前面我们重点讲述了领域建模、微服务设计和前端设计方法，它们组合在一起就可以形成中台建设的整体解决方案。而中台大多基于分布式微服务架构，这种企业级的数字化转型有很多地方值得我们关注和思考。

我们不仅要关注企业商业模式、业务边界以及前中台的融合，还要关注数据技术体系、微服务设计、多活等多领域的设计和协同。结合实施经验和思考，今天我们就来聊聊分布式架构下的几个关键问题。

一、选择什么样的分布式数据库？

分布式架构下的数据应用场景远比集中式架构复杂，会产生很多数据相关的问题。谈到数据，首先就是要选择合适的分布式数据库。

分布式数据库大多采用数据多副本的方式，实现数据访问的高性能、多活和容灾。目前主要有三种不同的分布式数据库解决方案。它们的主要差异是数据多副本的处理方式和数据库中间件。

1. 一体化分布式数据库方案

它支持数据多副本、高可用。多采用 Paxos 协议，一次写入多数据副本，多数副本写入成功即算成功。代表产品是 OceanBase 和高斯数据库。

2. 集中式数据库 + 数据库中间件方案

它是集中式数据库与数据库中间件结合的方案，通过数据库中间件实现数据路由和全局数据管理。数据库中间件和数据库独立部署，采用数据库自身的同步机制实现主副本数据的一致性。集中式数据库主要有 MySQL 和 PostgreSQL

数据库，基于这两种数据库衍生出了很多的解决方案，比如开源数据库中间件 MyCat+MySQL 方案，TBase（基于 PostgreSQL，但做了比较大的封装和改动）等方案。

3. **集中式数据库 + 分库类库方案**

它是一种轻量级的数据库中间件方案，分库类库实际上是一个基础 JAR 包，与应用软件部署在一起，实现数据路由和数据归集。它适合比较简单的读写交易场景，在强一致性和聚合分析查询方面相对较弱。典型分库基础组件有 ShardingSphere。

小结：这三种方案实施成本不一样，业务支持能力差异也比较大。一体化分布式数据库主要由互联网大厂开发，具有超强的数据处理能力，大多需要云计算底座，实施成本和技术能力要求比较高。集中式数据库 + 数据库中间件方案，实施成本和技术能力要求适中，可满足中大型企业业务要求。第三种分库类库的方案可处理简单的业务场景，成本和技能要求相对较低。在选择数据库的时候，我们要考虑自身能力、成本以及业务需要，从而选择合适的方案。

二、**如何设计数据库分库主键**？

选择了分布式数据库，第二步就要考虑数据分库，这时分库主键的设计就很关键了。

与客户接触的关键业务，我建议你以客户 ID 作为分库主键。这样可以确保同一个客户的数据分布在同一个数据单元内，避免出现跨数据单元的频繁数据访问。跨数据中心的频繁服务调用或跨数据单元的查询，会对系统性能造成致命

的影响。

将客户的所有数据放在同一个数据单元，对客户来说也更容易提供客户一致性服务。而对企业来说，“以客户为中心”的业务能力，首先就要做到数据上的“以客户为中心”。

当然，你也可以根据业务需要用其它的业务属性作为分库主键，比如机构、用户等。

三、数据库的数据同步和复制

在微服务架构中，数据被进一步分割。为了实现数据的整合，数据库之间批量数据同步与复制是必不可少的。数据同步与复制主要用于数据库之间的数据同步，实现业务数据迁移、数据备份、不同渠道核心业务数据向数据平台或数据中台的数据复制、以及不同主题数据的整合等。

传统的数据传输方式有 ETL 工具和定时提数程序，但数据在时效性方面存在短板。分布式架构一般采用基于数据库逻辑日志增量数据捕获（CDC）技术，它可以实现准实时的数据复制和传输，实现数据处理与应用逻辑解耦，使用起来更加简单便捷。

现在主流的 PostgreSQL 和 MySQL 数据库外围，有很多数据库日志捕获技术组件。CDC 也可以用在领域事件驱动设计中，作为领域事件增量数据的获取技术。

四、跨库关联查询如何处理？

跨库关联查询是分布式数据库的一个短板，会影响查询性能。在领域建模时，很多实体会分散到不同的微服务中，但很多时候会因为业务需求，它们之间需要关联查询。

关联查询的业务场景包括两类：第一类是基于某一维度或某一主题域的数据查询，比如基于客户全业务视图的数据查询，这种查询会跨多个业务线的微服务；第二类是表与表之间的关联查询，比如机构表与业务表的联表查询，但机构表和业务表分散在不同的微服务。

如何解决这两类关联查询呢？

对于第一类场景，由于数据分散在不同微服务里，我们无法跨多个微服务来统计这些数据。你可以建立面向主题的分布式数据库，它的数据来源于不同业务的微服务。采用数据库日志捕获技术，从各业务端微服务将数据准实时汇集到主题数据库。在数据汇集时，提前做好数据关联（如将多表数据合并为一个宽表）或者建立数据模型。面向主题数据库建设查询微服务。这样一次查询你就可以获取客户所有维度的业务数据了。你还可以根据主题或场景设计合适的分库主键，提高查询效率。

对于第二类场景，对于不在同一个数据库的表与表之间的关联查询场景，你可以采用小表广播，在业务库中增加一张冗余的代码副表。当主表数据发生变化时，你可以通过消息发布和订阅的领域事件驱动模式，异步刷新所有副表数据。这样既可以解决表与表的关联查询，还可以提高数据的查询效率。

五、如何处理高频热点数据？

对于高频热点数据，比如商品、机构等代码类数据，它们同时面向多个应用，要有很高的并发响应能力。它们会给数据库带来巨大的访问压力，影响系统的性能。

常见的做法是将这些高频热点数据，从数据库加载到如 Redis 等缓存中，通过缓存提供数据访问服务。这样既可以降低数据库的压力，还可以提高数据的访问性能。

另外，对需要模糊查询的高频数据，你也可以选用 ElasticSearch 等搜索引擎。

缓存就像调味料一样，投入小、见效快，用户体验提升快。

六、**前后序业务数据的处理**

在微服务设计时你会经常发现，某些数据需要关联前序微服务的数据。比如：在保险业务中，投保微服务生成投保单后，保单会关联前序投保单数据等。在电商业务中，货物运输单会关联前序订单数据。由于关联的数据分散在业务的前序微服务中，你无法通过不同微服务的数据库来给它们建立数据关联。

如何解决这种前后序的实体关联呢？

一般来说，前后序的数据都跟领域事件有关。你可以通过领域事件处理机制，按需将前序数据通过领域事件实体，传输并冗余到当前的微服务数据库中。

你可以将前序数据设计为实体或者值对象，并被当前实体引用。在设计时你需

要关注以下内容：如果前序数据在当前微服务只可整体修改，并且不会对它做查询和统计分析，你可以将它设计为值对象；当前序数据是多条，并且需要做查询和统计分析，你可以将它设计为实体。

这样，你可以在货物运输微服务，一次获取前序订单的清单数据和货物运输单数据，将所有数据一次反馈给前端应用，降低跨微服务的调用。如果前序数据被设计为实体，你还可以将前序数据作为查询条件，在本地微服务完成多维度的综合数据查询。只有必要时才从前序微服务，获取前序实体的明细数据。这样，既可以保证数据的完整性，还可以降低微服务的依赖，减少跨微服务调用，提升系统性能。

七、数据中台与企业级数据集成

分布式微服务架构虽然提升了应用弹性和高可用能力，但原来集中的数据会随着微服务拆分而形成很多数据孤岛，增加数据集成和企业级数据使用的难度。你可以通过数据中台来实现数据融合，解决分布式架构下的数据应用和集成问题。

你可以分三步来建设数据中台。

第一，按照统一数据标准，完成不同微服务和渠道业务数据的汇集和存储，解决数据孤岛和初级数据共享的问题。

第二，建立主题数据模型，按照不同主题和场景对数据进行加工处理，建立面向不同主题的数据视图，比如客户统一视图、代理人视图和渠道视图等。

第三，建立业务需求驱动的数据体系，支持业务和商业模式创新。 数据中台不仅限于分析场景，也适用于交易型场景。你可以建立在数据仓库和数据平台上，将数据平台化之后提供给前台业务使用，为交易场景提供支持。

八、**BFF 与企业级业务编排和协同**

企业级业务流程往往是多个微服务一起协作完成的，每个单一职责的微服务就像积木块，它们只完成自己特定的功能。那如何组织这些微服务，完成企业级业务编排和协同呢？

你可以在微服务和前端应用之间，增加一层 BFF 微服务（Backend for Frontends）。BFF 主要职责是处理微服务之间的服务组合和编排，微服务内的应用服务也是处理服务的组合和编排，那这二者有什么差异呢？

BFF 位于中台微服务之上，主要职责是微服务之间的服务协调；应用服务主要处理微服务内的服务组合和编排。在设计时我们应尽可能地将可复用的服务能力往下层沉淀，在实现能力复用的同时，还可以避免跨中心的服务调用。

BFF 像齿轮一样，来适配前端应用与微服务之间的步调。它通过 Façade 服务适配不同的前端，通过服务组合和编排，组织和协调微服务。BFF 微服务可根据需求和流程变化，与前端应用版本协同发布，避免中台微服务为适配前端需求的变化，而频繁地修改和发布版本，从而保证微服务核心领域逻辑的稳定。

如果你的 BFF 做得足够强大，它就是一个集成了不同中台微服务能力、面向多渠道应用的业务能力平台。

九、分布式事务还是事件驱动机制？

分布式架构下，原来单体的内部调用，会变成分布式调用。如果一个操作涉及多个微服务的数据修改，就会产生数据一致性的问题。数据一致性有强一致性和最终一致性两种，它们实现方案不一样，实施代价也不一样。

对于实时性要求高的强一致性业务场景，你可以采用分布式事务，但分布式事务有性能代价，在设计时我们需平衡考虑业务拆分、数据一致性、性能和实现的复杂度，尽量避免分布式事务的产生。

领域事件驱动的异步方式是分布式架构常用的设计方法，它可以解决非实时场景的数据最终一致性问题。基于消息中间件的领域事件发布和订阅，可以很好地解耦微服务。通过削峰填谷，可以减轻数据库实时访问压力，提高业务吞吐量和处理能力。你还可以通过事件驱动实现读写分离，提高数据库访问性能。对最终一致性的场景，我建议你采用领域事件驱动的设计方法。

十、多中心多活的设计

分布式架构的高可用主要通过多活设计来实现，多中心多活是一个非常复杂的工程，下面我主要列出以下几个关键的设计。

1. 选择合适的分布式数据库。数据库应该支持多数据中心部署，满足数据多副本以及数据底层复制和同步技术要求，以及数据恢复的时效性要求。

2. 单元化架构设计。将若干个应用组成的业务单元作为部署的基本单位，实现

同城和异地多活部署，以及跨中心弹性扩容。各单元业务功能自包含，所有业务流程都可在本单元完成；任意单元的数据在多个数据中心有副本，不会因故障而造成数据丢失；任何单元故障不影响其它同类单元的正常运行。单元化设计时我们要尽量避免跨数据中心和单元的调用。

3. 访问路由。访问路由包括接入层、应用层和数据层的路由，确保前端访问能够按照路由准确到达数据中心和业务单元，准确写入或获取业务数据所在的数据库。

4. 全局配置数据管理。实现各数据中心全局配置数据的统一管理，每个数据中心全局配置数据实时同步，保证数据的一致性。