缩小版整站，它是全能的，因为部署了所有应用；但它不是全量的，因为只能操作一部分数据。能够单元化的系统，很容易在多机房中部署，因为可以轻易地把几个单元部署在一个机房，而把另外几个部署在其他机房。通过在业务入口处设置一个流量调配器，可以调整业务流量在单元之间的比例。



从这个对单元的定义和特性描述中，可以推导出单元化架构要求系统必须具备的一项能力：数据分区，实际上正是数据分区决定了各个单元可承担的业务流量比例。数据分区（shard），即是把全局数据按照某一个维度水平划分开来，每个分区的数据内容互不重叠，这也就是数据库水平拆分所做的事情。 仅把数据分区了还不够，单元化的另外一个必要条件是，全站所有业务数据分区所用的拆分维度和拆分规则都必须一样。若是以用户分区数据，那交易、收单、微贷、支付、账务等全链路业务都应该基于用户维度拆分数据，并且采用一样的规则拆分出同样的分区数。比如，以用户 id 末 2 位作为标识，将每个业务的全量数据都划分为 100 个分区（00-99）。

有了以上两个基础，单元化才可能成为现实。把一个或几个数据分区，部署在某个单元里，这些数据分区占总量数据的比例，就是这个单元能够承担的业务流量比例。 执行

数据分区时一个很重要的问题是分区维度的选择，一个好的维度，应该：

- 粒度合适：粒度过大，会丧失流量调配的灵活性和精细度；粒度过小，会给数据的支撑资源、访问逻辑带来负担。
- 足够平均：按这个分区维度划分后，每个部署单元的数据量应该是几乎一致的。

通常对于以用户为服务主体的系统（很多 To C 的系统，比如支付宝），最佳实践是按照用户维度对数据分区。

1.2. 单元化基础术语

术语	含义
单元	应用层按照数据层相同的拆片维度，把整个请求链路收敛在一组服务器中，从应用层到数据层就可以组成一个封闭的单元。 数据库只需要承载本单元的应用节点的请求，大大节省了连接数。“单元”可以作为一个相对独立整体来挪动，甚至可以把部分单元部署到异地去。
逻辑单元	逻辑单元是单元化架构的基础，一个逻辑单元被称为一个 Zone。根据业务特点不同，您可以将系统部署在不同类型的逻辑单元中。有 3 种不同类型：RZone GZone、CZone。单元的特点如下： • 同一个应用在每个单元中拥有独立使用的资源。 • 同一个应用的业务在不同单元中按水平方向拆分。 • 不同单元处理的业务分片不重叠。

部署单元	所谓部署单元（Cell），是指一个能完成所有业务操作的自包含集合，在这个集合中包含了所有业务所需的所有服务，以及分配给这个单元的数据。 单元化架构就是把单元作为部署的基本单位，在全站所有机房中部署数个单元，每个机房里的单元数目不定，任意一个单元都部署了系统所需的所有应用，数据则是全量数据按照某种维度划分后的一部分。
租户	租户通常是一个公司、组织或项目组，基于业务对系统实例和程序组件有定制化需求，且需要与其他租户之间的数据相隔离。不同租户也会建立不同的账号系统及其对应的权限。
工作空间	工作空间的本质，是指 “网络互通、安全策略一致、访问延时极小” 的一组资源。您可以通过工作空间方便地将资源进行分组管理，例如：根据不同的研发交付需求，将工作空间划分为开发工作空间、测试工作空间、生产工作空间等。同时，不同工作空间中的资源互相隔离，可以为每个工作空间分配单独的操作员权限进行管理。
工作空间组	工作空间组对多个工作空间进行管理。 • 同城双活架构下，推荐工作空间和机房之间是 1:1 的关系，每个机房部署独立的 k8s 集群，从而实现工作空间组跨多个 k8s 集群，应用可以跨多个 k8s 集群进行部署。 • 异地多活架构下，工作空间组可以跨地域，从而实现应用的跨地域部署。
GZone	无法拆分的业务，如配置型的业务。数据库可以和 RZone 共享，多租户隔离，全局只有一组，可以配置流量权重。会被 RZone 依赖 。

RZone	核心业务和数据单元化拆分，拆分后分片均衡，单元内尽量自包含（调用封闭），拥有自己的数据，能完成所有业务。 一个可用区可以有多个 RZone 。
CZone	为了解决异地延迟问题而特别设计，适合读多写少且不可拆分的业务。 一般每个城市一套应用和数据，是 GZone 的快照，被 RZone 高频访问。
路由规则	单元化架构下的路由规则是一个 json 字符串，包含了路由信息、灾备信息、机房部署信息、灰度信息等，主要用于 http 请求转发、rpc 路由计算、msg 目标 Zone 计算、zdal 数据源连接、zcache 集群选择等。
路由权重	指同一个逻辑单元下，不同部署单元所承载的流量占比。
UID 分片	也叫 sharding key，指应用层和数据层的拆分维度，默认划分成 100 份（00-99），在 LHC 里可以指定 UID 分片与部署单元的映射关系。在实际使用过程中 UID 可以对应着多种类型的数据，如用户 ID、交易流水，只要能映射到两位分片位即可。

1.3. 单元化的适用场景

单元化架构是实现异地多活的解决方案，同时也提供了可演进的架构升级路径，可根据用户的业务现状和物理机房规划，提供可持续的架构升级方案。

- 同城双活

  在同一地域，建立 2 个机房，实现同城双活单元化架构。

- 两地三中心

  在同城双活单元化的基础上，增加一个异地机房，做应用和数据备份。

- 异地多活

  异地多活，异地机房也承担日常流量，可以做到任意数量地域的多活。

- 异构基础设施下的混合云

  通过 Kubernetes 屏蔽掉底层 IaaS 层的差异，可充分利用公有云上的资源，将业务同时在专有云和公有云上进行部署，并进行统一运维管控，实现弹性扩容，使业务能够按需进行机房级的无限水平扩展。

1.4. 选择单元化的决策依据

1.4.1. 单元化的成本

单元化架构的成本包含部署成本和业务改造成本两部分。

部署成本

- 需要额外基础设施，如统一接入网关、LHC 单元化应用服务。
- 每个单元需要部署一套业务应用

改造成本

- **业务梳理**

  梳理业务类型和业务链路，明确服务拆分和数据拆分原则。

- **应用拆分**

- 拆分和改造微服务。因为框架本身不理解业务逻辑，要确定应该调用哪个单元的服务，只能从业务参数中获取信息。所以需要有一种方案，能让每一个业务请求中携带某些标识，通过对这些标识的提取、识别、计算，可以得到这笔业务所属的单元。比如在服务接口中包含路由参数，消息中包含分片字段，批量任务感知路由规则等，需要进行相应的应用服务改造。

- 需不断地通过架构调优，将服务依赖链路单元内收敛，尽量减少跨单元的服务调用。

- **数据拆分**

  确定数据分片规则，如按业务领域垂直拆分，按业务字段水平拆分（分库分表）等。

1.4.2. 单元化的收益

单元化架构的收益包含：高可用容灾、弹性扩容、灰度变更管控三部分。

高可用容灾

金融行业，尤其是银行，出于监管需求或者业务连续性需求，对容灾能力有较高的要求。单元化架构能够在以下方面提升容灾能力，实现高可用。

- **支持同城、异地容灾，RPO=0，RTO≈0**

  - 数据进行水平拆分，通过采取自选举数据库，保证数据分片有同城和异地副本，灾难时支持自动切主，RPO 可以实现等于 0。

  - 单元应用流量与数据分片一致，在故障发生时，支持无脑切换，快速容灾，RTO 可以很短。

- **单元化多活，缩小故障影响范围**

  - 业务按单元分散在多个数据中心、多个地域，故障隔离域粒度非常小。

- **应对故障手段多，控制灵活，故障切换速度快**

- 通过自动化容灾平台，对接多产品的容灾切换和恢复操作，将容灾过程自动化。
- 提供多场景下的故障模拟和演练方案，将容灾演练日常化。
- 故障发生时，根据容灾预案快速进行容灾切换，恢复业务，缩短业务影响时间。

弹性扩容

随着金融行业思路的转变，将金融业务内嵌到个人生活服务中，必然会产生诸多高频场景。采用分布式架构，可以让业务在出现热点后，进行拆分、扩容，以应对流量激增。单元化架构能够在以下方面，更好的提升弹性能力，实现无限弹性扩容。

- **提升容量预估能力，支持全链路压测**

  按单元进行全链路压测，更方便的进行容量预估。

- **提升扩容效率，按单元灵活部署**

  按单元进行应用和数据的弹入弹出，灵活调配流量。

- **提升扩展性，异地多活，理论上无限扩展**
  - 数据库只需承载本单元的应用节点的请求，大大减少了数据库连接数，突破了单数据库连接数瓶颈，不受单库、容量限制。
  - 异地多活，突破了城市物理限制，理论上只要机房够多，容量可以无限扩展。

灰度变更管控

在单元化架构下，通过灰度发布可以更好的适应金融技术风险保障需求，实现大规模金融场景下的运维落地。

- **支持灰度发布，灵活调拨灰度流量**

  在新版本发布时，可以通过流量调拨，在小规模生产流量（UID，单元）范围内充分

验证新版本后，再逐步扩大发布范围。发布过程中如果验证有问题，可以及时回切，减少发布变更影响面，保证平滑上线。

- **支持蓝绿发布，新老调用单元隔离**

  通过单元流量封闭逻辑隔离能力，可以在发布过程中，隔离新老单元间调用，避免新老应用交叉访问兼容性问题，提升发布效率。

1.4.3. 决策关键点

	决策关键点	单元化收益
容灾	✓ 是否对系统容灾能力要求很高 ✓ 是否对业务连续性要求很高	• 业务分散在多个数据中心、多个地域，故障隔离域粒度非常小。 • 应对故障手段多，控制灵活，故障切换速度快。 • 抵御机房级和城市级灾难，RPO=0，RTO≈0。
弹性	✓ 是否期望系统支持的业务容量能理论上无限增长	• 突破单数据库连接数瓶颈和容量瓶颈，不受单库连接、容量上限限制。 • 突破单机房容量瓶颈和城市容量瓶颈，异地多活，可以水平扩展。 • 缩小微服务和数据库粒度，灵活部署，灵活扩缩容。

灰度	✓ 是否期望灰度发布能力	• 支持灰度发布，灵活调拨流量。
架构	✓ 是否期望未来能支持异地多活架构	• 单元化流量收敛，是实现异地多活的必要条件。

2. 单元化架构

2.1. CRG 架构

从单元化的角度来说，一个系统当中实际上存在 3 类数据：

- **可分区数据**

可以按照选择好的维度进行分区的数据，真正能被单元化的数据。这类数据通常在系统业务链路中处于核心位置，单元化建设最重要的目标实际上就是把这些数据处理好。比如订单数据、支付流水数据、账户数据等，都属于这一类型。

这类数据在系统中的占比越高，整体单元化的程度就越高，如果系统中全部都是这样的数据，那就能打造一个完美的单元化架构。不过现实中这种情况存在的可能性几乎为零，因为下面提到的两类数据，或多或少都会存在于系统当中。

- **全局数据（不被关键链路业务频繁访问）**

不能被分区的数据，全局只能有一份。比较典型的是一些配置类数据，它们可能会被关键链路业务访问，但并不频繁，因此即使访问速度不够快，也不会对业务性能造成太大的影响。

因为不能分区，这类数据不能被部署在经典的单元中，必须创造一种非典型单元用以承载它们。

- **全局数据（需要被关键链路业务频繁访问）**

与上一类数据相似，但两者有一个显著的区别，即是否会被关键链路业务频繁访问。如果系统不追求异地部署，那么这个区别不会产生什么影响；但如果希望通过单元化获得多地多活的能力，这仅有的一点儿不同，会让对这两类数据的处理方式截然不同，后者所要消耗的成本和带来的复杂度都大幅增加。

究其原因是异地部署所产生的网络时延问题。根据实际测试，在网络施工精细的前提下，相距约 2000 公里的 2 个机房，单向通信延时大约 20ms 左右，据此推算在国内任意两地部署的机房之间延时在 30ms 左右。假如一笔业务需要 1 次异地机房的同步调用，就

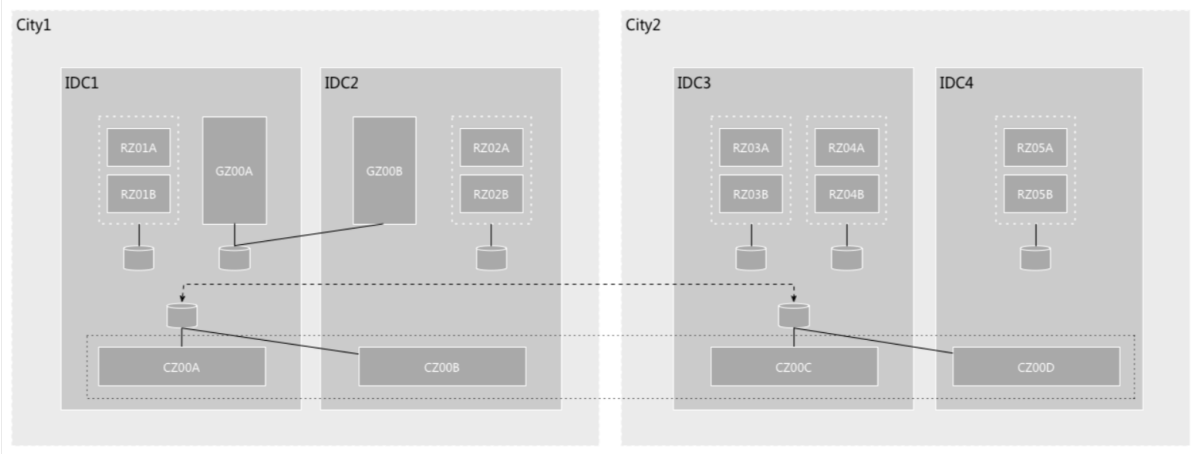
需要至少 60ms 的延时（请求去+响应回）。如果某个不能单元化的数据需要被关键业务频繁访问，而业务的大部分服务都部署在异地单元中，网络耗时 60ms 的调用在一笔业务中可能有个几十次，这就是说有可能用户点击一个按钮后，要等待数秒甚至数十秒，系统的服务性能被大幅拉低。

这类数据的典型代表是会员数据，对于 To C 的系统来说，几乎所有的业务都需要使用到会员信息，而会员数据却又是公共的。因为业务必然是双边的，会员数据是不能以用户维度分区的。

Zone（单元）

蚂蚁单元化架构中，把单元称为 “Zone” ，并且为上面所说的 3 类数据分别设计了 3 种不同类型的 Zone。

- RZone（Region Zone）：最符合理论上单元定义的 Zone，每个 RZone 都是自包含的，拥有自己的数据，能完成所有业务。
- GZone（Global Zone）：部署了不可拆分的数据和服务，这些数据或服务可能会被 RZone 依赖。GZone 在全局只有一组，数据仅有一份。
- CZone（City Zone）：同样部署了不可拆分的数据和服务，也会被 RZone 依赖。跟 GZone 不同的是，CZone 中的数据或服务会被 RZone 频繁访问，每一笔业务至少会访问一次；而 GZone 被 RZone 访问的频率则低的多。



RZone 是成组部署的，组内 A/B 集群互为备份，可随时调整 A/B 之间的流量比例。可以把一组 RZone 部署到任意机房中，包括异地机房，数据随着 Zone 一起走。

GZone 也是成组部署的，A/B 互备，同样可以调整流量。GZone 只有一组，必须部署在同一个城市中。

CZone 是一种很特殊的 Zone，它是为了解决最让人头疼的异地延时问题而诞生的，是蚂蚁单元化架构的一个创新。

CZone

CZone 解决这个问题的核心思想是把数据搬到本地，并基于一个假设：大部分数据被创建（写入）和被使用（读取）之间是有时间差的。

- 把数据搬到本地：在某个机房创建或更新的公共数据，以增量的方式同步给异地所有机房，并且同步是双向的，也就是说在大多数时间，所有机房里的公共数据库，内容都是一样的。这就使得部署在任何城市的 RZone，都可以在本地访问公共数据，消除了跨地访问的影响。整个过程中唯一受到异地延时影响的，就只有数据同步，而这影响，也会被下面所说的时间差抹掉。

- 时间差假设：举例说明，两个用户分属两个不同的 RZone，分别部署在两地，用户 A 要给用户 B 做一笔转账，系统处理时必须同时拿到 A 和 B 的会员信息；而 B 是一个刚刚新建的用户，它创建后，其会员信息会进入它所在机房的公共数据库，然后再同步给 A 所在的机房。如果 A 发起转账时，B 的信息还没有同步给 A 的机房，这笔业务就会失败。时间差假设就是，这种情况不会发生，也就是说 B 的会员信息创建后，过了足够长的时间后，A 才会发起对 B 的转账。符合这个假设的公共数据，就可以部署在 CZone，解决异地高频访问的问题。

2.2. 单元化流量路由

2.2.1. 流量路由层次

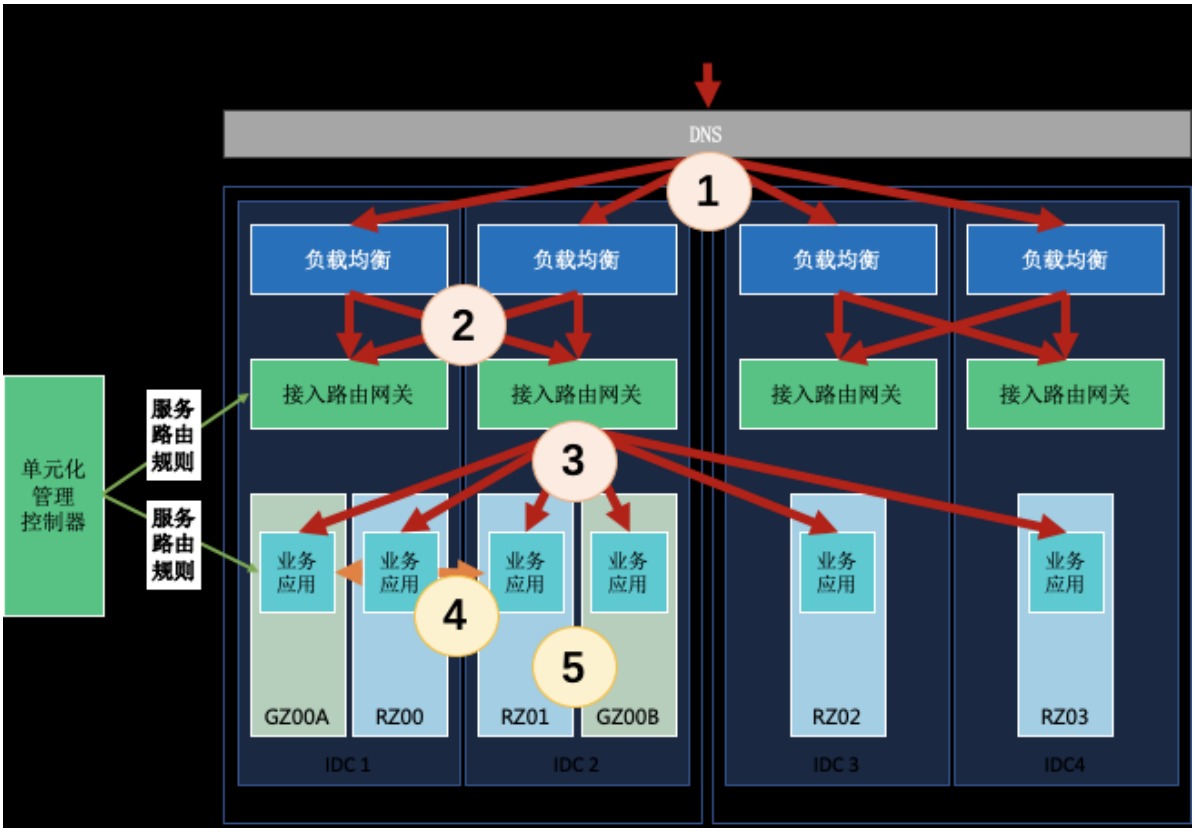
流量路由层次包含入口流量路由和跨 Zone 服务调用路由两种。

入口流量路由

- 域名->机房入口：DNS 解析，可通过配置解析规则调整流量。
- 入口负载->接入路由：按权重比例配置路由，一般配置为随机。
- 接入路由->应用：根据单元化服务路由规则计算路由目标。

跨 Zone 服务调用路由

- 同城跨 Zone 调用：由 RPC 服务框架根据调用参数自动调用。
- 跨城跨 Zone 调用：由 RPC 服务框架根据调用参数自动调用。



2.2.2. 入口流量路由

入口流量路由要解决的问题是，如何识别一次业务请求属于哪一个单元，从而把请求路由到正确的单元中去。

解决这个问题的关键是：需要一种方案，能让每一个业务请求中携带有某些标识，通过对这些标识的提取、识别、计算，得到这笔业务所属的单元。

蚂蚁单元化架构中，针对以 HTTP 为主要请求协议的业务，解决入口流量路由问题的方案是，与业务应用配合，在业务请求 Cookie 中设置标识。具体描述如下（以下方案假设单元划分维度为 '用户'）：

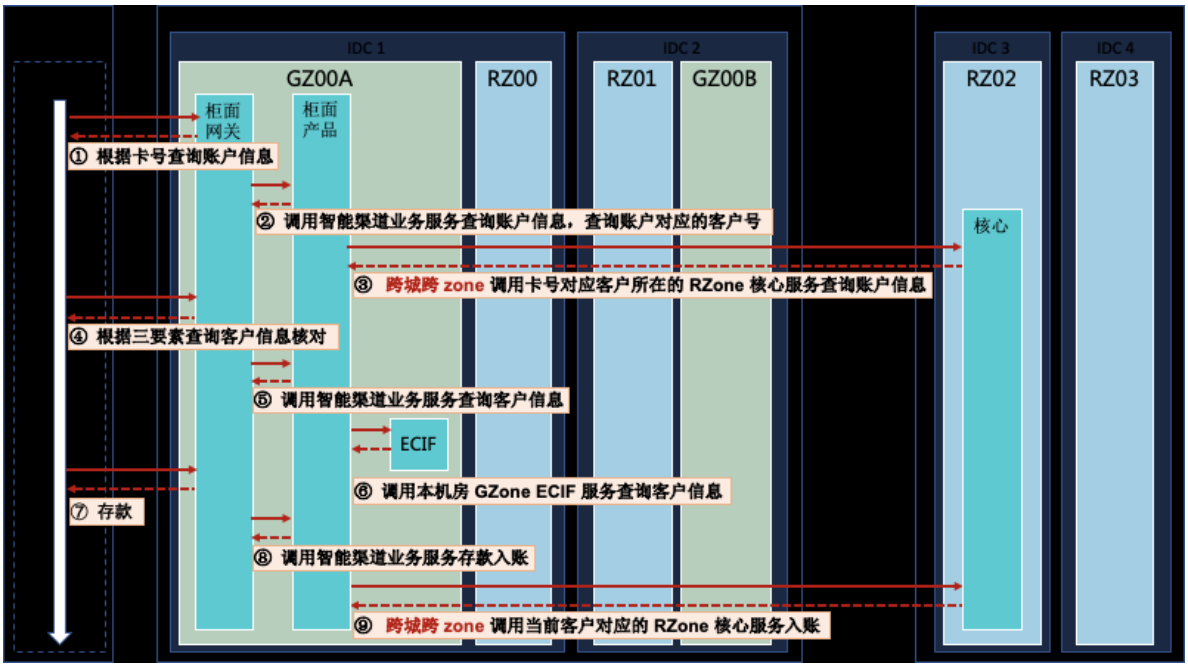
- 用户从客户端（PC、移动 APP 或其他端）发起一笔业务，通常这笔业务的第一个请求是登录请求。因为用户还没有登录，此时是无法判断这笔业务属于哪个单元的。因此，首次请求将被随机路由。
- 用户登录后，由认证中心（登录系统）负责将用户 ID 以某种约定好的格式写入响应 Cookie，并要求客户端在后续同一笔业务的每个请求中携带这个 Cookie。
- 接入路由网关将识别后续请求中的 Cookie 值，根据单元化服务路由规则，对 Cookie 值进行解析计算，确定业务所属单元，并将其路由到正确的单元中。

2.2.3. 跨 Zone 服务调用路由

一笔业务处理过程中，可能会涉及到跨 Zone 服务，有以下两种情况：

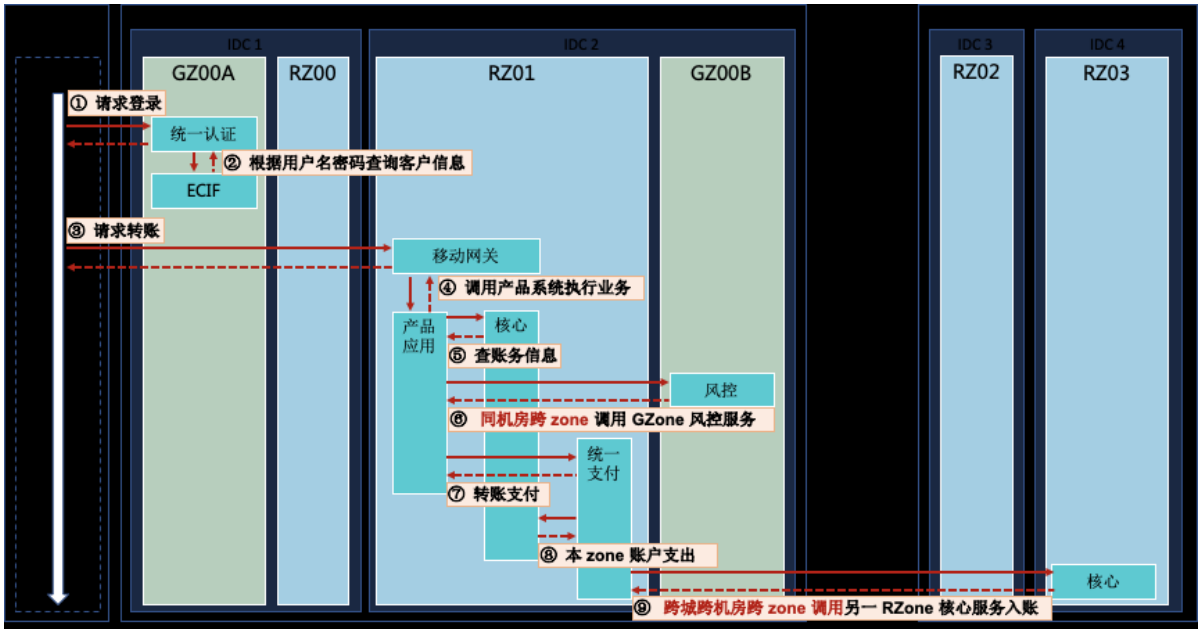
- **情况一：业务处理主要的应用在 GZone，但要使用 RZone 中的服务（分片数据）**
  - 如果涉及到的 RZone 与 GZone 在相同的城市，会产生同城跨 Zone 服务调用。
  - 如果涉及到的 RZone 与 GZone 在不同的城市，会产生跨城跨 Zone 服务调用。

示例：银行业柜面业务



- **情况二：业务处理主要的应用在 GZone，但要使用 GZone 中的服务（公共数据）**
  - 如果 GZone 与业务所属 RZone 在相同的城市，会产生同城跨 Zone 服务调用。
  - 如果 GZone 与业务所属 RZone 在不同的城市，会产生跨城跨 Zone 服务调用。

示例：银行业网银 APP 转账



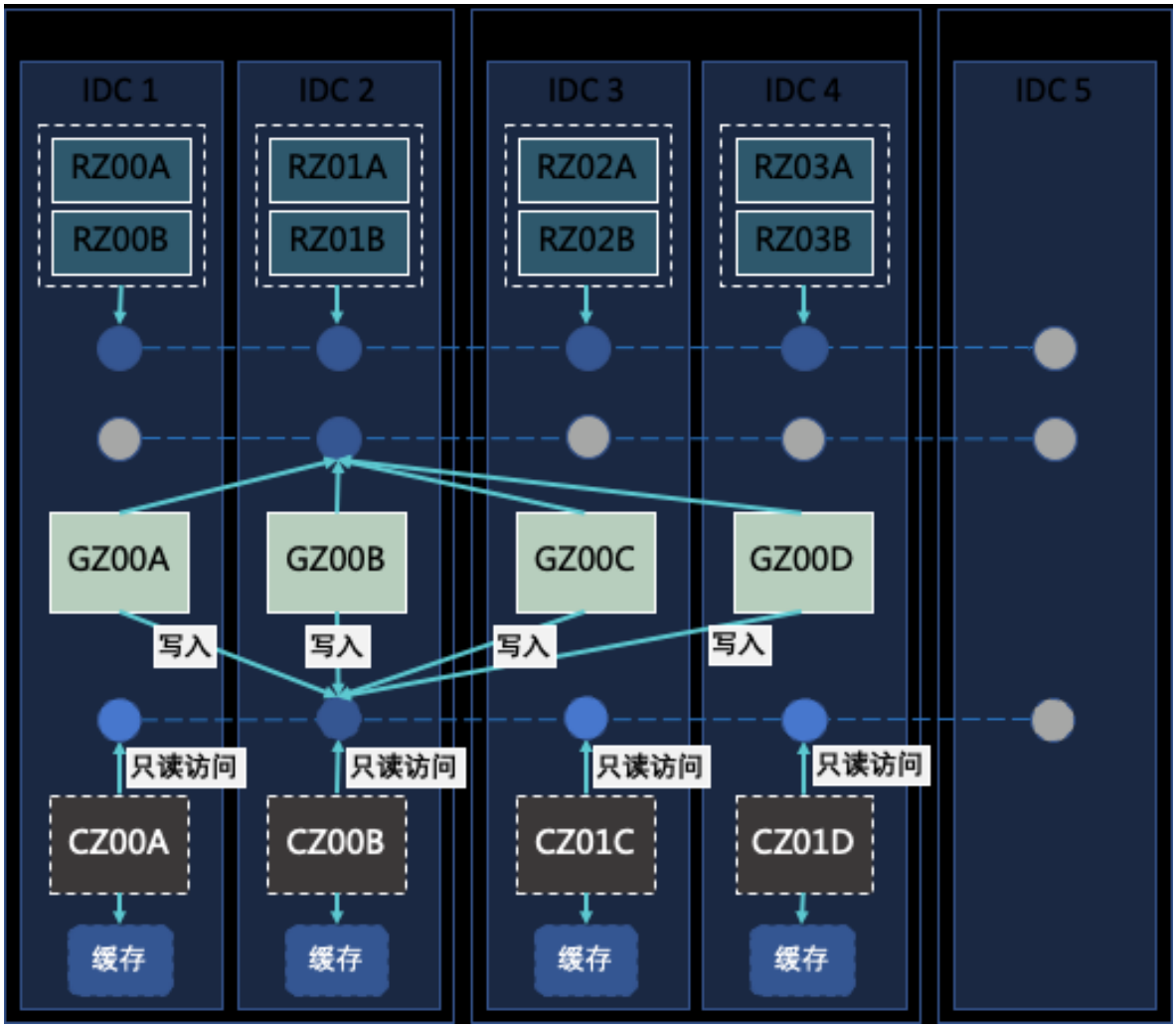
跨 Zone 服务调用对应用是透明的，无需应用关心具体如何实现跨 Zone 调用，由单元化产品的 RPC 服务框架自动完成。

但应用需要通过某种方式，向服务框架提供计算目标单元的信息。通常的做法是，在服务调用参数中包含数据分片标识信息，并在单元化产品中配置解析逻辑，供服务框架解析、计算出目标单元。

2.3. 单元规划

一个单元化架构系统，应尽可能提高 RZone 应用的比例，减少 GZone 应用，有限地使用 CZone。

在一个典型单元化架构的三地五中心系统中，可按下图规划单元。



规划说明如下：

- 在业务双活的两个主要城市部署 RZone，承载可分片的业务。至少在两地 4 个数据中心各部署一组 RZone，共 4 组，可根据业务需求增加。
- 在其中一个主要城市（City 1）部署主 GZone，承载无法水平拆分的业务和各类公共服务；在另一个主要城市（City 2）部署备 GZone，作为主 GZone 的异地灾备，

为了保障灾备能力可用，日常可承载少量业务流量。

- 如果两个主要城市网络延迟不大（ping < 7ms），可不设置 CZone，由 GZone 承载全部公共服务，以减低部署成本、业务应用研发成本、运维复杂性成本。

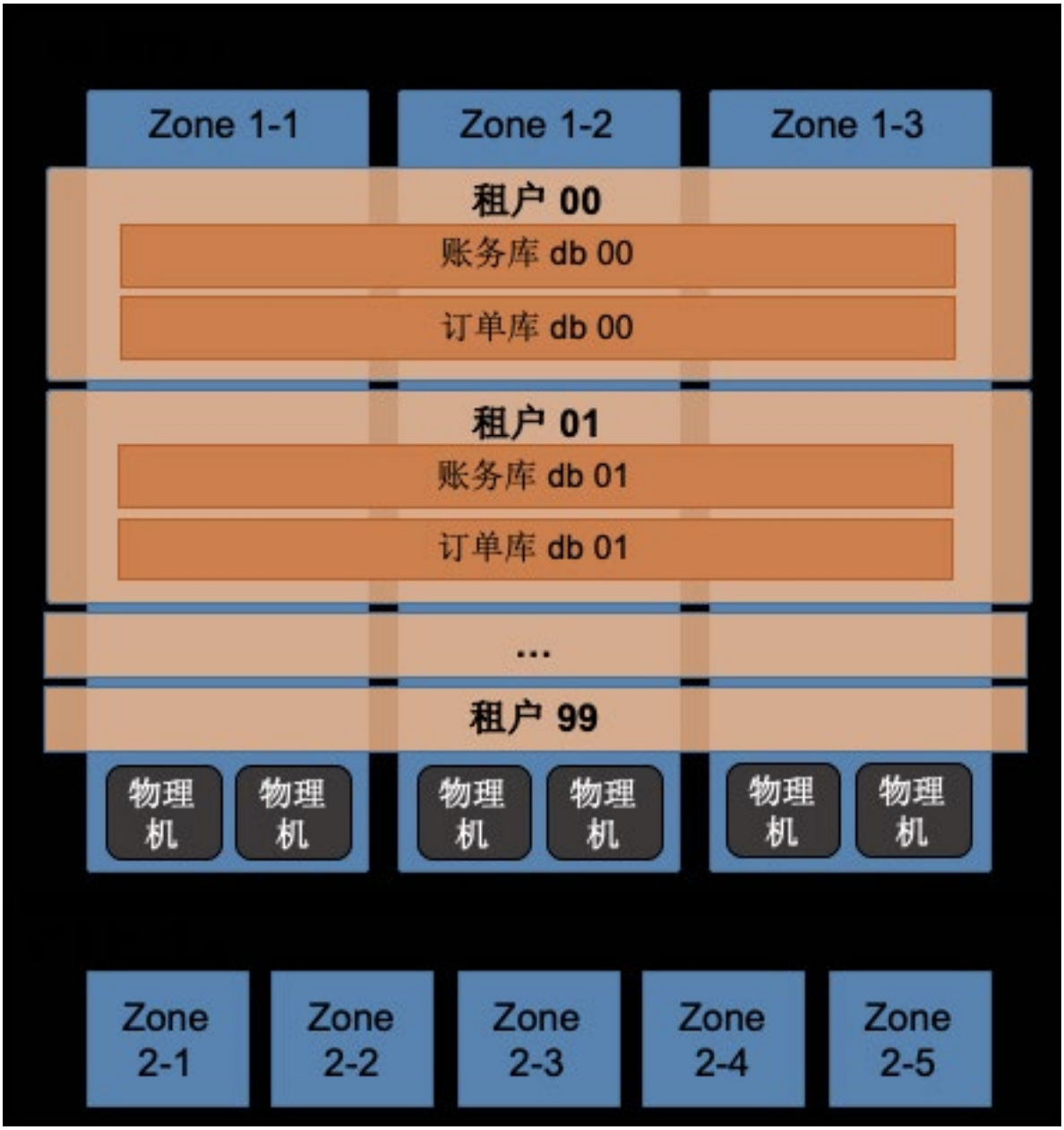
决策是否要增加 CZone 的方式如下：

- 实际运行过程中，如果观察到 City 2 的 RZone 或 备 GZone 跨城访问 City 1 的 GZone 数据已经对业务产生不可容忍的影响（比如 RT 过长），则触发决策新增 CZone 部署。
- 新增 CZone 除了部署以外，主要工作在于拆分出 CZone 应用和服务。

2.4. 单元数据库规划

蚂蚁单元化架构配合 OceanBase（OB）数据库使用时，需要对 OB 数据库进行合理规划，以更有效地利用 OB 数据库能力。

OceanBase 数据库基本概念



- **OB 集群**：一套物理资源独立、完整可提供服务的数据库系统。
- **OB Zone**：一个集群中的可用单元，即所谓“三副本”、“五副本”中的副本。每个 Zone 可部署多台物理服务器，以扩展容量。
- **OB 租户**：相当于 MySQL 中的“实例”，一个集群中的逻辑隔离单元，可独立配置可用的 CPU、内存资源，可独立控制主节点（Zone）。

- **Database（DB）**：与传统数据库中的 Database 概念相同，隔离表的逻辑单位。

OB 集群规划原则

一个单元化架构的大型业务系统中，根据不同的业务诉求，可以部署 1 套或多套 OB 集群，集群的划分原则如下：

- 重要系统数据单独部署集群。
- 需要强隔离的数据部署在不同的集群（集群间物理隔离）。
- 考虑集群的灰度升级，即成本允许的情况下，保证每个集群的每个 Zone 中至少有 2 台物理机。

OB 租户规划原则

租户是 OB 集群中可独立享有资源的最小逻辑单位，一个 OB 租户中可以创建多个 Database。OB 支持在租户级别控制主节点，这个特性对于单元化架构中的 RZone 数据库有非常重要的意义。需要根据业务的不同诉求，来选择 RZone 数据库租户的规划方式。

主节点的选择对于 RZone 数据库的意义是：一个 RZone 承载全系统业务的某几个分片，并且期望把这些分片的业务流、数据访问流都封闭在一个单元当中。为了封闭数据访问流，就要求 RZone 数据库的主节点与 RZone 部署在同一个机房当中，否则就会产生 RZone 跨机房甚至跨城市访问数据库。

假设一个单元化系统的业务被水平拆分为 100 份，在 OB 租户规划上有两种选择：

- 为每一份业务划分一个 OB 租户，总共 100 个租户。
- 多份业务划分到同一个租户，比如：每 10 份业务划分到一个租户，则总共有 10 个租户。

两种方式的区别：

规划方式	优势	劣势
每个分片一个租户	可以为每一个业务分片设置主节点，当需要进行容灾切换时，可以针对每个业务分片单独切换，切换粒度最小、最灵活。	租户数量多，每个租户都至少要占用最低要求的资源（CPU、内存），部署成本高，管理复杂。
一个租户多个分片	租户数量少，部署成本可控，管理较容易。	多个业务分片在同一个租户中，在进行容灾切换时必须一起切，切换颗粒度变粗，灵活性降低

您可根据业务实际诉求选择不同方式：如果成本充裕，对故障隔离粒度要求高，可选择第一种方式。如果对成本和管理复杂性有更多考虑，可以选择第二种方式。

2.5. 单元应用设计与开发

详情请参见《SOFAStack 单元化入门指南》中的 开发单元化应用。

3. 单元化产品

本节主要介绍单元化架构中涉及到的产品组件及其跟单元化相关的能力。

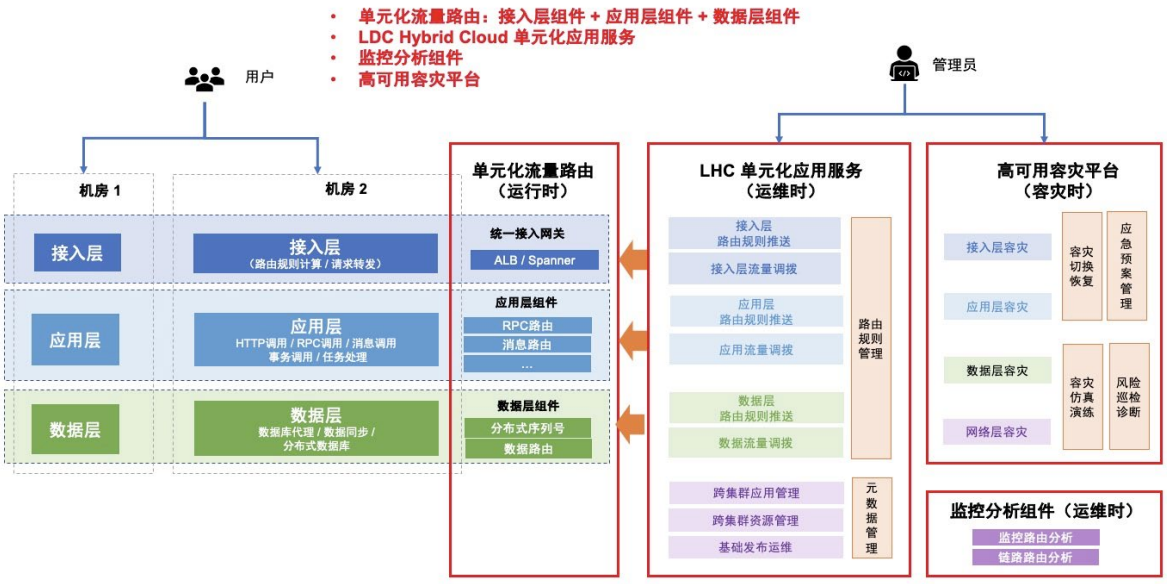
关于这些产品的全面功能介绍，请参见各产品的《产品简介》和《技术白皮书》手册。

3.1. 单元化产品组成

单元化产品，将以云原生为载体输出蚂蚁多年沉淀的异地多活单元化架构和金融级运维管理经验，包括分布式架构、发布运维、监控分析、容灾应急等方面的能力。同时提供可持续演进的架构升级路径，从同城双活、两地三中心、异地多活到异构基础设施下的混合云，满足不同场景的容灾、弹性、灰度需求，打造大规模高可用分布式架构，支撑金融业务创新。

单元化产品包含以下三大部分：

- 运行时：单元化流量路由能力，包含统一接入网关、应用层组件和数据层组件。
- 运维时：
  - 跨集群资源管理、发布运维和单元化管控能力，包含单元化应用服务（LHC）。
  - 单元化监控分析能力，包含实时监控组件和分布式链路跟踪组件。
- 容灾时：巡检诊断、容灾演练和切换能力，包含高可用容灾平台。



3.2. 统一接入网关

产品描述

统一接入网关，是将访问流量根据转发规则分发到多台后端服务器的流量分发控制服务。

在单元化场景下，按单元化路由规则（如用户 ID 到单元的映射关系以及单元间权重占比），将用户的 http 请求路由至目标机房和单元，实现流量动态调拨。

统一接入网关的作用如下：

- 拦截请求，识别目标应用的单元类型（如是 RZone 应用，还是 GZone 应用）
- 根据目标类型，进行路由转发，支持跨机房异地转发，实现单元化流量调拨。
  - GZone 路由：根据路由规则（单元权重占比），将用户 http 请求转发至目标单元的应用服务器中。
  - RZone 路由：从 cookie 中读取分片字段（如 uid），根据路由规则（用户 ID 到单元的映射关系），将用户 http 请求转发至目标机房的应用服务器中。

功能特性

- 运行时：接入层流量调控。根据业务信息和路由规则，将业务流量路由至正确的机房/单元，实现流量分拨。
- 容灾时：接入层流量调控。动态修改路由规则后，实时获取新的路由规则，将业务流量路由至新的目标机房/单元，从而实现流量分配、容灾切换。

部署架构

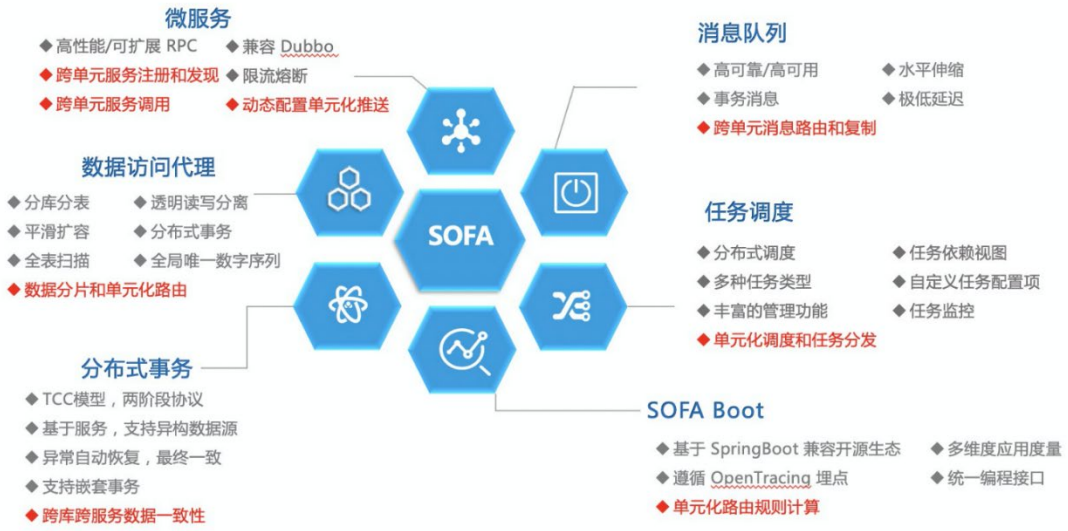
按机房部署，每个机房部署一套统一接入网关集群。

3.3. 中间件

蚂蚁分布式中间件的产品发展路径，一直秉承引领和拥抱业界先进标准和实践的理念，同时亦能满足传统金融架构的平滑迁移和融合适配，以稳妥应对业务升级变更，并积极

应对金融交易系统所面临的服务和数据扩展性、事务一致性、秒级容灾、弹性供给与调度等关键技术的挑战。

在单元化架构下，中间件从应用层到数据层均提供了单元化流量路由能力。



3.3.1 微服务

微服务主要包括三部分：**注册中心**、**RPC 服务框架**和**配置中心**，每部分都主要通过组件描述、功能特性、部署方式和产品架构来进行说明。

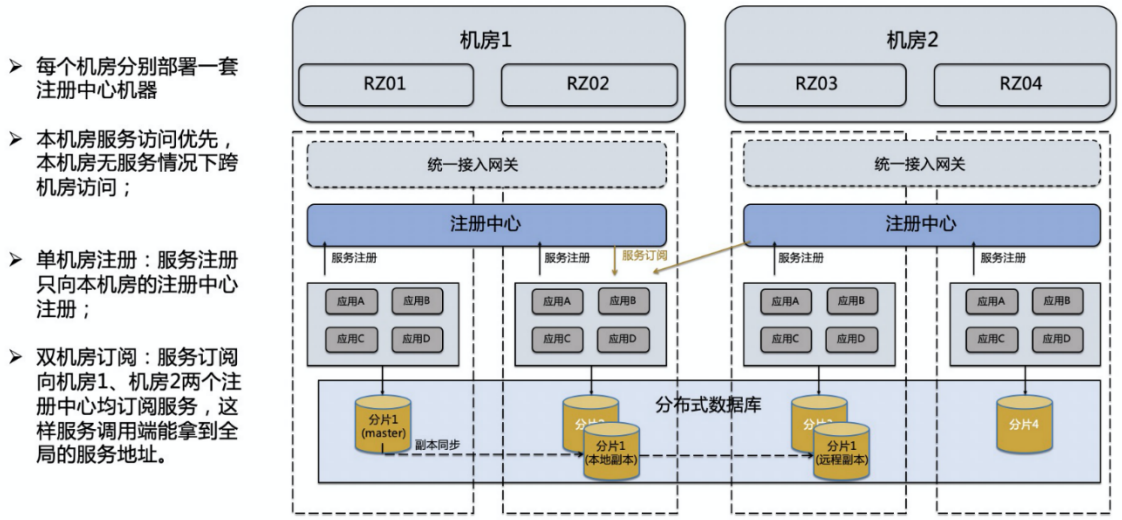
注册中心

- **组件描述**
  - 注册中心提供了服务注册和服务发现功能。
  - 在单元化架构下，提供本机房服务发现和跨机房服务发现的功能。服务框架进行微服务调用时，需要首先发现目标服务地址，通过注册中心进行本机房服务发现和跨机房服务发现，然后发起远程服务调用。
- **功能特性**
  - 本单元服务访问优先，本单元无服务的情况下跨单元访问。
  - 单机房注册：服务注册只向本机房的注册中心注册。
  - 双机房订阅：服务订阅向机房 1、机房 2 两个注册中心均订阅服务，这样服务调用端能拿到全局的服务地址。

- **部署方式**

  每个机房分别部署一套注册中心机器。

- **产品架构**



RPC 服务框架

- **组件描述**

  负责微服务 RPC 远程方法调用。

- **功能特性**

  服务框架进行微服务调用时，需要首先发现目标服务地址，通过注册中心进行本机房服务发现和跨机房服务发现，然后发起远程服务调用。

  - 本机房服务端调用：服务调用端向本地注册中心订阅服务，获取服务地址后，发起本机房服务调用。
  - 跨机房服务端调用：服务调用端向异地注册中心订阅服务，获取服务地址后，发起跨机房服务调用。

- **部署方式**

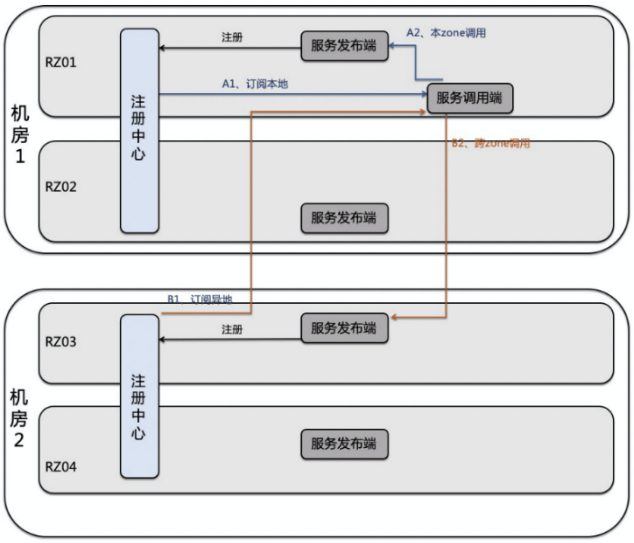
  作为 SDK 和业务系统集成在一起。

- **产品架构**

➢ 服务发布端，向本机房注册中心注册服务地址；

➢ A 本机房服务端调用：服务调用端向本机房注册中心订阅服务，获取服务地址之后，发起本机房服务调用；

➢ B 跨机房服务调用：服务调用端异地机房注册中心订阅服务，获取服务地址之后，发起跨机房服务调用；



配置中心

- **组件描述**
  - 执行配置的动态修改和推送。
  - 在单元化场景下，可以指定单元执行配置的动态修改和推送。
- **功能特性**

  指定单元执行配置的动态修改和推送。

3.3.2 任务调度

产品描述

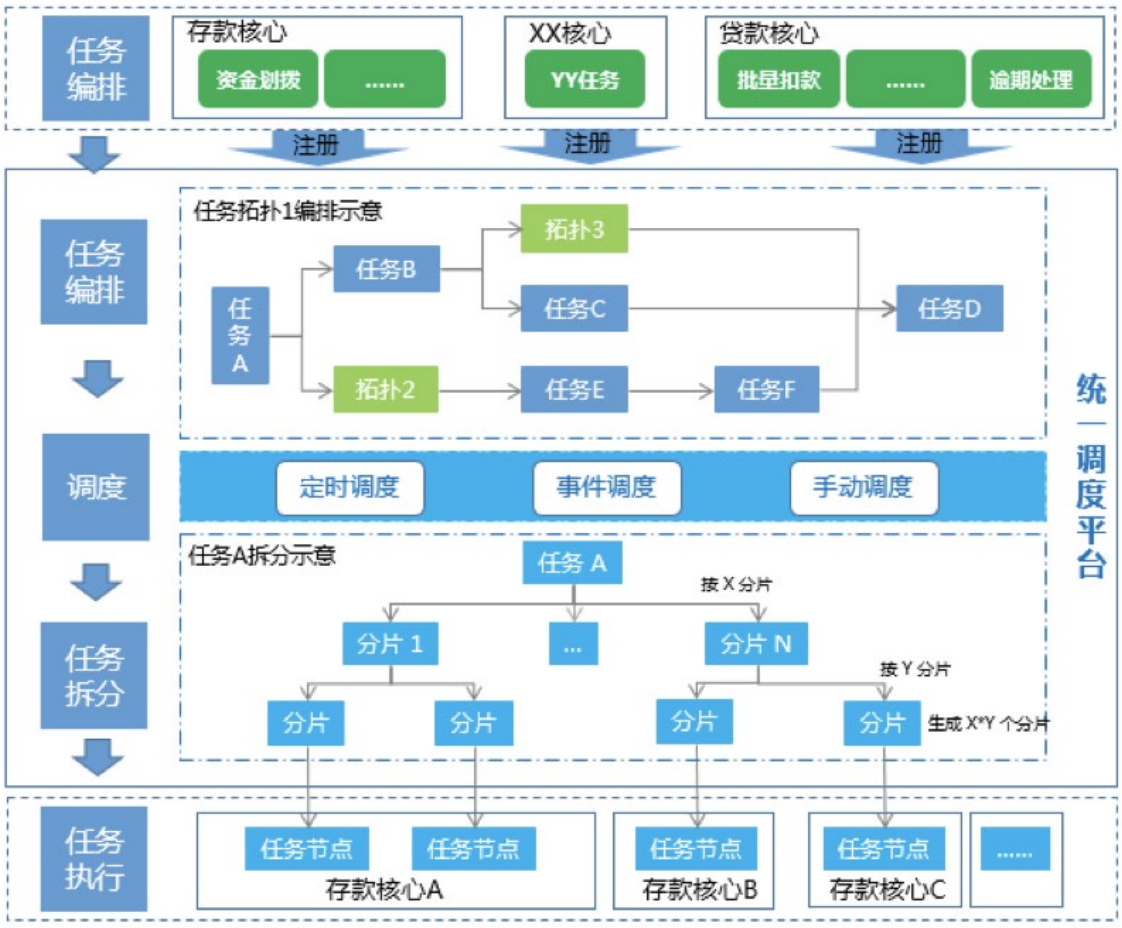
任务调度（Task Scheduler，简称 TS）提供分布式任务调度框架，实现任务的分布式处理，并能规范化、自动化、可视化和集中化地对金融企业不同业务系统的任务进行统一的调度和全方位的监控运维管理。

在单元化架构下，可以识别到任务客户端所在的单元信息，指定单元进行任务触发。

功能特性

- 客户端注册时携带所在单元信息。
- 控制台可以指定单元触发任务。

产品架构



3.3.3 分布式事务

产品描述

分布式事务（Distributed Transaction-eXtended，简称 DTX）是一款金融级分布式事务中间件，用来保障在大规模分布式环境下业务活动的最终一致性。在蚂蚁金服内部被广泛地应用于交易、转账、红包等核心资金链路，服务于亿级用户的资金操作。

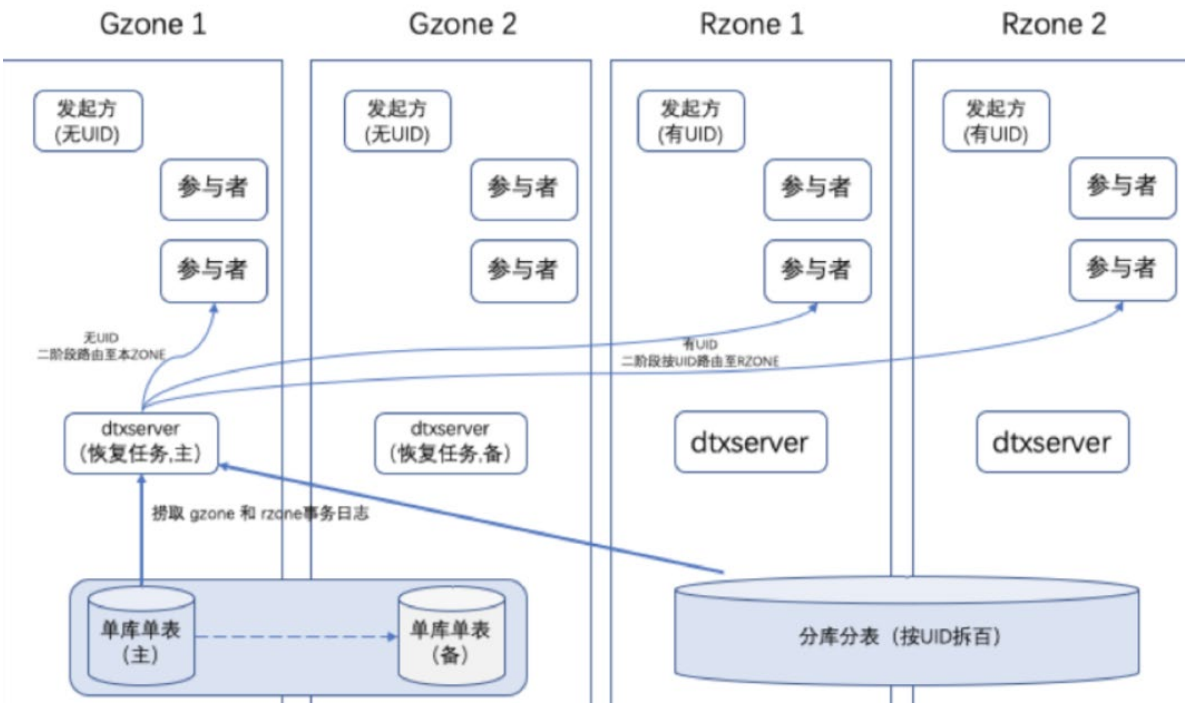
分布式事务可以与服务框架（如 SOFABoot、Spring Cloud、Dubbo）、数据源（如数据访问代理 ODP、RDS、MySQL、OceanBase）、以及消息队列等蚂蚁金融科技中间件产品配合使用，轻松实现服务链路级事务、跨库事务和消息事务等各种组合。

在单元化架构下，分布式事务在 GZone 和 RZone 都会发起，GZone 是无法拆分的业务无分片 ID，RZone 是可拆分的业务有分片 ID；GZone 发起方的事务，其数据会路由至单库单表数据库中存储；RZone 发起方的分布式事务，其数据会根据分片 ID 路由到

分库分表的数据库中存储。

产品架构

- GZone 发起事务
  - Dtxserver GZone 数据库不拆分，单库单表。
  - 事务发起后，主事务日志和分支事务日志，均写入 Dtxserver 的单库单表数据库中。
- RZone 发起事务
  - Dtxserver RZone 数据库，按照和业务数据库相同的拆分策略和数据库路由规则。
  - 事务发起后，Dtxserver 按照 ID 路由事务日志至目标库和表。
- 事务恢复
  - 事务恢复任务只运行在 GZone。
  - 恢复任务轮询 GZone 单库单表的数据库和 RZone 分库分表的数据库，根据事务日志中是否有分片信息，判断二阶段路由的目标 Zone。



3.3.4 数据访问代理

产品描述

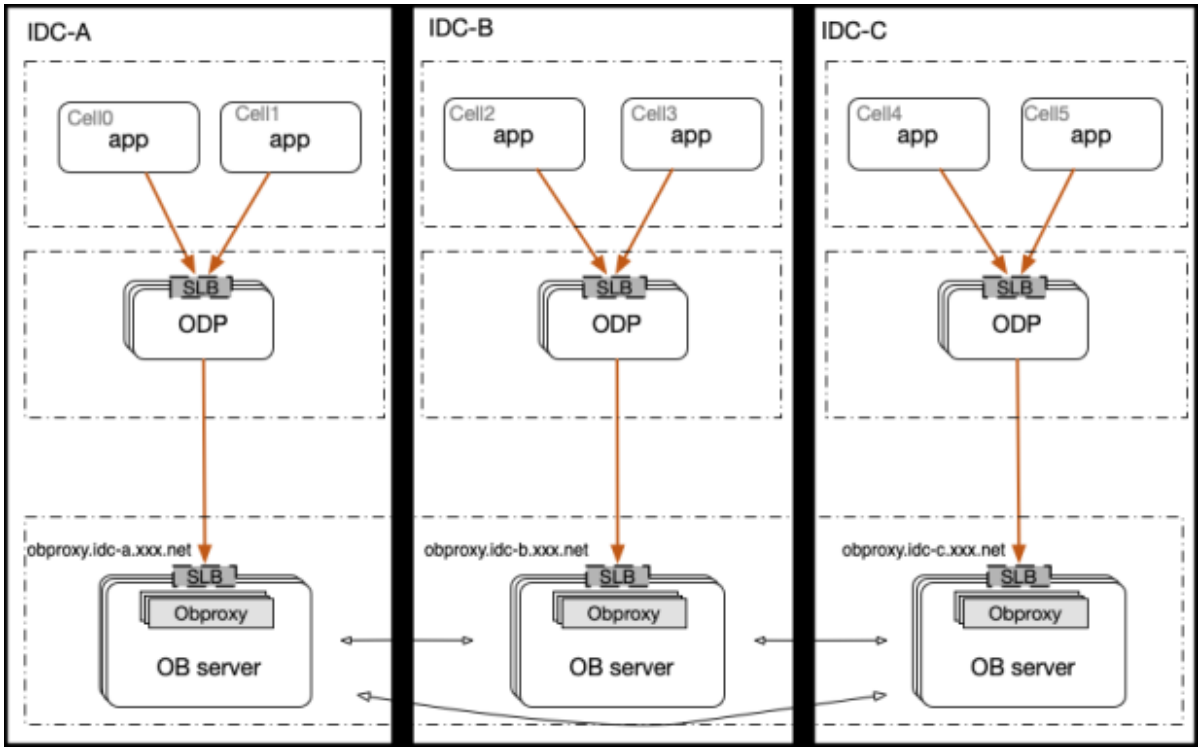
数据访问代理（Open Database Proxy，简称 ODP）能够解决海量请求下的数据访问瓶颈和数据库的容灾问题，提供水平拆分、平滑扩缩容、读写分离的在线分布式数据库服务，为海量数据访问提供低消耗、高性能、高可用的轻量级解决方案。

在单元场景下，ODP 将根据业务配置的数据库地址访问当前机房的数据库连接，完成数据的读写。

部署方式

每个机房部署一套 ODP 集群，业务应用在访问 ODP 时，寻址到当前机房的 ODP 集群，ODP 内部根据业务配置的数据库地址访问当前机房的数据库连接，完成数据的读写。

产品架构



3.3.5 消息队列

产品描述

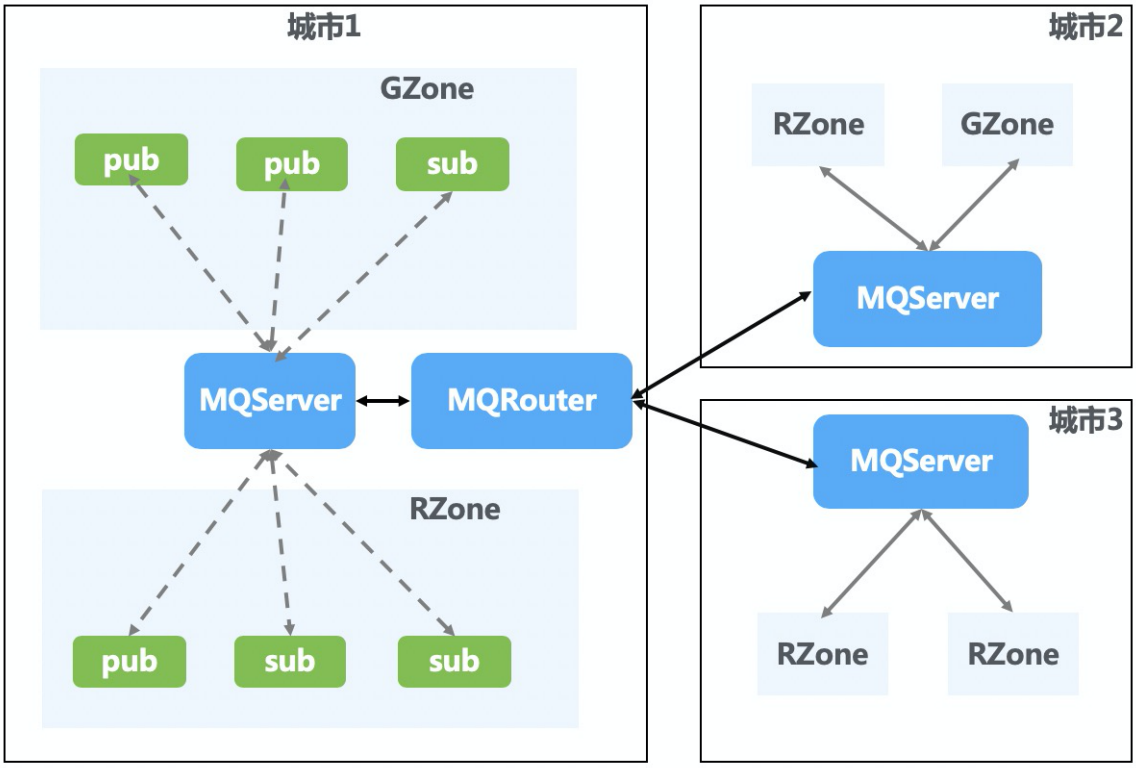
消息队列是基于 Apache RocketMQ 构建的分布式消息中间件，为分布式应用提供异步解耦和削峰填谷的能力，支持多种消息类型，提供高可靠、高吞吐量、高可用、事务一致性的异步通讯能力。

在单元化场景下，支持单元间逻辑隔离和跨单元的消息投递和消费。

功能特性

- 消息的发布和消费在单元间逻辑隔离。
- 支持多种消费模式：本单元消费和跨单元消费。
  - 本单元消费：消息发布端向本机房消息服务器 mqserver 发布消息，消息订阅端从本机房 mqserver 订阅消息。
  - 跨单元消费：消息发布端向本机房消息服务器 mqserver 发布消息，MQRouter 根据消息属性和单元化路由规则，路由至目标单元，再投递给消费者。
- 跨单元消费，可以通过控制台创建消息路由策略，实现消息的复制和投递。
  - RZone 消费：表示消息要在 RZone 消费，消息发送时在消息属性里面设置分片 id，服务端会根据分片信息和单元化路由规则，投递到对应的 RZone 单元。
  - GZone 消费：表示消息要在 GZone 消费，服务端会根据单元化路由规则（单元权重），投递到对应的 GZone 单元。
  - CZone 消费：表示消息要在 CZone 消费，如果一个城市有多个 CZone，服务端会根据单元化路由规则（单元权重）选择一个进行投递。

产品架构



3.4. 单元化应用服务（LHC）

产品描述

单元化应用服务，简称 **LHC**（LDC Hybrid Cloud）定位于在云原生基础设施之上，在多机房、多地域的多 Kubernetes 集群场景，提供应用管理、发布运维、流量调拨、配置同步等能力。作为开发运维人员日常接触的 PaaS 管控层产品，帮助解决应用和逻辑单元管理、按单元的配置变配、网络流量调拨、监控元数据配置等能力。 不仅满足金融场景下多活和容灾的业务需求，同时能够让基础设施享受到容器化基础设施、云原生架构的技术红利。

适用场景

提供从单 Kubernetes 集群向多活联邦集群演进的能力，适用于具备容灾能力的同城双活、两地三中心及更多机房级多活容灾场景。并可以配合 SOFAStack 各中间件产品、OceanBase 分布式数据库，形成单元化异地多活架构解决方案。

部署方式

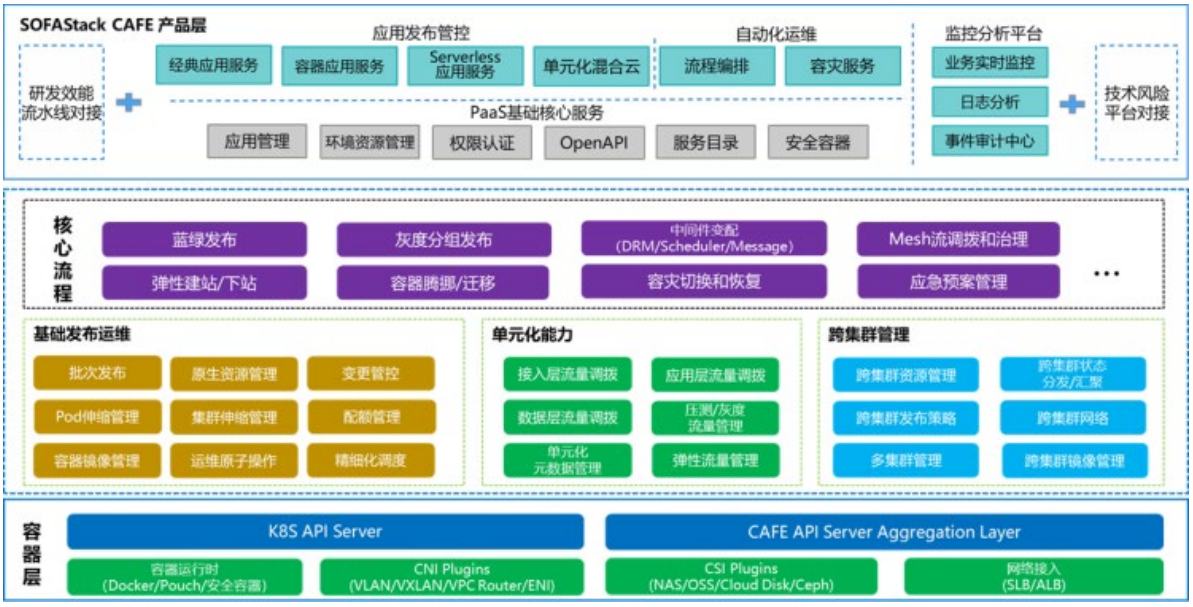
- 一个可用区（AZ）将部署一套完整的 kubernetes master + Etcd 集群，从而保证单机房出现故障时，其他机房的 kubernetes 仍可以正常工作。
- LHC 等核心 PaaS 产品每机房双副本高可用部署。
- Federation API Server 做双机房部署，本身无状态。Federation Controller 通过内部机制保证选主可靠性，可在应用、机房级故障下正确选主不脑裂。

产品架构

单元化应用服务，基于 Kubernetes 基础设施提供跨集群资源管理、发布运维及单元化管控能力。其纳管的每个集群，均为完整、标准的 Kubernetes 集群及相关扩展。在集群之上，通过联邦管控平面，协调各集群资源、应用和配置，以提供应用变更管控、分组发布、镜像管理、流量调拨、元数据管理、集群资源管理等功能。用户可以通过控制台、命令行或 SDK 以标准方式对单集群及联邦管控层进行交互。

作为分布式架构的平台管控层，单元化应用服务还承担与各相关系统的对接能力。

- 研发效能产品，可提供持续集成与交付功能，生成 LHC 产品的发布单。
- 与各 PaaS 基础产品对接，提供权限、资源、应用元数据、工作空间、部署单元等领域模型的管理能力。
- 与监控分析平台对接，向用户提供完整集群资源和业务应用的可观测性。



3.5. 监控分析

产品描述

分布式链路跟踪（Distributed System Tracing，简称 DST）是一款面向分布式架构、微服务架构和云原生架构的应用可观察性的金融级解决方案，帮助用户厘清应用间复杂的调用关系，迅速定位故障或缓慢节点。

实时监控（Realtime Monitor Service，简称 RMS），为大规模和复杂业务场景提供全方位的可观测性和洞察分析能力。

在单元化架构下，通过 DST 可以在链路追踪过程中，快速识别流量在单元间的走向，定位单元内的流量和跨单元的流量。通过 RMS，可以在应用监控过程中，按单元进行流量的聚合统计，提供单元间的流量趋势对比。

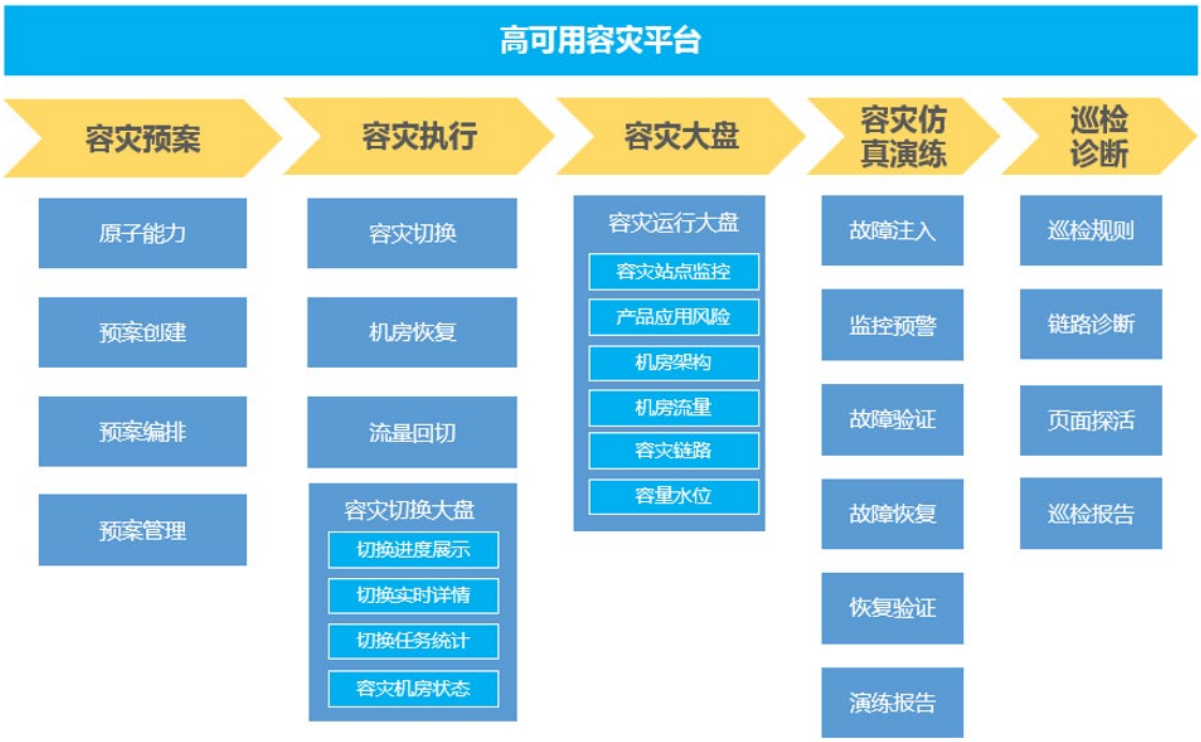
产品能力

3.6. 容灾平台

产品描述

高可用容灾平台（High Availability Service，HAS）是以容灾为主的高可用管控平台产品，可实现容灾方案的端到端整体能力，从客户业务到中间件、PaaS 以及 IaaS 整体的容灾切换及恢复、容灾规划、容灾模拟演练等能力，并包含整体机房及容灾状态的监控能力、容灾大盘展示、环境巡检、风险应急等。

产品架构


高可用容灾平台
容灾预案
容灾执行
容灾大盘
容灾仿真演练
巡检诊断
原子能力
预案创建
预案编排
预案管理
容灾切换
机房恢复
流量回切
容灾切换大盘
切换进度展示
切换实时详情
切换任务统计
容灾机房状态
容灾运行大盘
容灾站点监控
产品应用风险
机房架构
机房流量
容灾链路
容量水位
故障注入
监控预警
故障验证
故障恢复
恢复验证
演练报告
巡检规则
链路诊断
页面探活
巡检报告

4. 附录：单元化入门指南

如果您想上手体验如何使用蚂蚁产品实现单元化能力，请参见 SOFAStack 单元化入门指南中的操作指导。