### 2.1 未来发展策略

随着数据源不断增长，数据规模越来越大，种类越来越多，变化速度越来越快，企业想要对这些数据做出更实时的响应，数据形势正在不断变革。

十年前，数据几乎等同于关系数据库。如今，数据则可能呈现出各种形态，包括以HBase和MongoDB代表的NoSQL数据库、时间序列、像CockRoachDB和Spanner这样提供全局一致性的SQL存储，以及提供聚合日志文件查询功能的事件流。

多信源不只是趋势，而是已经发生的现实。

随着国内云计算的发展，企业上云进程加速，用户开始将应用系统逐步向云上迁移。云原生作 为一种先进的技术理念在行业内得到关注和认可，以容器技术、微服务架构和DevOps为代表，加速变革着传统的生产方式， 帮助企业构建更加适用于云中的应用服务，企业也借助云原生技术更好的发挥云的优势。

自阿里提出中台的概念以来，越来越多的企业和组织在逐步接受或向之靠拢，企业能否真正做到“以用户为中心”，并不断提升自己的用户响应力来追随甚至引领用户的脚步，持续规模化创新，终将决定企业能否在这样充满挑战和机遇的市场上笑到最后，在商业上长久保持创新活力与竞争力。

如何规避新技术带来的风险并有效获取对应的技术红利，是每一个从业者都应该密切关注和持续探索的问题。

#### 2.1.1 云原生

##### 2.1.1.1 理解云原生

定义

云原生应用旨在充分利用云计算模型，从而提高速度、灵活性和质量并降低部署风险。虽然名字中包含“云原生”三字，但云原生方案的重点并不是应用部署在何处，而是如何构建、部署和管理应用。

云原生方案与微服务架构类似。然而，尽管微服务可通过构建云原生应用来交付，可企业仍需采取许多措施，才能在生产中熟练地管理微服务。而要想享受云原生应用的各种益处，也并非一定需要微服务。很多企业都通过基于相同的原则，构建出更优秀的模块化单体式应用，从而取得云原生方案的种种效益。

向云原生应用的开发和交付转型，是一次全方位的变革，涉及企业的文化、流程、架构和技术。因此，这是企业的必经之路，而不是所要到达的目标。这段旅程代表着一个变革周期，而实现变革从来不是一件容易的事。

针对云原生技术在行业应用中的诸多挑战，结合用户数字化转型改造需求，建立相应的标准体系，规范和引导云原生技术在行业应用中的最佳实践，促进云计算领域有序健康发展。

##### 2.1.1.2 云原生架构分析

云原生应用架构包含三个特征：容器化、微服务和 DevOps。

###### 2.1.1.2.1 容器化——Kubernetes技术

容器其实已有10来年的历史，2013年开源的 Docker 容器引擎，被开发者所广泛熟悉，到如今发展成为包含容器云 PaaS 等一系列商业化应用实践。

容器技术具有占用资源少、部署快、易迁移等特点，容器可以理解为隔离环境的“运行时”，这也很好诠释了 Docker 集装箱的理念 --- Build, Ship and Run。容器看做是一个简易版的 Linux 环境(包括root用户权限、进程空间、用户空间和网络空间等)和运行在其中的应用程序。

云原生应用依靠容器来构建跨技术环境的通用运行模型，并在不同的环境和基础架构（包括公共、私有和混合云）间实现真正的应用可移植性。容器技术会利用操作系统虚拟化功能来划分多个应用的可用计算资源，并确保这些应用安全无虞、相互独立。云原生应用采用横向扩展模式，因此，只需添加更多应用实例，即可增加容量，而且这样的添加常通过容器基础架构中的自动化功能来实现。由于容器的费用低、密度高，因而可在同一虚拟机或物理服务器内托管大量容器，这使得容器成了交付云原生应用的理想之选。

放眼望去，容器化集群管理Kubernetes 正在被越来越多的企业青睐，Kubernetes已经为一种大规模部署容器化应用程序的标准。随着它不断的快速迭代发展， 企业在实际的应用部署过程中也面临着各种复杂的问题和挑战。如何克服K8S技术难关，使其更好的服务于容器化应用成为大家关注的重点。

容器云 PaaS 平台是云原生在企业重要的落地形态之一，它包含了 PaaS 本身，以及 DevOps、微服务等。

发展到 PaaS 的时候，用户不需要去关心底层的基础设施，只需要专注业务应用本身，容器 PaaS 以应用为中心，标准化、自动化应用的构建（Build）、交付（Ship）、部署运行（Run）流程，支撑应用的完整生命周期管理。通过容器云 PaaS 提供的丰富基础服务及之上的 SaaS 服务，提高 IT 设施自服务能力以及新业务的交付效率。

PaaS 最早其实是跟 IaaS 同步发展的，2011年时，国内出现了很多 PaaS 平台，包括 SAE、BAE等。第一代 PaaS 侧重提供支撑应用运行的应用引擎，我们现在所说的容器云 PaaS，则是基于云原生理念，融入 DevOps、微服务，解决了应用的完整生命周期管理问题。

在 IDC 的时代，用户需要管机房、物理机、包括网络、业务应用。上云之后，我们简化了这种资源的交付流程，用户获取计算、存储、网络资源变的更简单

Kubernetes是Google基于Borg开源的容器编排调度引擎，作为CNCF（Cloud Native Computing Foundation）最重要的组件之一，它的目标不仅仅是一个编排系统，而是提供一个规范，可以让你来描述集群的架构，定义服务的最终状态，Kubernetes可以帮你将系统自动得达到和维持在这个状态。

更直白的说，Kubernetes用户可以通过编写一个yaml或者json格式的配置文件，也可以通过工具/代码生成或直接请求Kubernetes API创建应用，该配置文件中包含了用户想要应用程序保持的状态，不论整个Kubernetes集群中的个别主机发生什么问题，都不会影响应用程序的状态，你还可以通过改变该配置文件或请求Kubernetes API来改变应用程序的状态。

Kubernetes在设计之初就充分考虑了针对容器的服务发现与负载均衡机制，提供了Service资源，并通过kube-proxy配合cloud provider来适应不同的应用场景。随着Kubernetes用户的激增，用户场景的不断丰富，又产生了一些新的负载均衡机制。目前，Kubernetes中的负载均衡大致可以分为以下几种机制，每种机制都有其特定的应用场景：

* Service：直接用Service提供cluster内部的负载均衡，并借助cloud provider提供的LB提供外部访问
* Ingress：还是用Service提供cluster内部的负载均衡，但是通过自定义LB提供外部访问
* Service Load Balancer：把load balancer直接跑在容器中，实现Bare Metal的Service Load Balancer
* Custom Load Balancer：自定义负载均衡，并替代kube-proxy，一般在物理部署Kubernetes时使用，方便接入公司已有的外部服务

K8S技术相对复杂，受限于人力时更需了解如何能简单落地。比如一共需要哪些基础服务来保证大应用落地、如何构建一套简单的外部程序、线下线上如何采用统一的K8S环境来保证开发测试的一致性、统一的K8S接口该如何设计等。

###### 2.1.1.2.2 微服务

微服务是一种架构模式，它提倡将单一应用程序划分成一组小的服务，服务之间互相协调、互相配合，为用户提供最终价值。每个服务运行在其独立的进程中，服务与服务间采用轻量级的通信机制互相沟通（通常是基于HTTP的RESTful API）。以往企业应用主要是面向服务的架构（SOA），SOA 是一种粗粒度、松耦合服务架构，服务之间通过简单、精确定义接口进行通讯，不涉及底层编程接口和通讯模型。它的缺点是架构重，难以利用云的一些特点和优势。微服务倡导细粒度的轻量级应用架构，每一个服务相对独立的，具有轻量级、易迁移、更高效等特性。

微服务可以提供以下能力：

* 多编程语言支持:支持多种编程语言的适配，允许采用不同的语言构建微服务,Java Go Python Cpp
* 服务通信:微服务之间实现数据交换的能力,支持RESTful风格协议，同时可支持其它协议方式、自定义协议,支持多种服务通信方式
* 服务注册:服务在启动后自动提供自身的元数据以被请求方发现,在服务发生故障或不可用的情况下，自动对服务进行注销
* 服务发现:支持服务节点在注册中心被增加或剔除后，能自动感知,保证微服务节点数据的准确性和及时性
* 认证鉴权：微服务间服务调用的身份认证与访问鉴权，包括微服务与注册中心、配置中心之间的认证，以 及微服务之间的认证, token、session、cookies、（API级别认证和鉴权）
* 流量控制：支持多种流量控制策略，如请求量总并发，瞬时并发，时间窗口并发等，负载情况限流策略,支持多种流控算法，如令牌桶，桶漏，计数器等
* 熔断隔离：当服务的输入负载激增时， 避免服务被迅速压垮导致雪崩效应，而对负载进行断路的一种方式,支持多种熔断判断(熔断请求数，异常百分比 )和熔断隔离方式：手动熔断、自动熔断、实例隔离…
* 服务容错：依赖的服务出现故障时候能够自动或者半自动的在应用层面实现平滑的失败处理,快速失败，失败重试，失败转移，并行调用，故障沉默，失败缓存等
* 服务路由：根据预置路由策略将请求转发到微服务上，且能够生效,支持多种路由策略，如三动态和静态权重路由、规则路由等
* 负载均衡：微服务平台客户端和服务端的负载均衡能力,支持按照权重、轮询、随机等负载策略 服务契约,支持统一的微服务契约接口描述，符合OpenAI规范接口：swagger..
* 健康检查：支持主动和被动多种健康检查方式，如服务状态数据上报、log日志上报，主动监控数据采集 (队列长度、内存大小)
* 故障注入：通过向微服务应用注入延迟和中止故障，观察应用程序是否能够处理故障，保证故障处理预 期设计目标

基于服务的架构（如微服务）提倡构建松散耦合的模块化服务。其他模块化架构

方案（如迷你服务）采用基于服务的松散耦合设计，可帮助企业提高应用创建速

度，但不会增加复杂性。

目前，在微服务技术架构实践中主要有侵入式架构和非侵入式架构两种实现形式。侵入式架构是指服务框架嵌入程序代码，实现类的继承，其中以 Spring Cloud 最为常见。

非侵入式架构则是以代理的形式，与应用程序部署在一起，接管应用程序的定义引自《云计算发展白皮书 2018 年》第二章-云计算发展特点云计算开源产业联盟 云原生技术实践白皮书（2019 年）网络且对其透明，以服务网格为代表。在中国信息通信研究院制定的《分布式应用架构技术要求第一部分：微服务平台》中也对这两种架构进行了详细的描述，并提出了相应的技术要求。

* Spring Cloud

Spring Cloud 是一系列框架的有序集合。它利用 Spring Boot的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务注册发现、配置中心、消息总线、负载均衡、断路器、数据监控等，都可以用 Spring Boot 的开发风格做到一键启动和部署。Spring Cloud并没有重复制造轮子，它只是将目前各家公司开发的比较成熟、经得起实际考验的服务框架组合起来，通过 Spring Boot 风格进行再封装屏蔽掉了复杂的配置和实现原理，最终给开发者留出了一套简单易懂、易部署和易维护的分布式系统开发工具包。

* Service Mesh

Service Mesh 处理服务间请求/响应的可靠传递，并可用于服务治理、遗留系统的零侵入接入以及异构框架开发的微服务。ServiceMesh 作为服务间通信的基础设施层，是应用程序间通讯的中间层，实现了轻量级网络代理，对应用程序透明，解耦了应用程序的重试/超时、监控、追踪和服务发现。Service Mesh 的开源软件包括 Istio、Linkderd、Envoy、SOFAMesh、Dubbo Mesh 等。

###### 2.1.1.2.3 DevOps

DevOps 是一组过程、方法与系统的统称，用于促进开发（应用程序/软件工程）、技术运营和质量保障（QA）部门之间的沟通、协作与整合。它的出现是由于软件行业日益清晰地认识到：为了按时交付软件产品和服务，开发和运营工作必须紧密合作。

DevOps 旨在统一软件开发和软件操作，与业务目标紧密结合，在软件构建、集成、测试、发布到部署和基础设施管理中大力提倡自动化和监控。DevOps 的目标是缩短开发周期，增加部署频率，更可靠的发布。用户可通过完整的工具链，深度集成代码仓库、制品仓库、项目管理、自动化测试等类别中的主流工具，实现零成本迁移，快速实践DevOps。

DevOps 帮助开发者和运维人员打造了一个全新空间，构建了一种通过持续交付实践去优化资源和扩展应用程序的新方式。DevOps和云原生架构的结合能够实现精益产品开发流程，适应快速变化的市场，更好的服务企业的商业目的。

要完成云原生应用的构建之路，开发和 IT 运维团队必须进行多方面的变革，以便更加快速高效地构建和部署应用。无论身处哪个行业、规模如何，所有企业都需要周全地考虑各种活动、技术、团队和流程，因为这些要素综合起来才能实现 DevOps 文化。要想充分利用新技术，采用更加快速的方案，实现更为密切的合作，企业必须切实遵循 DevOps 的原则和文化价值，并围绕这些价值来进行组织和规划。

身处日新月异的数字化创新时代，管理多个分布式环境、高度定制的传统应用以及各种全新的应用工作负载会十分复杂，DevOps 对于一些企业而言可能会充满挑战。对于在应用组合中拓宽。

##### 2.1.1.3 云原生产业发展趋势

###### 2.1.1.3.1 云原⽣助⼒⼈⼯智能从云端扩展⾄边缘

微服务架构的盛行正在加速 AI 生态系统中功能解耦。AI 模型中不同粒度的功能如分类、聚类、识别、预测、自然语言处理、卷积、重现等都再向微服务化转变。借助云原生技术，云端智能正在加速向边缘智能扩展。要实现边缘智能化，需要把云上的模型，快速迁移到线下，将云上智能改造为边缘可用的轻量级智能，适配边缘软硬件环境和使用场景。

容器作为一种轻量级操作系统隔离技术，发挥着重要作用，这种轻量级的打包技术将微服务之间依赖关系与预配置文件打包，统一环境和预配置并输出标准化的镜像，很好的解决环境一致性问题，实现快速的扩展和移植，将云端的应用快速在边缘部署。同时，数以亿计云计算开源产业联盟边缘设备汇聚成大量的小型数据中心，利用容器编排引擎，可以轻松完成超多规模集群的分发调度，实现云端的互联互通。

###### 2.1.1.3.2 运营商搭载云原⽣强化 5G ⽹络能⼒

5G 是 ITU、3GPP 等国际标准组织定义的新一代移动通信标准，被广泛认为是将改变社会的通信技术。2017 年启动的 5G R15 标准定义了服务化的系统架构，采用云原生技术中的微服务模式定义了近20个网络能力（NF）接口，强调自包含、独立管理等微服务原则，对传统通信网中紧耦合和功能过于集中的网元进行重构和分解。在 NF 接口的技术实现上，R15 标准可以使用 NFV/SDN 标准化 IT 技术，底层是云计算中通用的 REST 风格接口、x86 硬件平台和虚拟化技术。通过上述标准，5G 网络解决了原有通信网软硬件平台封闭、弹性不足，业务部署和变更困难等固有问题，也为云原生技术应用到 5G 网络及其承载的上层应用打好基础。

云计算技术已应用于现有的通信网络，包括基于 OpenStack 平台

实现网元虚拟化和快速部署，基于开放的编排标准实现资源编排。这些能力对于 5G 网络愿景来说还远远未够，包括分钟级的网络能力部署和变更要求、和云计算主流的容器管理平台 Kubernetes 的互操作等。

在已经发布的 ETSI、NGMN 等机构的 5G 白皮书上，通信运营商、设备商都提出了引入云原生技术到 5G 网络中以实现其愿景，相关的建议包括：对 NF 服务更轻量级的容器化封装，使用 Kubernetes 等云计算动态平台实现弹性部署和开发运营一体化。参与云原生基金会旗下各个开源项目，确保与 5G 业务需求的匹配并增加所需的功能；改造边缘机房为数据中心实现低时延和本地管控的行业应用。

从目前已经试点商用的 5G 案例和业务初期的需求来看，云原生技术非常适合应用于 5G 多接入边缘计算场景，充分利用 5G 用户面/控制面分离、云化改造的边缘机房和 IT 基础设施等能力，通过云原生技术与不同合作伙伴云化应用更为无缝地集成，实现低时延、分钟级弹性部署、支持异构多平台等特性的 5G 应用。

从 5G 核心网乃至通信网的中长期演进来看，云原生技术可以应用到网络能力的研发运营一体化流程中，与持续发展的白盒化通信软硬件平台、人工智能技术相辅相成，促成通信网真正实现开放、智能、敏捷、软硬件解耦、云网融合的转型。

###### 2.1.1.3.3 云原⽣领域的标准化进程将持续深化

我国云原生技术的标准建设仍处于起步阶段，标准较为分散、不成体系。需要国内的标准化组织统筹规划，做好云原生领域的标准规划和顶层设计，充分发挥标准的引领指导作用。

中国信息通信研究院一直紧跟云计算领域风向，密切关注云原生技术的发展变化，相继制定了《可信云开源解决方案评估方法第二部分：容器类》、《分布式应用架构技术能力要求第一部分:微服务平台》、《研发运营一体化（DevOps）能力成熟度模型》系列标准、《函数即服务评估方法》等云原生技术领域的一系列标准规范，对相关产品的服务能力进行要求，促进行业了健康有序发展，引导服务贴近产业实际需求。

云原生技术仍在不断成熟发展，越来越多的开源技术被引入生产，但是开源技术的大规模商用需要充分的评估与实践，而基于开源技术的商业发行版差异性较大，服务水平参差不齐。以用户需求为中心的技术规范和能力要求将会持续深化。

###### 2.1.1.3.4 联盟组织在云原⽣产业中的角⾊日趋重要

虽然云原生技术已在行业中广泛应用，但行业间的应用程度存在较大差异。产业链的分布较为分散，技术提供方与业务需求方难以实现需求的精准对接，需要中立的第三方机构来承担桥接作用，将云原生领域的分散力量凝聚起来，进一步推动云原生应用的均衡发展。

由中国信息通信研究院牵头，联合云原生技术提供商与众多行业的用户代表共同成立的云原生产业联盟，一直致力于促进我国云原生行业间的交流，加强技术服务商与行业用户间的沟通，推进云原生技术在我国的快速发展与落地实践。联盟作为云原生行业的对话平台，将行业的资源汇聚一起，共同构建技术带动实践、实践反哺技术的良性生态，不断提升着云原生技术在市场的影响力。

#### 2.1.2 中台战略

##### 2.1.2.1 中台定义

在当今互联网时代，⽤户才是商业战场的中心，为了快速响应用户的需求，借助平台化的力量可以事半功倍。不断快速响应、探索、挖掘、引领⽤户的需求，才是企业得以⽣存和持续发展的关键因素。

那些真正尊重用户，甚⾄不惜调整⾃己颠覆⾃己来响应⽤户的企业将在这场以⽤户为中心的商业战争中得以⽣存和发展；⽽反之，那些在过去的成就上故步⾃封，存在侥幸⼼理希望⽤户会像之前一样继续追随⾃己的企业则会被用户淘汰。

⽽平台化之所以重要，就是因为它赋予或加强了企业在以用户为中心的现代商业战争中最最最核心的能力：⽤户响应力。这种能力可以帮助企业在商战上先发制⼈，始终抢得先机。可以说，在互联网时代，商业的斗争就是对于用户响应力的比拼。

说起中台，最先想到的应该就属是阿⾥的“⼤中台，⼩前台”战略。阿⾥⼈通过多年不懈的努力，在业务的不断催化滋养下，将⾃己的技术和业务能力沉淀出一套综合能力平台，具备了对于前台业务变化及创新的快速响应能力。

海尔也早在⼗年前就已经开始推进平台化组织的转型，提出了“平台⾃营体⽀撑⼀线⾃营体”的战略规划和转型⽬标。构建了“⼈单合一”、“⽤户付薪” 的创客文化，真正将平台化提⾼到了组织的⾼度。

华为在几年前就提出了“⼤平台炮火支撑精兵作战”的企业战略，“让听得到炮声的人能呼唤到炮火” 这句话形象的诠释了大平台⽀撑下小前台的作战策略。这种极度灵活又威力巨⼤的战法，使之可以迅速响应瞬息万变的战场，一旦锁定目标，通过大平台的炮火群，迅速精准对于战场进行强大的火⼒支援。

但是平台化并不是一个新概念，很多企业在这个方向上已经做了多年的努力和积淀。那为什么最近几年“中台”这个相对较新的概念又会异军突起？对于企业来讲，传统的“前台 + 后台”的平台化架构又为什么不能满足企业的要求呢?

* 前台：由各类前台系统组成的前端平台。每个前台系统就是一个用户触点，即企业的最终用户直接使用或交互的系统，是企业与最终用户的交点。例如用户直接使用的网站，手机 App，微信公众号等都属于前台范畴。
* 后台：由后台系统组成的后端平台。每个后台系统一般管理了企业的一类核心资源（数据 + 计算），例如财务系统，产品系统，客户管理系统，仓库物流管理系统等，这类系统构成了企业的后台。基础设施和计算平台作为企业的核心计算资源，也属于后台的一部分。大多数企业已有的后台，要么前台根本就用不了，要么不好用，要么变更速度跟不上前台的节奏。

我们看到的很多企业的后台系统，在创建之初的目标，并不是主要服务于前台系统创新，而更多的是为了实现后端资源的电子化管理，解决企业管理的效率问题。这类系统要不就是当年花大价钱外购，需要每年支付大量的服务费，并且版本老旧，定制化困难；要不就是花大价钱自建，年久失修，一身的补丁，同样变更困难，也是企业所谓的“遗留系统”的重灾区。

但就算是新建的后台系统，因为其管理的是企业的关键核心数据，考虑到企业安全、审计、合规、法律等限制。导致其同样往往⽆法被前台系统直接使用，或是受到各类限制⽆法快速变化，以⽀持前台快速的创新需求。

此时的前台和后台就像是两个不同转速的⻮轮，前台由于要快速响应前端用户的需求，讲究的是快速创新迭代，所以要求转速越快越好；⽽后台由于⾯对的是相对稳定的后端资源，⽽且往系统陈旧复杂，甚至还受到法律法规审计等相关合规约束，所以往往是稳定至上，越稳定越好， 转速也自然是越慢越好。

所以，随着企业务的不断发展，这种“前台 + 后台”的⻮轮速率“匹配失衡”的问题就逐步显现出来。

随着企业业务的发展壮大，因为后台修改的成本和⻛险较⾼，所以驱使我们会尽量选择保持后台系统的稳定性，但还要响应用户持续不断的需求，自然就会将大量的业务逻辑 (业务能力) 直接塞到了前台系统中，引入重复的同时还会致使前台系统不断膨胀，变得臃肿，形成了一个个⼤泥球的“烟囱式单体应用”。渐渐拖垮了前台系统的“⽤户响应⼒”，用户满意度降低，企业竞争力也随之不断下降。

对于这样的问题，Gatner 在 2016 年提出的一份《Pace-Layered Application Strategy》报告中，给出了一种解决方案，即按照“步速”将企业的应用系统划分为三个层次（正好契合前中后台的三个层次），不同的层次采用完全不同的策略。

而 Pace-Layered Application Strategy 也为“中台”产生的必然性，提供了理论上的支撑。

在这份报告中 Gatner 提出，企业构建的系统从 Pace-Layered 的⾓度来看可以划分为三类: SOR(Systems of record )，SOD(Systems of differentiation) 和 SOI(Systems of innovation)。

处于不同 Pace-Layered 的系统因为⽬的不同，关注点不同，要求不同，变化的“速率”自然也不同，匹配的也需要采⽤不同的技术架构，管理流程，治理架构甚至投资策略。

⽽前面章节我们提到的后台系统，例如 CRM、ERP、财务系统等，它们⼤多都处于 SOR 的 Pace-Layered。这些系统的建设之初往往是以规范处理企业底层资源和企业的核⼼可追溯单据 (例如财务单据，订单单据) 为主要⽬的。它们的变更周期往往比较⻓，⽽且由于法律律审计等其他限制，导致对于它们的变更需要严谨的申报审批流程和更高级别的测试部署要求，这就导致了它们往往变化频率低，变化成本高，变化⻛险高，变化周期⻓。⽆法满⾜由⽤户驱动的快速变化的前台系统要求。

我们又要尽力保持后台 (SOR) 系统的稳定可靠，⼜要前台系统 (SOI) 能够⼩而美，快速迭代。

中台是真正为前台而生的平台（可以是技术平台，业务能力甚至是组织机构），它存在的唯一目的就是更好的服务前台规模化创新，进而更好的响应服务引领用户，使企业真正做到自身能力与用户需求的持续对接。

中台就像是在前台与后台之间添加的⼀组“变速⻮轮”，将前台与后台的速率进行匹配，是前台与后台的桥梁。它为前台而生，易于前台使用，将后台资源顺滑流向用户，响应用户。

中台很像 Pace-Layered 中的 SOD，提供了比前台（SOI）更强的稳定性，以及⽐后台（SOR）更高的灵活性，在稳定与灵活之间寻找到了⼀种美妙的平衡。

##### 2.1.2.2 中台发展类型

###### 2.1.2.2.1 技术中台

研发中台是关注应用开发效率的管理平台。软件开发和系统建设是一项工程，涉及项目管理、团队协作、流程、测试、部署、运营、监控等方面。如何将在企业应用开发过程中的最佳实践沉淀为可重用的能力，从而更好地快速迭代开发创新型的应用，也是很多企业目前的一个关注点。这个关注点也是企业能力的体现，即研发中台。研发中台为应用开发提供了流程和持续交付的能力，包括敏捷开发管理、开发流水线、部署流水线、持续交付。敏捷管理一般由问题、迭代、实施等组成，并管理研发人员的日常工作和任务。开发流水线则涉及源代码的版本管理、分支的创建、合并和提交，半成品的构建、存储和使用以及产成品的构建。将产成品部署到指定环境并上线运行是部署流水线的职责。线上的应用需要监控，包括基础设施监控、应用监控、日志洞察、浏览器监控、链路分析和追踪等功能。研发中台为应用的开发提供了流程、质量管控和持续交付的能力。

###### 2.1.2.2.2 研发中台

消费者接触得最多的企业前台触点在移动端。如何保障移动端的迭代效率和稳定性也是企业需要着重考虑的。一个电商业务一开始可能只是一个工具型的 App，完成对商品全生命周期的闭环支持。随着在业务中台基础上发展出相似业务，需要平台级的移动端开发支持。继续深化发展可能还需要支持多业态。因此为快速开发移动 App、H5 和小程序以支撑前台业务发展所进行的最佳实践就逐渐沉淀为移动中台。

移动 App 与其他前端技术比较，有其特殊性。比如移动 App 作为一个 C/S 架构，其发版模式需要通用应用市场的审核，而其客户端的更新是使用者控制的，提供远程配置、动态更新有助于控制 App 端。

移动业务是在线业务，对网络存在强依赖，而移动链路本身的稳定性和连通率等相比有线网络有一定的不足，因此消息推送的实现需要考虑网络因素。

因移动端质量相关问题，需要提供热修复等功能。

对移动 App 本身的安全扫描和加固也是一个需要着重考虑的因素。由于前端有不同的实现技术，如果完全使用不同的开发方式，对于企业来说是重复投入，且资源和技术不能共享。因此，使用 Hybrid 混合开发的方式，既可以支持移动 App，又可以支持 H5，甚至小程序，这也是移动中台需要研究的一部分。因此，尽可能将前端组件化，比如 UI 组件和图表组件，在此之上组装成业务组件，能大大提高移动端开发效率和质量。

###### 2.1.2.2.3 移动中台

前面所提的业务中台、数据中台等都是从技术系统层面展开的中台演变。企业在进行中台建设时，容易着手的也是对技术体系的改进。但要发挥中台的能力，让中台战略实际落地到企业，并为企业的业务目标服务，需要有与中台技术架构相匹配的组织架构。从 Supercell 的“部落”，比如阿里巴巴的共享业务事业部、数据平台事业部，京东的前、中、后台，大家都可以看到建设中台需要两手抓，两条路线相匹配，齐头并进。如果将业务中台、数据中台等称为“战斗部队”，那么为企业提供的项目投资管理、风险管理、资源调度等的组织中台则是“战场指挥部”，指挥前线，调度后方。

“大中台，小前台”这种组织形式，并不是什么新鲜事物，实际上它是一种理想化的支撑模式。前台业务足够灵活，配套支撑足够快捷，资源还能够高效复用。不过要让中台模式在企业中发挥作用，对企业本身也是有一定要求的，比如企业有一定规模，业务比较丰富，值得去提炼共性元素形成共享能力。如果同时开展多种相类似的业务，那么从业务 A 提炼出来的能力可能提供给业务 B 使用；或者虽然业务单一，但同一业务在不同地域有不同的模式，也能沉淀出很多共享能力。

数据中台提供了数据分析报表来响应运营，并在此基础上提供数据能力直接服务于业务。那能不能更进一步，提供诸如个性化服务等与智能相关的能力？答案是肯定的，通过 AI 中台就可实现。AI 中台借助数据中台的能力，尝试解决模型的训练、发布，智能服务的构建自动化，统一的元数据管理体系，模型的全生命周期管理等问题，通过 AI 能力平台化，降低对人员能力的要求。与数据中台利用 CPU 级别的资源不同，AI 中台需要扩展对 GPU 资源管理和整合能力，为算法模型的开发者、训练者、标注管理者、数据管理者等构建智能服务的人员服务，并最终为业务人员提供智能化的服务。

##### 2.1.2.3 中台的进化策略

虽然中台概念的提出到现在仅几年时间，但中台已经在这几年中走出了自己的路径。根据中台的进化和演变的历史及可能的方向，目前可以看到共有广度和深度两种途径。

广度是指中台所涉及的内容会越来越多，即可以认为各种中台的不断出现，也可以认为是一个中台内部的共享服务中心会不断横向扩展，从一开始所提的业务中台、数据中台，逐步演化到 AI 中台、技术中台、研发中台等。另一方面，一个中台范围内的共享能力也在扩展，从用户中心、交易中心、营销中心等扩展到内容中心、工单中心、成长中心等。中台团队如发现某一前台业务模式很好，则将其沉淀为共享服务，从而提供更多的业务，这也是在建设和加强中台。由于中台作为中枢点同时支撑多个前台业务，因此中台成为打通前台业务的最好着力点，让不同的前台业务可以互相借力和引流，互相促进发展。

中台所沉淀的共享服务能力并不要求支撑所有前台业务，只要有多于一个前台业务需要某一种能力，此能力即可沉淀为中台能力，因此我们不能大而全地建设中台。如果企业认为现在企业各系统的用户管理能力需要统一，那就可以着手进行用户中心的建设。在此基础上，如果企业发现会员需要统一管理，订单需要全局视图，那么就构建会员中心和订单中心。因此，中台的建设是可以分阶段逐步实施的，无须将所有重构全部一起推动，而后者既会增加复杂性，又会提高风险，还不能及时得到反馈。

中台的成长离不开前台业务的创新。只有不断进行业务迭代和更新试错，对中台提出新的挑战和沉淀，才会让中台做得更好。另一方面，中台团队也需要有自己的产品化、平台化建设思考，并作为新业务的孵化器。

中台还需要建设成为开放的体系。开放不仅仅是对企业内部开放，也要对企业外部开放。通过中台建设，企业可以将自有的系统变为开放式平台，从而为其他企业充分提供第三方的数据和服务。再者，中台本身通过开放也可以充分利用其他第三方数据和服务。开放可以接口的方式，通过开放 API，开创新的商业机会和应用模式。

中台的开放也意味着中台需要支持个性化需求。通过抽象能沉淀共性的流程、数据模型等。但不同业务总有不同点，这些不同的需求就需要个性化的支撑。中台和前台一般是由不同团队负责的。因此为了提高效率，中台必须留出足够灵活的扩展点，以便不同前台业务根据其需求进行定制化扩展。

中台作为平台，必然需要考虑拆分整体应用形成业务组件，通过业务抽象建模，解决共性的问题，从而更好地为业务服务。对业务问题的抽象程度越高，中台对业务的适配度就越高，需要对具体业务参与度就越低，从而更能发挥中台及中台团队的价值。因为越好的抽象越能发挥业务应用开发的创造性。在考虑拆分的同时，必须设计整体框架和组装策略，即组件间的协作机制。通过协作机制，才能让各业务组件协同实现业务场景以达到业务目标。

中台作为一个平台，其本身的运营也需要数据支撑。比如需要统计和观察中台以 API 形式提供的共享能力，从而了解中台哪些能力被业务引用及引用的频率，所使用的参数模式等；哪些设计的接口能力没有用处等。有了实际的数据，可更好地迭代中台。

中台建设是一个综合性的系统工程，因此还需要有效的方法论的指导。