

同城双活应用参考实现指南

|  |  |  |  |
| --- | --- | --- | --- |
| **作者** | 汤闻达 | **审核人** |  |
| **最终修改日期** | 2020-12-17 | **最终审核日期** |  |
| **版本号** | 3.0 | **最终发布日期** |  |

中汇信息技术（上海）有限公司

变更履历

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 文档版本号 | 变更时间 | 变更范围及简要说明 | 变更类别 | 变更人 |
| 1.0 | 2020.11.16 | 全文 | N | 汤闻达 |
| 2.0 | 2020.12.6 | 全文  根据申冬东老师意见，修订全文  将应用重点关注的内容和非重点关注的内容分开描述。  针对应用(不仅仅只是交易类应用，综合考虑交易中心各类应用)重点关注的内容，提供几种同城双活的参考架构(应该不只是拆分双活和对称双活，如一些基于数据库的简单应用，可能把数据库换成分布式数据库就可以支持同城双活，无需业务单元划分等步骤)，分别确定其中的架构要点、关键技术选型、适用场景、应用举例、应用关注事项等内容，后续各系统同城双活改造直接在这几种模式中选择即可。  应用无需重点关注的内容可另外分区章节说明，如仲裁内部实现等。 | M | 汤闻达 |
| 3.0 | 2020.12.17 | 根据申冬东老师意见，增加若干应用双活参考双活架构 | A | 汤闻达 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

注：N-New 新建；M-Modify 修改；A-Add 新增；D-Delete 删除；

# 背景

## 应用双活特征

应用双活是指从应用层实现跨数据中心的业务双活，每个数据中心均可对外提供业务，并在一个机房发生灾难时，其他机房可接管所有业务。

同城双活要求最少在同城具备2个机房，在需要第三方仲裁的场景，则需具备3个机房。在具备非常高要求的业务连续性保障场景中，如出现重大、灾难性机房级故障事件，同城双活系统能较快恢复继续服务的能力。

双活建设涉及业务处理全流程的多个环节，不同层次的应用双活改造涉及不同规模的改造成本，同时也是应用高水平双活建设面临的挑战之一。

## 应用双活部署模式

目前双活架构模式主要有以下三大类，满足不同应用的业务场景需求。

### 非对称双活



非对称双活架构中各数据中心的应用构成跨中心集群响应客户请求，业务数据统一向主数据中心数据库执行写操作。数据部分是切换式机房接管，双机房的应用部分均处于激活状态，与传统跨机房主备模式相比，故障切换路径的更短，因为缺少了应用部分的切换，故障切换速度完全取决于数据库的切换时间。

该架构中只有数据资源可复用，可复用度较小。其更强调紧急切换管理流程并保障切换可靠，同时，两站点都需要生产运行管理能力。从双活实现的难易程度上看，这种模式可以将无状态的应用快速部署为同城双活应用。

由于非对称双活架构中，所有数据的CURD操作均是在源数据中心机房中进行(具备读写分离操作性的业务场景可以进行读优化)，且数据库的数据备份是单向的，即从源数据中心备份数据到同城数据中心。因此，该架构对应用的要求较小，通常将应用的数据访问路径选择同一个数据源即可帮助应用获得低投入成本的双活能力。

由于用户可以从不同的机房进行接入，但是可用数据只有一份，所以对于数据强一致要求高的业务，远离数据源的应用针对数据的读取均需要进行跨机房的数据访问，这对应用的性能具有一定的影响，对业务访问时延敏感的应用不太友好，因为在统计意义上看，不同机房的接入用户通常具有不平等的服务响应时间。

而对于数据强一致要求不高的业务，可根据业务需求进行读写分离。双机房场景下还是一个主库，只是在另外一个机房多出了另外的一些从库，并且写入是直接跨机房连接到主库写入。

有如下几个使用读写分离的场景：

1、读写量很大，为了提升数据库读写性能，将读写进行分离；

2、多机房下如果写少读多，同时基于数据一致性考虑，只有一个主库存入所有的数据写入，本地再做从库提供读取，减少多机房间直接读取带来的时延。

由于远离数据源的应用接入在数据写入操作时不得不接受跨机房距离所带来的客观网络延迟，因此，该架构要求应用能够容忍业务频繁的跨机房延时。

### 拆分双活

拆分双活架构要求每个机房按一定的拆分规则独立承担业务。具体表现在，各数据中心的应用及数据库集群可独立运行, 业务数据按照数据中心进行拆分，各拆分部分的数据写操作只在源数据中心的数据库中执行。

从部署上看，单个机房用于部署部分应用与数据，对外提供特定类型的服务或特定用户的服务，其余部分应用和数据则在另一机房处于热备状态；另一机房反之部署。



### 对称双活

各个机房的应用及数据库均为主用，均可由单一数据中心完成完整的业务事务。根据是否需要进行应用层面和数据持久化层面的实时同步，主要细分有数据一致无关、状态无关、交易一致和最终一致等业务场景。



结合业务系统的需求，双活部署模式的选择主要需要从应用双活改造复杂度、改造前后的性能差异、改造前后的业务连续性能力差异等角度综合分析。

下表对上述双活部署模式进行了简要对比分析：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 简要描述 | 数据运维复杂度 | 故障恢复能力 | 应用改造成本 | 应用性能 |
| 非对称双活 | 各数据中心的应用构成跨中心集群响应客户请求，业务数据统一向主数据中心数据库执行写操作。 | 低 | 低 | 低 | 中 |
| 拆分双活 | 各数据中心应用及数据库集群独立运行, 业务按照数据中心进行拆分，写操作只在源数据中心数据库执行。 | 高 | 高 | 高 | 高 |
| 对称双活 | 各个数据中心应用及数据库均为主用，单一数据中心完成完整的业务事务。 | 中 | 中 | 中 | 中（强一致）  高（弱一致） |

三种部署模式在数据运维复杂度、故障恢复能力、应用改造成本投入和应用双活改造后的性能等方面具有一定的区分度，具体细节将在下文进行阐述。

### 交易中心业务适用双活参考

根据《中国外汇交易中心产品指引》的有关说明，银行间市场按照交易的驱动因素，将交易模式分为报价驱动（QDM，Quote Driven Model）、订单驱动（ODM，Order Driven Model）和协商驱动（NDM，Negotiate Driven Model）。其中，QDM 指做市商向市场持续公开报价，对手方点击价格后双方达成交易；ODM 指交易双方提交订单或点击报价，系统按“价格优先、时间优先”原则进行匹配达成交易；NDM 指交易双方通过协商交易要素，一方录入、一方确认达成交易。

银行间市场的交易模式对应3种交易方式，分别是询价、请求报价和点击成交。询价是指交易双方通过电子交易系统自行协商价格、成交量等交易要素，经历意向报价、对话报价，再确认成交。请求报价指有交易意向的机构向一个或多个做市商发出只含量、不含价的报价请求，再由这些做市商进行回价，发出交易请求的机构可接受或拒绝该价格。点击成交是指做市商报出分组、带量可成交价，满足授信要求的对手方通过点击报价或提交限价订单的方式与做市商达成交易。

虽然以询价和请求报价为代表的NDM模式的交易环节冗长，但该模式仍是目前银行间债券市场上的**主流交易模式**。

下表对三种交易模式和其有关特征进行了概述。

|  |  |  |  |
| --- | --- | --- | --- |
| 交易模式 | QDM | ODM | NDM |
| 成交过程描述 | 做市商根据市场行情和自有资金、债券的情况，提供双边报价。其他投资者均以做市商为交易对手方，做市商之间也可进行交易 | 本质上属于竞价交易，交易双方是匿名的，参与者提交市价单或限价单完成交易，系统根据“价格优先、时间优先”的原则进行匹配，成交价格由交易双方的指令决定 | 报价邀请发出后不会立即成交，一方对对手方返回的报价确认后即可成交，事前难以知晓交易价格 |
| 特征 | 非做市商之间不直接进行交易 | 买卖双方直接成交，不存在做市商作为流动性的提供者 | 需要进行多轮对话，无法立即成交 |
| 涵盖交易方式 | 点击成交，匿名点击 | 市价成交，限价成交 | 请求报价，询价 |
| 代表性市场 | 银行间债券市场，上交所固定收益平台 | 交易所债券市场等 | 银行间债券市场等 |

从技术实现上看，不同的交易模式在大体上可规约为撮合类业务和非撮合类业务两大类，撮合类业务主要体现形式为ODM类交易，其具备业务响应时延敏感，业务连续性要求高等特征。非撮合类交易主要体现为NDM类交易，其业务响应时延相较ODM类交易不敏感。从交易中心业务的体量上来看，撮合类业务在本币市场约占20%，外汇市场约50%。剩下的数十个外围系统，其大部分属于非撮合类业务，主要由查询信息类，如LPR发布系统，人民币对美元中间价发布等组成。

由前文所述的双活架构模式特征上看，撮合类应用的双活部署模式的可选范围不多，非撮合类则同时具备三种部署模式的选择。三种部署模式的不同选择主要在于用户对业务性能的追求和业务连续性恢复能力的侧重上，除此此外，还有双活应用的改造成本上的考虑。

以本币系统为例，其NDM订单的交易延迟容忍较高（客户端界面发起：300-500ms，API：200-300ms）,因此可做到上述三种双活部署模式。而ODM订单的交易延迟容忍较低（客户端界面发起：200ms，API：8ms），则宜采用跨数据中心的拆分双活部署，从而可减少跨数据中心的频繁网络交互，避免跨数据中心的网络延迟影响交易服务的响应时间。

对于大部分非撮合类业务，如前文所述的信息查询类布系统等可完全依托于数据层面的跨机房容灾能力，实现同城非对称双活能力。

本文主要以本币同业拆借为业务模型提供NDM交易的双中心的拆分双活、对称双活参考实现，以外汇C-Trade为业务模型的ODM双中心的拆分双活参考实现。

# 双活应用关键技术

## 非对称双活

### 数据主从复制

非对称双活中，主数据源只有一处，数据库服务器通常搭建主从集群，一主一从、一主多从等。数据库主机负责读写操作，从机只负责读操作。数据库主机通过复制将数据同步到从机，每台数据库服务器都存储了所有的业务数据。业务服务器将写操作发给数据库主机，将读操作发给数据库从机。需要注意的是，这里用的是“主从集群”，而不是“主备集群”。“从机”是需要提供读数据的功能的；而“备机”一般被认为仅仅提供备份功能，不提供访问功能。

主从复制方案详细解释如下：

（1）主机存储数据，通过复制通道将数据复制到从机。

（2）正常情况下，客户端写操作发给主机，读操作可发送给主机也可以发送给从机，具体如何选择，可以根据业务的特点选择。可以随机读，可以轮询读，可以只读主机，等等。

（3）主机故障的情况下（例如，主机宕机），客户端无法进行写操作，但可以将读操作发送给从机，从机继续响应读操作，此时和写操作相关的业务不可用，但和读操作相关的操作不受影响。

（4）如果主机能够恢复（人工/自动），客户端可继续将写操作请求发送给主机，主机继续将数据复制给从机。

（5）若主机不能恢复，则需要人工操作，将从机升为主机，然后将客户端访问新的主机（即原来的从机）；同时为了继续保持主从架构，需人工增加新的机器作为从机。

（6）主机不能恢复的情况下，成功写入了主机但还没有复制到从机的数据会丢失，需要人工进行排查和恢复，也许有的数据就完全丢失了，业务上需要考虑如何应对此类风险。

（7）如果主从间数据复制延迟，则会出现主从读取的数据不一致问题。例如，用户刚发了一个报价，此时数据还没有从主机复制到从机，用户的读操作被发送到了从机，该读操作就无法获得最新的写入数据。

（8）如果主从间延时较大，恰好此时主机又宕机，则可能丢失较多数据，因此对于复制延迟需要提供相应的监控措施，当延迟的数据量较大及时告警，由人工干预处理。

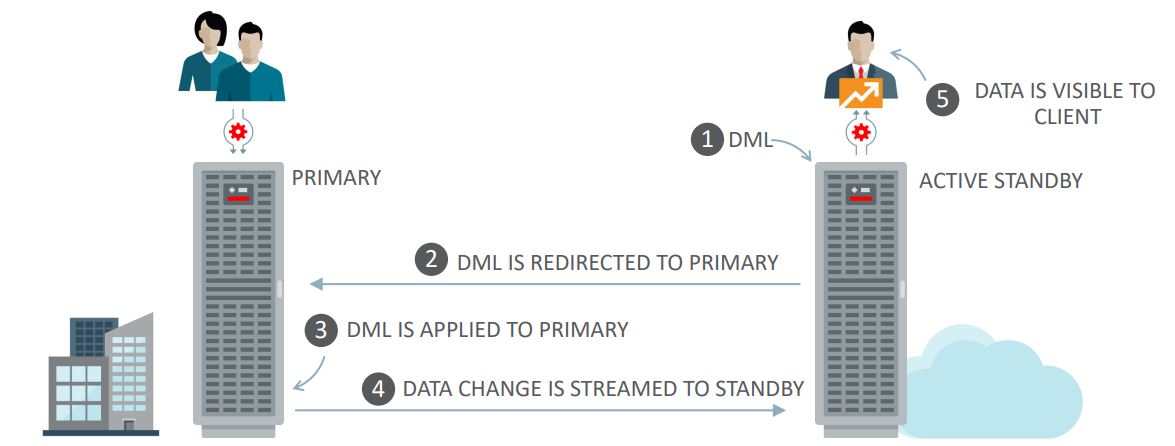
具体到主从复制技术的实现层面，Oracle数据库产品的Active DataGuard提供三种数据保护模式：最大保护（Maximum Protection），最大可用（Maximum Availability）和最大性能（Maximum Performance）。如果按照对数据的保护程度或者说主从库数据的同步性 由低到高排序，三种保护模式的顺序应该是：最高性能、最高可用、最大保护 。

最大性能(maximize performance):这是data guard默认的保护模式。primay上的事务commit前不需要从standby上收到反馈信息，该模式在primary故障时可能丢失数据，但standby对primary的性能影响最小。

最大可用(maximize availability):在正常情况下，最大可用模式和最大保护模式一样；在standby不可用时，最大可用模式会自动降低成最大性能模式，所以standby故障不会导致primay不可用。只要至少有一个standby可用的情况下，即使primary down机，也能保证不丢失数据。

最大保护(maximize protection):最高级别的保护模式。primay上的事务在commit前必须确认redo已经传递到至少一个standby上，如果所有standby不可用，则primary会挂起。该模式能保证零数据丢失。

Oracle 19c 增加了新特性：ADG的自动DML（Data Manipulation Language）重定向增强读写分离。可将偶然发送到ADG上的DML操作，自动转发到主库执行，然后通过主库日志传递到备库实时应用，在保证了ACID的前提下，大大增强了备库的实用性，这被称为 DML Redirection 。



Oracle 19c Active Data Guard-DML Redirection

### 应用读写分离实现

读写分离在应用的代码实现逻辑上并不复杂，有两个细节点引入了设计复杂度：主从复制延迟和分配机制。

（1）复制延迟

以对应的 MySQL数据层为例，主从复制延迟可能达到 1 秒，如果有大量数据同步，延迟更长也是有可能的。主从复制这种客观延迟会带来一个问题：如果业务服务器将数据写入到数据库主服务器后立刻（1 秒内）进行读取，此时读操作访问的是从机，主机还没有将数据复制过来，到从机读取数据是读不到最新数据的，业务上就可能出现不一致问题。

解决主从复制延迟有几种常见的方法：

* 写操作后的读操作指定发给数据库主服务器

例如，注册账号完成后，登录时读取账号的读操作也发给数据库主服务器。这种方式和业务强绑定，对业务的侵入和影响较大。

* 读从机失败后再读一次主机

顾名思义，即“二次读取”，二次读取和业务无绑定，只需要对底层数据库访问的 API 进行封装即可，实现代价较小，不足之处在于如果有很多二次读取，将大大增加主机的读操作压力。

* 关键业务读写操作全部指向主机，非关键业务采用读写分离

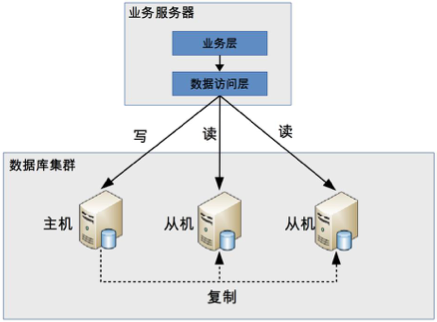
例如，对于一个用户管理系统来说，注册 + 登录的业务读写操作全部访问主机，用户的介绍、等级等业务，可以采用读写分离，因为即使用户改了自己的自我介绍，在查询时却看到了自我介绍还是旧的，业务影响与不能登录相比就小很多，还可以忍受。

（2）分配机制

将读写操作区分开来，然后访问不同的数据库服务器，一般有两种方式：程序代码封装和中间件封装。

* 程序代码封装

程序代码封装指在代码中抽象一个数据访问层（所以有的文章也称这种方式为“中间层封装”），实现读写操作分离和数据库服务器连接的管理。例如，基于 Hibernate 进行简单封装，就可以实现读写分离，基本架构是：

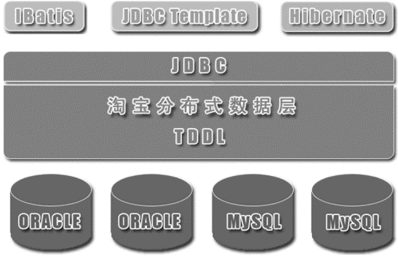


程序代码封装的方式具备几个特点：

实现简单，而且可以根据业务做较多定制化的功能。每个编程语言都需要自己实现一次，无法通用，如果一个业务包含多个编程语言写的多个子系统，则重复开发的工作量比较大。

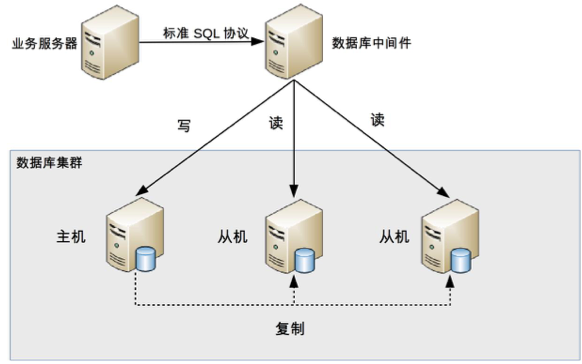
故障情况下，如果主从发生切换，则可能需要所有系统都修改配置并重启。

目前开源的实现方案中，淘宝的 TDDL（Taobao Distributed Data Layer）是一个通用数据访问层，所有功能封装在 jar 包中提供给业务代码调用。其基本原理是一个基于集中式配置的 jdbc datasource 实现，具有主备、读写分离、动态数据库配置等功能，基本架构是：



* 中间件封装

中间件封装指的是独立一套系统出来，实现读写操作分离和数据库服务器连接的管理。中间件对业务服务器提供 SQL 兼容的协议，业务服务器无须自己进行读写分离。对于业务服务器来说，访问中间件和访问数据库没有区别，事实上在业务服务器看来，中间件就是一个数据库服务器。其基本架构是：



数据库中间件的方式具备的特点是：

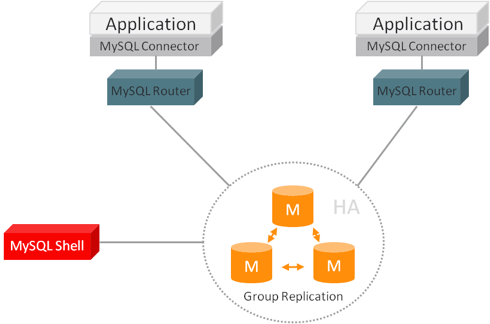
能够支持多种编程语言，因为数据库中间件对业务服务器提供的是标准 SQL 接口。

数据库中间件要支持完整的 SQL 语法和数据库服务器的协议（例如，MySQL 客户端和服务器的连接协议），实现比较复杂，细节特别多，很容易出现 bug，需要较长的时间才能稳定。

数据库中间件自己不执行真正的读写操作，但所有的数据库操作请求都要经过中间件，中间件的性能要求也很高。

数据库主从切换对业务服务器无感知，数据库中间件可以探测数据库服务器的主从状态。例如，向某个测试表写入一条数据，成功的就是主机，失败的就是从机。

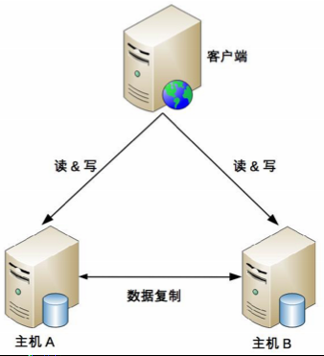
目前的开源数据库中间件方案中，MySQL 官方先是提供了 MySQL Proxy，但 MySQL Proxy 一直没有正式 GA，现在 MySQL 官方推荐 MySQL Router。MySQL Router 的主要功能有读写分离、故障自动切换、负载均衡、连接池等，其基本架构如下：



本文第四部分的数据层服务将对数据库中间件与分布式数据库进行对比分析和阐述。

## 对称双活

与非对称双活架构采用的数据主从复制不同，对称双活架构采用数据的主主复制。主主复制指的是跨机房两数据库服务器都是主机，互相将数据复制给对方，处于跨机房部署的双活客户端可以任意挑选（可优先同机房）其中一台机器进行读写操作，下面是基本架构图。



主主复制架构的详细解释如下：

（1）两台主机都存储数据，通过复制通道将数据复制到另外一台主机。

（2）正常情况下，客户端可以将读写操作发送给任意一台主机。

（3）一台主机故障情况下，例如主机A宕机，客户端只需要将读写操作发送给主机B即可，反之亦然。

（4）如果故障的主机A能够恢复（人工/自动），则客户端继续访问两个主机，两台主机间继续互相复制对方数据。

（5）如果故障的主机A不能恢复（例如硬盘损坏等短期内无法恢复场景），则需要人工操作，增加一台机器作为主机。

（6）原有故障主机A不能恢复的情况下，成功写入了原有故障主机但还没有复制到正常主机B的数据会丢失，需要人工进行排查和恢复，和异步主从复制情况类似，也许有的数据就完全丢失了，业务上需要考虑如何应对该类风险。

（7）如果两台主机间复制延迟，则可能出现客户端刚写入了数据到主机A，然后主机B去读取，此时读不到刚刚写入的数据。

相比主从切换架构，主主复制架构具有如下特点：

两边都是主机，不存在切换的概念。客户端无须区分不同角色的主机，随便将读写操作发送给哪台主机都可以。

从上面的描述来看，主主复制架构从总体上来看要简单很多，无须状态信息传递，也无须状态决策和状态切换。然而事实上主主复制架构也并不简单，而是有其独特的复杂性，具体表现在：如果采取主主复制架构，必须保证数据能够双向复制，而很多数据是不能双向复制的。例如：用户注册后生成的用户 ID，如果按照数字增长，那就不能双向复制，否则就会出现 X 用户在主机 A 注册，分配的用户 ID 是 100，同时 Y 用户在主机 B 注册，分配的用户 ID 也是 100，这就出现了冲突。

交易额度涉及的额度扣减等场景不能双向复制。例如，用户交易额度为100，主机 A 上减了 1变成 99，主机 B 上减了 2变成 98，然后主机 A 将额度扣减备 99 复制到主机 B，主机 B 原有的剩余额度 98 被覆盖，变成了 99，而实际上此时真正的剩余额度是 97。

因此，主主复制架构对数据的设计有严格的要求，一般适合于那些临时性、可丢失、可覆盖的数据场景。例如，用户登录产生的 session 数据（可以重新登录生成）、用户行为的日志数据（可以丢失）等。

具体到主从复制的技术实现层面，Oracle GoldenGate支持两种双向复制配置，一种是active-passive模式，这种模式一般被用来实现live standby，做主备库，还有一种是active-active模式，这种模式被用来实现HA高可用、分摊负载压力。Active-Active模式通常用来解决一些分摊负载提高高可用性的需求，这种配置在两个系统上拥有相同的数据集，并且应用用户可以在任意一台机器上进行数据更改。Oracle GoldenGate复制每一台的事务数据变化到其他的节点来保持所有的数据集一致。

在这种双向配置中，在每一台系统上都有一个完整的激活的Oracle GoldenGate进程组集合。在一台系统上由Extract进程捕获的数据会传播到其他系统上用作Replicate的复制应用。

## 拆分双活

与非对称双活和对称双活不同，拆分双活架构尤其适用于对跨机房服务调用时延敏感的应用，该类应用既想获得双活应用的业务连续性提升，亦不愿丢失原本的性能。主要涉及有关应用层面的业务单元化改造工作。

### 业务单元化

现阶段交易中心上海的同城双机房之间的网络物理链路 RTT 或会存在 2ms 的数据传输时延，在某些对性能具有极致要求的业务场景下，如果一次业务请求在后台系统会发生数十次甚至上百次跨机房交互，那全链路的业务响应时间势必无法满足用户需求。

如前文所述，跨数据中心服务调用与数据同步存在客观不容忽视的时延与网络抖动，尽管该部分由基础设施层提供平台技术支撑与质量保证，但其优化指标存在一定的瓶颈，应用的双活设计仍需要结合业务对服务响应时延的敏感程度等因素来确定应用部署的细节，以尽可能降低跨数据中心的服务调用频率，从而降低业务请求的全链路响应时间，保证用户的业务体验。

目前，业界的理想解决方式就是采用单元化改造的思想对业务系统进行划分。具体来说，单元化是指通过对应用层和数据层按照相同的数据分片维度进行业务流量的划分，从而把一整套的业务请求链路收敛在一个数据中心内部，即一个封闭的“业务处理单元”。这个单元可以作为一个相对独立的整体来挪动，甚至可以把部分单元部署到异地去，从而实现理想化的“异地多活”。事实上，单元化的核心思想就是单元内服务高内聚，单元间服务低耦合，考虑到跨单元的服务调用在业务流程上可能无法回避，因此，应该通过业务系统改造或调整数据分片维度、粒度等，以尽量减少单元间的服务调用。

具体业务单元的划分方案离不开业务数据的梳理工作，找到合适的维度进行数据切分，实现各个数据分片能够独立进行业务处理。

### 业务单元上/下线保护

业务单元保护机制对业务的请求响应进行保护和纠错，进而保证在实现数据的零脏写的前提下，尽最大可能提升服务请求的成功率。

单元化的业务包括多个进程，若发生进程故障，外围系统的故障等，需要将单元视为一个整体处理。比如管理单元消息入口故障和切换，识别单元内进程故障是否影响单元运行并（自动）接收相应的操作指令进行业务单元的关闭（下线）处理。

可对业务单元中的进程作如下的分类：

（1）业务单元入口进程。

（2）链路关键进程，一旦进程故障则整个单元无法运行。

（3）链路出口进程或通道。

如果业务单元内部发生部分进程故障，导致无法继续正常处理业务服务，那么需要考虑针对该业务单元的下线操作。如果因为如果在处理过程中将通信链路（如业务单元入口进程、链路出口进程）直接“暴力”切断，则容易导致正在处理消息的相关进程无法将正在处理的状态信息传递到业务单元外的进程，容易引起消息丢失。我们把这样的“暴力”切断视为业务单元的“刚性下线”。

无状态的业务单元遭遇“刚性下线”并不会对业务造成应用。考虑有状态的业务单元，“刚性”关闭会引发请求失败，引起数据状态丢失、数据链路截断流程不完整，导致后续业务数据丢失。因此，有状态业务的连续性支持依赖于针对业务单元的下线操作的精细管理。类似的，业务单元的上线操作也需要提供相应的管理操作。

### 数据主备复制

如前所述，拆分双活将原始的应用按一定业务维度拆分，所有的数据写入操作均在机房内完成，同时兼顾跨机房的数据备份，这部分的备份通常采用主备复制的方式进行，与主从复制的不同在于是否在主机工作时提供备机的数据操作。

主备复制和主从复制存在两个共性问题：

（1）主机故障后，无法进行写操作。

（2）如果主机无法恢复，需要人工指定新的主机角色。

主备倒换方案在上述方案的基础上增加“倒换”功能，即由系统自动决定主机角色，并完成角色切换。如要有如下的设计要点：

（1）主备间状态判断

包括两个方面：状态传递的渠道（是相互间互相连接还是第三方仲裁）和状态监测的内容（机器掉电，进程，响应速度等）。

（2）倒换决策

包括倒换时机、倒换策略、自动程度。

倒换时机：什么情况下备机升级为主机，是机器掉电后备机才升级，还是主机上的进程不存在就升级，还是主机响应时间超过2秒就升级，还是3分钟内主机连续重启3次就升级，等等。

倒换策略：原来的主机故障恢复后，要再次倒换，确保原来的主机继续做主机，还是原来的主机故障后自动成为新的备机

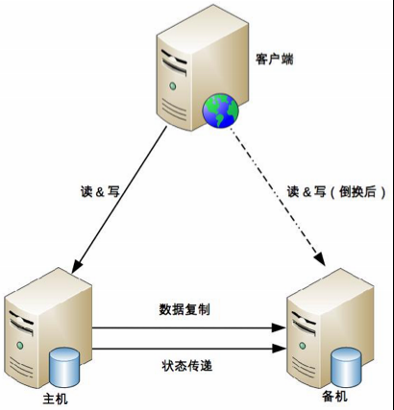
自动程度：倒换是全自动的，还是半自动。例如系统判断当前需要倒换，但需要人工做决策。

（3）数据冲突解决

当原有故障的主机恢复后，新旧主机之间可能存在数据冲突。如何进行取舍，需要在做拆分双活设计时考虑这样的风险。

根据状态传递渠道的不同，常见的主备切换架构有三种形式：互连式、中介式和模拟式。

互连式：指主备机直接建立状态传递的渠道。



通道的具体实现可以有很多方式：可以是网络连接（例如，各开一个端口），亦可以是主机发送状态给备机，也可以是备机到主机来获取状态信息。可以和数据复制通道共用，也可以独立一条通道。状态传递通道可以是一条，也可以是多条，还可以是不同类型的通道混合。

为了充分利用切换方案能够自动决定主机这个优势，客户端这里也会有一些相应的改变，常见的方式有：

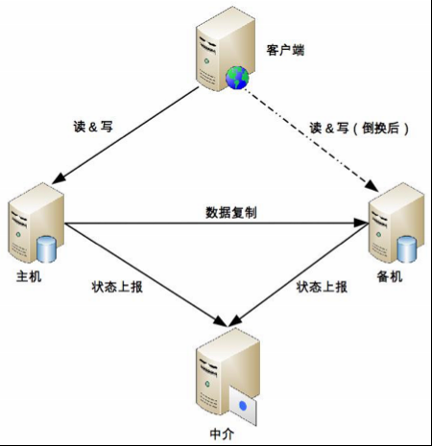
为了切换后不影响客户端的访问，主机和备机之间共享一个对客户端来说唯一的地址。例如虚拟 IP，主机需要绑定这个虚拟的 IP。

客户端同时记录主备机的地址，哪个能访问就访问哪个；备机虽然能收到客户端的操作请求，但是会直接拒绝，拒绝的原因就是“备机不对外提供服务”。

互连式主备切换主要的缺点在于如果状态传递的通道本身有故障（例如，网线掉线），那么备机也会认为主机故障了从而将自己升级为主机，而此时主机并没有故障，最终就可能出现两个主机。

虽然可以通过增加多个通道来增强状态传递的可靠性，但这样做只是降低了通道故障概率，不能从根本上解决这个缺点，且通道越多，状态决策会更复杂，因为对备机来说，可能从不同的通道收到了不同甚至矛盾的状态信息。

中介式指的是在主备两者之外引入第三方中介，主备机之间不直接连接，而都去连接中介，并且通过中介来传递状态信息，其架构图如下：



主机和备机不再通过互联通道传递状态信息，而是都将状态上报给中介这一角色。相比互连式架构，中介式架构在状态传递和决策上却更加简洁。主要有如下以下两点。

连接管理更简单：主备机无须再建立和管理多种类型的状态传递连接通道，只要连接到中介即可，降低了主备机的连接管理复杂度。

状态决策更简单：主备机的状态决策简单了，无须考虑多种类型的连接通道获取的状态信息如何决策的问题，只需要按照下面的算法即可完成状态决策：无论是主机还是备机，初始状态都是备机，并且只要与中介断开连接，就将自己降级为备机，因此可能出现双备机的情况。主机与中介断连后，中介能够立刻告知备机，备机将自己升级为主机。如果是网络中断导致主机与中介断连，主机自己会降级为备机，网络恢复后，旧的主机以新的备机身份向中介上报自己的状态。如果是掉电重启或者进程重启，旧的主机初始状态为备机，与中介恢复连接后，发现已经有主机了，保持自己备机状态不变。

主备机与中介连接都正常的情况下，按照实际的状态决定是否进行切换。例如，主机响应时间超过 3 秒就进行切换，主机降级为备机，备机升级为主机即可。

虽然中介式架构在状态传递和状态决策上更加简单，但该架构也有相应的代价，其关键就在于如何实现中介本身的高可用。如果中介宕机了，整个系统就进入了双备的状态，写操作相关的业务就不可用了。

仲裁节点是一种特殊的节点，它本身不必存储数据，主要的作用就是决定哪一个备节点在主节点宕机后提升为主节点，所以客户端不需要连接此节点。开源方案已经有很成熟的解决途径，如ZooKeeper等，主备倒换的中介可借助于Zookeeper来做状态同步，后文将对此内容进行叙述。

# 应用双活的若干参考架构

## 一般交易流程类双活架构

该架构适合业务的特征主要有：

1）具有时序性的交易业务

2）可分割的交易业务（可按用户、标的等维度）

3）具有一般的实时性要求

4）基础数据变化低

应用举例：NDM、QDM等实时性要求一般的业务场景。

其架构如下图所示：



具体有如下设计要点：

1）交易服务中的本地缓存用于存放基础数据，由于基础数据变化不大，采用延迟加载+时钟触发更新的组合方式进行缓存数据更新。具体来说，对于基础数据同步，采用延迟加载模式，通过应用模块的主动从数据库加载到内存的做法来解决同步问题；当基础模数据变动时，先保存到本次数据库，并对内存中对象失效标志；应用在处理时发现失效内存后，主动从数据库中加载；同时采用二级缓存策略，对已大量基础数据，应为数据量大，数据字段长等原因，使用本地缓存处理。

2）在多活并发量不高的情况下，应用自申请空间作为缓存，直接读取数据库数据进行加载的模式；在并发量较高的情况下，采用独立的缓存，避免直接读取数据库；

3）将常用计算结果和命中频度高的计算结果保留在缓存中，从而提高计算效率；

4）交易流程服务群中单个交易流程服务可以是无状态的，亦可采用一致性哈希算法生成hash值进行交易核心的选择，然后进行交易业务处理，包括新增、修改、撤销、确认等业务处理；交易流程服务也主要由交易核心处理模块和缓存延迟加载模块组成。

5）通过预处理集群，将业务上耗时比较长的事前验证，有效的进行分散化处理；系统通过横行的扩充预处理实例，可以有效提供向交易核心提供的交易数量，最大限度的使用交易核心资源；预处理集群可以根据业务压力情况，追加或减少预处理实例，有效的调节系统的并发服务能力。

6）额度扣减可采用无状态的多活部署，建议机房内双活部署，提升高可用能力。

7）预处理采用集群方式部署，报价状态为无状态消息，集群中任一预处理实例发生故障，剩余的预处理群也可以有效吸收故障服务所承担的处理工作量。多活核心交易进程采用机房间互备份方式，保证任一核心交易服务挂机的情况下，业务分发转移到其他业务群中，继续对外提供稳定服务。

架构可选技术：

*  接入网关：DSP，Nginx
*  缓存技术：redis
*  高可用技术：DSP多数据中心分发技术
*  数据库：分布式数据库，分布式数据库中间件

## 实时交易流程类双活架构

满足系统的高可用、高并发和高性能需求。

应用举例：ODM等实时性要求高的业务场景。

待二期ODM双活应用补充细节。

## 信息查询维护类双活架构

主要用于具备非交易属性的业务系统，该架构适合业务的特征主要有：

1）有时序性的信息查询维护

2）不包含复杂的业务要求

3）变化频度低的业务操作（用户操作、权限操作、基础数据操作等）

应用举例：信息查询系统LPR发布、备案类系统等

其架构如下图所示，具体有如下设计要点：

1）部分数据的时效性要求一般的话，可进行跨机房弱一致提高吞吐量，对于时效性要求高的数据，可进行跨机房强一致处理。

2）依托于数据层提供的一致性，无状态服务可实现横向扩展应对高并发的查询与信息修改。

3）基础类数据容易命中同一份缓存数据，采用主从架构+读写分离方案产生多份缓存副本，将请求分散到多个缓存服务器上，减轻缓存热点导致的单台缓存服务器压力。



架构可选技术：

*  接入网关：DSP，Nginx
*  缓存技术：redis集群
*  高可用技术：DSP多数据中心分发技术
*  数据库：Oracle GoldenGate（跨机房弱一致），分布式数据库，分布式数据库中间件（跨机房强一致）

# 应用双活改造实施步骤与参考实现

结合上述无论是采用拆分双活，还是对称双活部署模式，应用均需要首先进行业务单元化改造，并在业务单元化改造后，根据跨机房数据同步实时性需求，业务响应时延敏感性等方面确定其双活部署模式。

## 业务单元划分

单元化是指通过对应用层和数据层按照相同的数据分片维度进行业务流量的划分，从而把一整套的业务请求链路收敛在一个数据中心内部，形成由应用层到数据层组成一个封闭的“业务处理单元”。这个单元可以作为一个相对独立的整体来挪动，甚至可以把部分单元部署到异地去，从而实现理想化的“异地多活”。

针对业务系统进行业务单元划分，以在一个机房中完成业务功能的全链路，全面减少可能的跨机房业务延迟为目标。具体来说，事实上单元化的核心思想就是单元内服务高内聚，单元间服务低耦合。考虑到跨单元的服务调用在业务流程上可能无法回避，因此，应该通过业务系统改造或调整数据分片维度、粒度等，以尽量减少单元间的服务调用，从而把整个业务流程的耗时控制在用户可接受的范围内。

结合NDM的业务实际，其核心交易服务主要分由预处理，交易核心，交易辅助3个主要服务组成。

其中，预处理完成请求响应消息信息验证工作，然后转发给交易核心计算。并等待交易核心的反馈，然后将反馈信息交由给网关传送至客户端。具体来说，消息路由预处理服务,提供对前台传入消息或API发来的上行消息中的通用交易基础数据的校验。消息通过REST网关或是API网关后，进入到预处理模块,根据不同的业务场景消息完成该场景配置的所有验证规则的校验,所有规则验证通过时,依据消息中的交易产品和用户配置的路由策略将消息发送给特定的交易核心进行处理。

交易核心的功能则是处理报价和成交，它与交易辅助交互验证与扣减限额，加载基础数据，入库交易数据。核心业务处理完成后，反馈交易结果给预处理，发送DSP消息给交易辅助。

交易辅助，顾名思义是作为辅助交易核心的存在。它对外提供交易信息查询，信息设置，成交报价补全，行情计算，限额验证，基础数据同步。具体来说，交易辅助服务的设计主要为了协助交易核心完成除交易主流程外其他部分的交易处理。负责交易系统中交易数据补全，行情计算，交易数据查询、基本参数设置、数据转换发布和额度管理验证等功能其中交易补全功能会将交易相关的基础数据补全后入库保存；行情计算功能包括针对交易行情的逐笔交易行情，成交行情的行情计算；交易数据查询主要提供了对交易数据的成交、报价、行情数据查询到处功能，基础数据设置功能主要包括了交易所需的参数设置维护功能；数据交换模块提供交易数据的转换和发布等功能；额度处理模块主要包括额度维护、额度验证扣减等功能。

上述有关业务数据流程如下图所示：

 可以看出，预处理，交易核心，交易辅助在逻辑上组成了NDM业务处理的最小单位，即前文所述的业务单元。

以信用拆借交易市场为例，拆借期限最短为1天，最长为1年。目前交易中心按1天（IBO001）、7天（IBO007）、14天（IBO014）、21天（IBO021）、1个月（IBO01M）、2个月（IBO02M）、3个月（IBO03M）、4个月（IBO04M）、6个月（IBO06M）、9个月（IBO09M）、1年（IBO01Y）共11个品种进行了标的划分。

可按标的活跃度，划分出如下8个逻辑上的业务单元：

|  |  |
| --- | --- |
| 业务单元1 | 处理IBO001业务 |
| 业务单元2 | 处理IBO007业务 |
| 业务单元3 | 处理IBO014业务 |
| 业务单元4 | 处理IBO021业务 |
| 业务单元5 | 处理IBO01M业务 |
| 业务单元6 | 处理IBO02M业务 |
| 业务单元7 | 处理IBO03M业务 |
| 业务单元8 | 处理IBO04M、IBO06M、IBO09M、IBO01Y业务 |

从技术实现的实际角度来看，尽管预处理，交易核心，交易辅助在逻辑上组成了NDM业务处理的业务单元，但在部署上，可存在共享的部分，即预处理，交易核心和交易辅助的部署不用严格按照1:1:1的方式进行，更确切的说，可以按照任意A:B:C的方式进行多活部署，如下图所示。



## 双活部署模式选择

由前文所述的双活架构模式特征上看，撮合类应用的双活部署模式的可选范围不多，非撮合类则同时具备三种部署模式的选择。其中非对称双活而两种部署模式的不同选择主要在于业务性能的追求和业务连续性恢复能力的侧重上。

本币同业拆借业务提供的NDM交易模式既可以选择按标的拆分的到双数据中心的拆分双活，同时也可以进行选择对称双活进行部署。

## 业务单元的健壮性设计

#### 2PC/3PC

两阶段提交2PC是分布式事务中类型之一，两段提交就是将一个操作分两个阶段提交，第一阶段询问各个事务数据源是否准备好，第二阶段才真正将数据提交给事务数据源。三阶段提交是在二阶段提交上的改进版本，3PC最关键要解决的就是协调者和参与者同时挂掉的问题，所以3PC把2PC的准备阶段再次一分为二，这样就变成了三阶段提交。

业务单元通常由多个进程组成，其内部为了保障各个整个业务单元的状态一致，需要在各个进程间采用分布式事务的方案进行跨进程的事务处理，从而避免单进程的故障导致业务处理前后不一致。

2PC/3PC方案最大缺点就在于它的执行过程中间，节点都处于阻塞状态。各个操作的节点都会占用着一定的资源，当且仅当所有节点准备完毕，事务协调者才会通知进行全局提交，参与者进行本地事务提交后才会释放资源。因此，这样的过程会比较漫长，对性能影响比较大。

#### 本地消息表（消息队列）

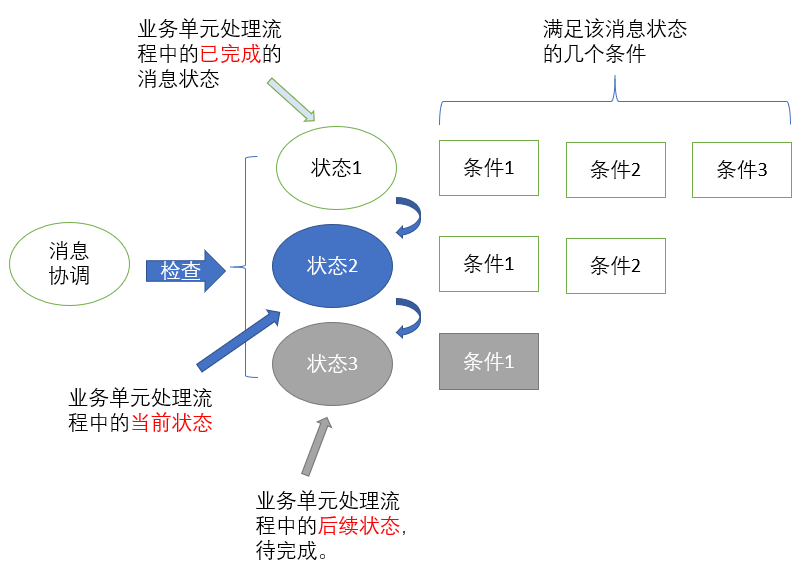
本地消息表实现方式是业界使用较多的分布式事务一致性方案，其核心思想是将分布式事务拆分成本地事务进行处理。基本思路就是：消息生产方需要额外创建一个消息表，并记录消息发送的状态。消息表和业务数据要在一个事务里提交，也就是说他们要在一个数据库里面。然后消息会经过MQ发送到消息的消费方。如果消息发送失败，会进行重试发送。

消息消费方则需要处理这个消息，并完成自己的业务逻辑。此时如果本地事务处理成功，表明已经处理成功了，如果处理失败，那么就会重试执行。如果是业务上面的失败，可以给生产方发送一个业务补偿消息，通知生产方进行回滚等操作。

生产方和消费方定时扫描本地消息表，把还没处理完成的消息或者失败的消息再发送一遍。

如果最终多次重试失败可以根据相关日志并主动通知开发人员进行手工介入。

具体到NDM系统的业务处理过程中，为保障跨机房业务连续性和流程完整性，在有入库动作时增加消息状态记录以及消息体与相关数据入库事件记录机制。当遇到机房故障时，双活可用机房中的消息协调服务可以通过业务状态机FSM机制进行状态识别，当消息协调服务收到监控发出的机房流量切换信号后，会从当前机房数据库读取出跨机房的未完成消息，验证目前消息状态，获取该消息后续状态所需要发送的消息和目标服务，而后发送消息给对应的服务，对接后续业务的服务保证业务的连续性和数据完整性。状态判断示意如下图：



通过事件判断当前消息处理状态，获取后续操作的消息状态对应的服务，然后发送对应消息到后续服务，至此保证跨机房业务连续性。

#### 消息幂等

在业务高峰期，由于网络等环境因素的影响，且生产端有可能就会重复发生了消息，这时候消费端就要实现幂等性，即意味着消费端在处理的时候，永远不会被消费多次，即使收到了一样的消息。

# 外围技术支撑实现

与双活应用实现相辅相成的部分需要外围数据中心基础设施的平台提供有力支撑，主要来自于流量接入，服务监控，数据备份，故障处理等方面。接入层主要负责把来自用户客户端（用户层）的请求流量按数据中心负载、服务健康状态等因素进行合理分发，以保障用户请求的高效、准确处理，并使得多个数据中心的资源可以得到充分利用。

在实现上述业务单元的逻辑划分后，应用双活的业务流量接入、交易结果持久化等与外围技术支持息息相关。应用双活的有效实施与应用的具体架构相关，典型的三层应用架构是接入层+应用层+数据层，每一层可根据其特点进行不同的双活容灾设计。

在确定业务单元划分和双活机房部署模式之后，需要由外围系统进行相应的业务访问支撑，以向业务单元提供连续性的服务请求、健康状态监测和故障处理。

除了业务单元层，还具体包括用户访问层、接入服务层、通用服务层、数据层以及服务健康监测处理层。



用户访问层是最终用户在个人操作系统上所看到的UI界面，包括统一终端，为会员和中心场务人员提供一个基于Windows的客户端操作界面。

接入服务层主要由统一接入平台提供身份验证、鉴权、协议转换、消息路由、服务注册等功能。细分为REST接入网关、API网关和推送网关，对外支持Http请求、Imix请求，对内统一提供ImixObject协议。

业务单元层在本文讨论的本币同业拆借为业务模型提供NDM交易来说，主要负责交易协商处理、限额扣减、业务数据落地保存，业务查询，信息设置等完整业务流程功能。

通用服务层提供登录验证服务、访问控制服务、消息连续性的协调服务。

数据层提供从企业数据模型、应用数据模型、数据交换服务、数据交换实例等端到端的管控。

## 流量接入

在同城双活场景下，若发生单个机房的故障，故障机房的流量可以快速的被引流到同城可用机房，由于业务应用本省是在双机房双活部署的，因此在业务连续性支持的能力体现上，很大程度上取决于机房间流量的切换时间，相比传统单中心业务部署所涉及故障的恢复时间有较大提升。

DNS（Domain Name System）域名访问的方式和固定 IP 地址的方式均是业务处理系统常有的两类接入方式。如何引流来自用户客户端的请求到部署相应的业务处理系统的数据中心是双/多活接入的一项关键问题。相应地，数据中心的接入层也要采用以上两种模式实现网络“双活”。在数据中心侧，目前有对应的两种技术实现：基于 DNS 的全局负载均衡和基于 RHI 的数据中心业务系统主备切换方案。

在客户端侧，可基于网络链路质量的实现自动机房切换功能，需要在数据中心的接入网关侧提供基于服务可用性的数据中心引流策略。

在互联网企业较多采用DNS技术进行服务IP地址与域名进行解耦，即向客户端提供稳定的域名地址，客户端无须关注具体的服务IP地址，即使后端服务更换了IP服务地址，对于客户端来说是无感知的。具体来说，关于DNS接入的方式。客户端通过对统一服务域名发起DNS请求，在获得可用业务服务地址列表后，发起网络连接。该方案实施较为简洁，但如果面对的是外网故障（外网链路故障、带宽攻击、接入设备故障），由于DNS的刷新频率较低，RTO<1分钟内完全靠DNS是不可行的，能考虑的办法是采用HTTPDNS技术，通过改造客户端进行HTTPDNS高频的拉或者服务端推送来实现快速切换。尽管HTTPDNS技术能够弥补传统DNS刷新率的不足，但其仅解决了一部分的网络接入问题，依然存在部分连接请求是直接基于IP地址的场景，该场景并不涉及DNS解析过程。如果数据中心服务地址不可用，将会产生直接的业务中断。

RHI（Route Health Injection，路由健康性注入）技术可以将虚拟 IP 地址的主机路由信息发布到外部网络中（通过如 OSPF、BGP等），结合服务器的运行健康状态，采用动态路由协议对虚拟 IP 地址进行宣告，通过动态路由协议的选路算法实现多接入点互为冗余，从而实现基于 IP 访问的应用的数据中心级别容灾。

具体来说，RHI 机制允许部署于两个数据中心的业务系统使用同一个 IP 地址。实际操作过程是同一个虚拟 IP 地址被不同数据中心宣告为不同的路由 metric，从而上游路由器（即外网接入路由）可以同时看到两条路径，其路由算法会将拥有更好 metric 的下一跳路径插入到其路由表中，即任意时刻只有一个数据中心将会收到实际的业务访问。当设备启用 RHI 时，它将在虚拟 IP 可用时将静态路径注入到设备的路由表中。当虚拟 IP 失效时，该静态路径将会被删除。如果该设备发生了故障，上游路由器将自动使用另一条路径到达另一数据中心的业务服务器，从而实现该业务系统的跨数据中心高可用。

如前所述，对于任何完全基于IP 访问的应用系统来说，在任意时刻，基于 RHI 的数据中心业务系统主备方案里只有一个数据中心被看作是其用来承担用户请求响应的主业务中心，另外一个作为备份业务中心。当且仅当主业务中心发生故障后，备份业务中心的应用系统才能由备切主，且同城容灾切换后IP地址不变；发生灾难时仅涉及网络主备出口切换，对客户端基本没有影响。

与上述两种基于网络协议的数据中心接入方案不同，配合用户层的基于链路质量的自动切换功能，可在数据中心的接入网关侧提供基于服务可用性的数据中心引流策略。

具体来说，对于双数据中心多活部署的应用服务，无论用户层的访问流量进入哪个数据中心，均可在同机房内部完成业务的全流程。

对于在任意时刻只“存活”于单一数据中心的应用服务，可根据业务响应时间（包括处理请求时间，网络延迟和排队延迟）的敏感程度进行灵活调整。具体来说，对于响应时间不敏感的业务请求，如果业务请求需要跨数据中心进行响应，可交由分布式平台进行跨数据中心服务处理。对于响应时间敏感的业务请求，如果业务请求需要跨数据中心进行业务响应，可向用户返回Service Unavailable等信息（如HTTP 503错误状态码），引导用户调整其服务接入地址。

## 服务路由

交易中心业务现状是大量系统处于专线网络，加上部分会员单位尚不支持DNS的解析过程，并已经针对交易中心的业务IP进行了内部地址的NAT转换，其内网交易员使用的客户端无法有效将DNS响应进行正确运用，因此基于DNS的GSLB技术在专线网的应用存在障碍。

因此，在实际操作过程中，可采取前文所述的通过在数据中心的接入网关侧提供基于服务可用性的数据中心引流策略。在客户端侧基于网络链路质量的实现自动机房切换功能，以实现业务单元的双数据中心服务接入。同时，客户端通过链路探活实现对数据中心接入IP的健康心跳监测，一旦发生接入点故障，客户端可在若干次重试后自动进行数据库中心接入地址切换。

随着接入流量的进入数据中心，业务路由可基于业务系统的切片规则对业务访问进行接入请求的路由控制，把用户请求路由到正确单元。若在接入层用户流量仍存在部分错误下发，应用层需进行二次路由校验纠错，从而保证用户请求正确的路由到对应的业务单元。

在服务路由方面，分布式服务平台支持跨数据中心服务发现与路由功能，同时，针对需要可通过UCFAPI提供机房内优先的服务访问策略，即针对进入数据中心的业务流量采取机房内优先，跨机房次优的路由策略。

以IBO市场的NDM为例，由于NDM已经按照前述逻辑拆分为8个业务单元，一旦流量进入数据中心后，需要由预处理进程通过包解析定位业务流量的属性，即该请求要求访问什么业务单元。为了向业务单元进程提供无感知的、透明的服务路由功能，通过在UCFAPI配置文件中指定SessionUnitID（产品单元）和ApplicationGroupID（业务单元），可实现不同业务单元的路由策略。

具体来说，整体架构以产品标的P进行业务分组，单元为同业务内完成该产品P运算处理相关进程的集合。总体设计思想是，dsp关于请求响应类型消息的分发以同产品P为分发组并在组内进行轮询，多活流量切换控制，可本机房，可跨机房分发消息。具体分发策略由预处理中的路由策略决定，每次分发消息会根据sessionid+产品P分组信息通知dsp服务。

具体在DSP服务的全局配置文件中，用SessionUnitID和AppGroupID标识该Application所属的业务单元和业务单元组信息。

|  |  |  |  |
| --- | --- | --- | --- |
| 符号 | 符号含义 | 序号表示样例 | 详细描述 |
| T | 交易辅助系统sessionid | TBS-TEST-0003 | 交易辅助单元中分配的sessionid 数列。 |
| S | 交易核心系统sessiond | NDTS-TEST-0010 | 交易核心单元中分配的sessionid 数列。 |
| U | 单元标识 | U1-U20 | 业务单元，是由一组完成完整业务链的多个进程组成。每个单元中可以有多个不同名进程，可有多个sessionid |
| Px | 产品标识  （即单元组标识） | P1-P8 | 产品标记如P1 = IBO001，P8=IBO09M-IBO01Y，  便于dsp消息按Px单元分组分发 |
| Ux | 多活单元 | P1U1, P1U2 | U1和U2是多活单元，可同时在线运行处理P1业务。 |
| Uxb | 备份单元 | U2b | U2b是U2的备用单元，不可同时在线运行，当U2故障时切换到备用单元U2b上线运行。字符串最后一个字符b代表备用单元。 |
| TBx | 交易辅助业务标识 | TB1-TB10，  同Px分组 | 交易辅助系统分组标识TB1~TB10 标识不同的辅助单元分组，便于dsp消息按TBx单元分组分发。 |

当应用的SessionID相同的时候，可以根据SessionUnitID进行划分不同的范围，应用在指定的范围内部进行轮询请求。若找不到指定的产品ID就会返回发送失败。

此外，通过设置Session级别配置LocalIdcFirst=1时，C端的应用会优先在本数据中心产品ID（SessionUnitID）内部进行轮询。当本数据中心P端均下线后，应用才会发请求给另一个数据中心的相同产品，实现“本机房优先，跨机房次优”的业务请求分发逻辑；当另一个数据中心的产品下线后，若找不同相同产品ID（SessionUnitID），发送请求会报错。

## 服务健康监测

业务单元需要通过UCFAPI进行内部进程的关联性上报，由监控从DSP接口中汇聚相关进程的信息，把他们标记为一个进程组，实现统一的健康监测与控制。

可依据进程状态和服务状态判断业务单元的健康状态取决于进程状态和服务状态：当进程存在且服务正常订阅时，则为健康状态；若当前业务单元存在告警信息（譬如CPU指标异常等），则为亚健康状态；否则为不健康状态。

通过将相关业务单元信息反映到监控大屏，可进行统一的调度控制管理。

## 分布式仲裁

分布式系统一般都需要一个仲裁系统，一般系统都自己提供这样的仲裁以免出现“脑裂”。“脑裂”就是 cluster 里面有两个处于主备关系的节点，它们都知道在这个 cluster 里需要选举出一个 master。那么当它们两个之间的通信完全没有问题的时候，就会达成共识，选出其中一个作为 master。

但是如果它们之间的通信出了问题，那么两个结点都会觉得现在没有 master，所以每个都把自己选举成 master，于是 cluster 里面就会有两个 master。

为了在这类场景下，依然可以可靠地把握主备关系的切换问题，因此需要进行“谁能继续运行”的结果仲裁。

目前，实现这样仲裁能力的系统有很多，例如：chubby，ZooKeeper，etcd和consul等。尽管这些系统的术语和协议不同，但是都提供基于key-value方式的分布式仲裁。ZooKeeper是第一个开源实现的仲裁软件，成为广泛使用的分布式系统后端，ZooKeeper使用心跳来判断客户端是否仍然活着。

部署设计：ZooKeeper集群中只要有过半的机器是正常工作的，那么整个集群对外就是可用的。也就是说如果有2个ZooKeeper进程，那么只要有1个ZooKeeper进程宕机，整个集群就不能工作，因为1没有过半，所以2个ZooKeeper的故障容忍度为0；同理，要是有3个ZooKeeper进程，一个宕机，还剩下2个正常的（过半），所以3个ZooKeeper的容忍度为1；事实上，2n节点和2n-1节点的故障容忍度是一样的，都是n-1，所以ZooKeeper集群默认配置为奇数个节点。

ZooKeeper提供了分布式数据的发布/订阅功能，能够让订阅者监听某一个主题对象，当这个主题对象自身状态变化时，会通知相关的订阅者，使他们能够做出相应的处理。在ZooKeeper中，引入了Watch机制来实现这种分布式的通知功能。ZooKeeper允许客户端向服务端注册一个watch监听，当服务端的一些指定时间触发了这个Watcher，那么就会向指定客户端发送一个事件通知来实现分布式的通知功能。

当客户端向ZooKeeper服务器注册Watcher的同时，会将Watcher对象存储在客户端的WatchManager中。当ZooKeeper服务器端触发Watcher事件后，会向客户端发送通知，客户端线程从WatchManager中取出对应的Watcher对象来执行回调逻辑。

### 业务单元级仲裁

如果两个业务单元存在主备关系，ZooKeeper可以帮助我们合理地从众多备选单元中选举中唯一的主业务单元，但是这样“民主”方式在面对跨机房的业务单元主备切换时，容易过于随意，任意一次主备选举后，都需要通过监控捕捉到目前服务的业务单元到底处于哪个机房。

因此，需要在一定程度上对主备选举的过程进行人工干预。因此有如下的设计：每个业务单元存在唯一的标识，因此在跨机房场景下，它的名字是唯一的。可以由管理人员（通过监控）对ZooKeeper的节点进行各个业务单元在数据中心部署规则进行标记，比如P1U1-张江-主，P1U1b-张江-备，P1U1bb-万国-备，表明处理P1标的三个业务单元部署要求是两个业务单元在张江，一主一备，第三个业务单元在万国，也是备。同时标记机房优先级为P1：张江（高），万国（低）。

每个业务单元都存在由UCFAPI提供扮演“sidecar”功能的用于统一（或多个）参与业务单元主备选举的进程E。在跨机房场景下，每个业务单元的选举进程E会根据自己的机房信息去ZooKeeper读取相关业务单元配置信息，并读取机房优先级。当万国机房的业务单元读取到机房优先级后，并且注意到张江机房尚有活着的业务单元，它就不会参与主备的选举。当张江机房的两个业务单元寻找到自身的主备标记后，处于备份角色安排的业务单元注意到主业务单元节点处于活跃状态，因此也不会进行主备的选举。所以，在正常情况下只有标记为高机房优先，且部署模式为主的业务单元健康时，它将毫无悬念成为主业务单元。监控系统通过注册ZooKeeper的节点变更信息后，可根据进程的选举结果进行相应业务单元的上线下线操作。

### 机房级仲裁

上述基于业务单元的跨数据中心仲裁可以很好地在数据中心间网络健康的情况下自主完成故障切换工作，但是当发生数据中心间网络故障时，ZooKeeper会进行脑裂处理，会让一个机房的ZooKeeper进程处于不可用状态，因此将会有一个机房的业务单元统统连接不到ZooKeeper集群参与分布式选举。属于机房级的仲裁，相关的运维操作可按该仲裁结果进行机房级故障处理。

理想情况下，相应的UCFAPI需要进行主动业务单元下线操作，相反，处于可以继续连接ZooKeeper的备份业务单元应当发现主业务单元从ZooKeeper上失去连接后，自动地触发选举为主业务单元。

但是，由于上述业务单元的部署安排是预先由运维人员定义好的，所以相应的业务单元无法主动进行选举操作，所以本方案设计由监控通过ZooKeeper API进行部署安排的策略更新操作，从而触发业务单元的自动化选举。

在机房级仲裁场景下，相应的部署安排更新操作可简单理解为监控让连不上ZooKeeper机房的所有业务单元进行下线，让可以继续提供服务能力机房的所有原备份业务单元上线。对于跨机房主-主部署的业务单元，由监控针对系统网关进行下线操作，避免仲裁选举失败的机房引入无效业务流量。

在与外围系统交互上，通过在同城双机房搭建一套ZooKeeper并向外暴露REST接口供使用（比如监控系统）。在部署方式上，原则上应在同城三机房搭建2-2-1部署的五节点ZooKeeper集群，当任一机房发生灾难时，均不影响ZooKeeper集群的正常运行。

## 业务单元管理

如前所示，业务单元的妥善管理依赖于定义双数据中心的单元化管理操作，包括单元部署规则定义，单元消息分流/切换、业务单元上/下线操作等部分组成。

单元操作：上线，下线，故障检测。

单元部署：主主部署，主备部署（含跨机房）。

单元消息分流，消息切换。

### 上线步骤

（1）ucf配置文件cfg中增加单元id，单元组id配置。

（2）每个服务的单元信息启动时上报到dsp监控，再由dsp监控采集提供接口，将单元信息上报到系统监控。

（3）dsp 监控设置每个启动进程的状态为上线状态，方可收发session通道消息。

（4）dsp 监控将进程状态信息而后单元id上报给监控系统。

### 下线步骤

（1）dsp 监控根据，ucf配置文件cfg中 增加单元id，单元组id配置。查找指定的单元id。

（2）再查找指定单元id内的每个进程，并设置状态为下线。标记为下线状态的进程无法收发session通道消息。

1. dsp 监控将进程状态信息含单元id上报给监控系统。

### 单元关闭策略设计

（1）刚性关闭：直接修改进程状态，立即停止该进程session消息通道的收发操作。副作用可能存在有数据正在处理中，当处理完毕后准备往后续服务发消息时，无法发出去的情况，造成数据已入库单消息因无法发出而丢失，业务流程的连续性被强制中断。需要事后检查并采取补偿措施修复丢失数据。

（2）柔性关闭：考虑到刚性关闭的副作用，增加单元内收发消息区分先关闭收消息进程会话通道，再延迟一段一个超时周期后关闭发送消息通道），具体方式可先构建业务单元的session链路图（有向图），可对该有向图的边进行赋权，表明session上下游的消息超时时间，通过广度优先遍历算法，在遍历每条边的时候显性等待一段时间（超时时间）后，停止该session，以达到柔性单元下线的目的。

### 故障检测

（1）检测单元内运行时的进程状态。发现单元内进程异常如心跳等，标记为故障状态。

（2）上报进程状态给监控，包含单元id信息，进程状态信息，由监控系统人工判断处置。

（3）如果单元内全部进程都为故障，则自动下线单元内全部进程停止收发消息功能，修改状态为下线。

（4）dsp 监控将进程状态信息而后单元id上报给监控系统。

## 数据层服务

数据层主要用来存储数据，一般是关系型数据库，也可以是缓存、消息队列、非结构化数据存储等。数据层容灾时，一般部署形态也是跨机房主备或集群模式。同城双活需要同时在双数据中心的机器上部署在线的业务数据，现阶段将数据分布在多个节点主要有以下两种方式：

1. 复制：即在多个节点上保存相同数据的副本，每个副本的存储位置可以不同。复制方法可以有效应对某些节点发生不可用时，其他节点可以继续提供数据访问服务。
2. 分片：将一个大体积的数据拆分成多个较小的子集即“分片”，然后将不同分片分配给不同的节点进行存储。具体有水平分片（按照某个字段的某种规则分散到多个节点库，每个节点中包含一部分数据）、垂直分片（按照业务将表进行分类并分布到不同的节点上）和混合分片（水平分片和垂直分片的结合）三种类型。

分片技术与复制技术通常结合使用，即每个分片在多个节点都有副本，这意味着某条数据记录属于特定的分片，而同样的内容会保存在不同的节点上以提供系统的容错性。

根据CAP理论，应用在分布式系统架构下如需保证强一致性C，则在分布式系统中必然降低A的指标，但由于应用的响应速度通常也是非常重要的技术指标，所以对A指标的牺牲通常会受相当多客观条件的制约。这在双活应用建设中，主要体现在所能达成系统容灾的指标上，即由完全业务连续性（RTO=0，牺牲一致性C）降级为容许一定的业务中断(RTO≠0，牺牲可用性A)以便对数据一致性进行处理，从而相应地确保RPO=0的恢复指标。

因此，前文所述的拆分双活和对称双活两种应用双活部署模式的有效实施，对数据层的分片和复制技术也提出了不同层次的要求。由于物理上存在客观的传输速度限制，传输线路本身也存在可用性问题，传输线路可能中断、拥塞、异常（错包、丢包）。因此，无论是正常情况下的传输延迟，还是异常情况下的传输中断，都会导致系统的数据在某个时间点或者时间段是不一致的，而数据的不一致又会导致业务问题；但如果完全不做冗余，系统的整体高可用又无法保证，所以双活场景下，做到存储高可用的难点不在于灾备系统中如何做好数据备份，而在于如何减少或者规避数据不一致对业务造成的影响。

从核心交易系统的双活业务需求出发，核心业务数据比如交易额度等建议是采取强一致处理，对于非核心数据比如用户管理等可采用最终一致性处理，可提高双机房业务访问吞吐量。

从数据复制技术采用的同步机制考虑，副本与分片技术所涉及的复制机制因分布式系统CAP定理的CP或AP不同选择分为强一致和最终一致数据同步机制。

对于数据库数据复制类，以NDM这类对服务响应时延不敏感的主-主部署业务应用为例，针对是否采用分片技术，主要有具体以下两种双活部署模式：

（a）同一市场同一标的同时在双中心提供服务的“标的级”的双活模式；

（b）同一市场不同标的分别在双中心提供服务的“市场级”的双活模式。

模式（a）通过利用强一致的数据复制机制针对标的数据进行跨数据中心的数据同步。用户无论在哪个数据中心接入都能对同一标的进行业务访问。

模式（b）先通过分片技术对同一市场的不同标的进行数据划分部署于不同数据中心，然后再针对标的的数据进行跨数据中心的数据同步。因此，用户针对任意标的的实施业务访问只能汇聚于单一数据中心。

模式（a）要求数据层实现分布式的“线性一致”[[1]](#footnote-1)数据读写。满足线性化的数据层，其读写请求的响应时间至少会与网络中延迟成正比。若数据中心间网络具有高度不确定的网络延迟，那么其数据平均读写性能会显著降低。模式（b）相比模式（a）在业务性能上有所提升，但需要在数据层实现自动化的数据分片技术。

目前，各类数据库产品在线性一致性数据读写和分片技术的支持能力上有所区别。

以交易系统常用的Oracle数据库产品为例，其当前版本（19c）尚不支持跨中心分布式的“线性一致性”数据读写。通过配合分布式数据库中间件尽管可以实现一定的分布式线性一致性能力，但二者的兼容性和实践效果较差。主要体现在分布式数据库中间件是架构在多个传统单点数据库系统上的中间层解决方案，通过将数据分拆到不同的数据库节点上，利用中间件来管理和访问各个数据库中的数据，通常需要用户参与到数据分拆和节点管理过程中。

同时，由于底层的每个节点都是一个独立的数据库系统，中间件很难实现分片副本在不同节点间的复制，因此多利用底层数据库的主备同步机制为每个节点配置独立的备份节点。在分布式事务处理方面，底层的节点都是独立的数据库系统，有各自的日志系统和事务处理机制，只能在中间件节点上来实现2PC，还需要在中间件节点上维护数据分布等元数据，实现查询解析、查询重写和结果聚合等功能，导致难以实现成熟的中间件产品。如前所述，尽管可以这么做，但是2PC在实际操作中的性能是较差的。

原生分布式数据库是指从架构设计、底层存储和查询处理均面向分布式数据管理需求，数据库集群作为一个整体对外提供服务，用户无需关注集群内部的实现细节。能够原生支持数据的自动分片，以及分片副本在集群节点间的自动迁移和复制。不仅如此，在分布式线性一致性读写能力上也具有原生的支持优势，其服务器利用率和管理复杂性上均优于中间件方案，但其要求应用层进行必要的适配调整和Oracle的数据迁移动作，具体涉及的改造范围和实施效果需要结合具体产品进行相应的验证，但通常是范围可控并且目标明确的。

总而言之，数据层服务作为双活应用的数据技术托底，根据具体业务需要和研发投入，它可向上层应用提供透明的跨机房弱一致、强一致、自动化数据分片复制等能力，为应用的双活实现提供了有力支撑。

# 典型故障处理场景

本场景中的故障恢复仅是需要切换机房的双活恢复方案，不考虑单机房恢复情况。下文假设故障机房为01机房，02机房为同城双活待切换机房。

## 网关故障

1、某一数据中心所有REST网关/推送网关故障，客户端需在尝试3次连接后切换；

2、某一数据中心所有API网关故障，API用户需在尝试3次连接后切换；

3、某一数据中心所有业务单元故障，REST网关/推送网关/API网关返回503错误码，客户端（包括API用户）在收到请求响应后，根据返回的503错误码，切换数据中心REST网关/推送网关/API网关接入IP；

## 交易单元故障

监控平台发现交易单元故障后恢复操作，发送通知给01机房DA注销所有服务，发送切换通知给02机房对应服务接收01切换流量消息进入切换后处理状态。运行消息协调服务，执行后续流程保障跨机房消息的业务连续性。

跨机房消息业务连续性恢复过程：

1. 消息协调服务加载数据库中含跨机房标识的最近未处理消息状态，消息体，消息与存储表关系。

2. 消息协调服务根据FSM状态机机制检查业务单元整体流程中未完成状态，获取下一步将执行的状态节点。

3. 消息协调服务给后续业务服务发送DSP处理消息，触发连续性业务处理。

4. 后续业务服务接收DSP业务消息完成后续业务处理。自此恢复跨机房业务连续性。

## 数据库故障

1、分布式数据库发生故障，由分布式数据库产品的故障恢复功能保证；

2、在数据库中间件场景下，单一机房oracle数据库故障，根据中间件的故障切换时间，恢复对应应用功能。

## 数据中心间网络故障

1、若采用分布式数据库方案(自带分布式仲裁)，发生数据中心间网络异常时，分布式数据库会根据仲裁结果选择某一数据中心为主数据中心，即在单一数据中心提供服务

那么在拆分双活的方案下，如果对应分片业务单元的数据中心非仲裁判断的主数据中心，则需要手动（亦可自动化策略）通过监控平台下线该数据中心的业务单元。

在对称双活的方案下，非仲裁判断的主数据中心应用由于连接数据库异常返回503错误，客户端参考业务单元故障，进行切换。

2、若采用数据库中间件方案，若发生数据中心间网络异常，在拆分双活和对称双活方案下，均需要手动（亦可自动化策略）通过监控平台下线某一数据中心业务单元。

# 应用双活实现总结

应用双活的高效实现同样离不开外围系统的通力配合，包括跨机房感知的服务路由、服务健康监测与控制，健壮的跨机房分布式数据访问与自动化复制、还有聚焦于业务单元故障恢复的链路消息控制等。

对于业务服务响应时延不敏感的应用（如查询、设置等业务）来说，其应用双活可轻易依托于上述外围技术支持通过较小的架构改造拥有低成本投入的双活能力。

而对业务服务响应时延敏感的应用（如撮合交易等业务）来说，双活应用的有效实施依赖于单元化的业务架构，即把单元作为系统部署的基本单位，在同城机房中部署数个业务单元，每个机房里的单元数目不定，任意一个单元都构成了业务系统所需的所有功能。涉及较多的与业务需求相关的改造成本投入，但提升了应用系统自身的业务连续性服务水平。

1. “线性一致性”（linearizability）：在一个线性一致的数据库系统中，只要一个客户端成功完成写操作，所有客户端从数据库中读取数据必须能够看到刚刚写入的值。 [↑](#footnote-ref-1)