SOFAStack

数据访问代理
使用指南

产品版本：AntStack Plus 1.11.0

文档版本：20230131

蚂蚁集团
ANT GROUP

法律声明

蚂蚁集团版权所有©2022，并保留一切权利。

商标声明

免责声明

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.什么是数据访问代理

1.1. 概述

数据访问代理是蚂蚁集团自主研发的金融级分布式数据库中间件，用于解决海量请求下数据访问的瓶颈及数据库的容灾问题，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。十年来专注于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，确保在高并发、数据库异常的情况下依然非常稳定与可靠。

数据访问代理兼容 MySQL 协议和语法，支持分库分表、平滑扩容、服务升降配、透明读写分离和分布式事务等特性，具备分布式数据库全生命周期的运维管控能力。

数据访问代理主要应用场景在大规模在线数据操作上，通过贴合业务的拆分方式，将操作效率提升到极致，有效满足用户在线业务对关系型数据库要求。

数据访问代理是一个分布式数据库系统，也是一个实现了 MySQL 协议的服务器。前端用户可以把它看作是一个数据库代理，用 MySQL 客户端工具和命令行访问，而其后端可以用 MySQL 原生协议与多个 RDS 或者 OceanBase 服务器通信。它解决了目前传统关系型数据库难以扩展、不可切分的问题，可以避免单机（单库）的性能缺陷，克服了数据存储和业务规模迅速增长情况下的数据瓶颈。

1.2. 功能特性

数据访问代理兼容 MySQL 协议和语法，支持分库分表、平滑扩容、服务升降配、透明读写分离和分布式事务等特性，具备分布式数据库全生命周期的运维管控能力。

分库分表

支持 RDS、OceanBase、MySQL 的分库分表。在创建分布式数据库后，只需选择拆分键，数据访问代理就可以按照拆分键生成拆分规则，实现数据水平拆分。

透明读写分离

通过使用数据访问代理只读实例或者 MySQL 备机实现读写分离，帮助应用解决事务、只读实例或者备机失效、指定主备访问等细节问题。对应用无侵入，在数据访问代理控制台即可完成读写分离相关操作。

数据存储平滑扩容

当出现数据存储容量和访问量瓶颈时，数据访问代理支持在线存储容量扩展，扩容无需应用改造，扩容进度支持可视化跟踪。

服务升降配

数据访问代理实例可以通过改变资源数量实现服务能力的弹性扩展。

全局唯一数字序列

数据访问代理支持分布式全局唯一且有序递增的数字序列。满足业务在使用分布式数据库下对主键或者唯一键以及特定场景的需求。

数据库账号权限体系

数据访问代理支持类单机 MySQL 账号和权限体系，确保不同角色使用的账号操作安全。

分布式事务

数据访问代理结合中间件分布式事务套件，可以支持分布式事务，保证分布式数据库数据一致性。

1.3. 产品优势

数据访问代理的主要优势如下：

- **分布式**：数据读写存储集群化，不受单机限制，业务使用无连接数限制。
- **弹性**：数据服务可升降配，数据存储扁平化 scale-up（纵向扩展） 和 scale-out（横向扩展），读写分离线性提升读能力。
- **高性能**：分库分表经典方案让操作聚焦少量数据，多种拆分方式适应数据特点，并具备特定 SQL 并行执行能力，进一步提升执行效率。
- **安全**：完整的类单机 MySQL 账号体系，提供具备授权鉴权的 OpenAPI 方便集成能力到业务管控中，产品服务支持体系化。
- **简单易用**：兼容 MySQL 协议和大部分 MySQL SQL 语法，无业务侵入式使用读写分离，支持全面的运维和监控能力。
- **成熟度高**：基于蚂蚁集团内部多年的金融级数据容灾场景，十年来专注于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，确保在高并发、数据库异常的情况下依然非常稳定与可靠。

1.4. 应用场景

海量数据读写

随着业务的快速增长，数据量不断的增大，就会出现单表/单库数据量太大、单台数据库服务器压力很大、读写速度遇到瓶颈等一系列问题。尽管可以通过增大数据库实例的物理配置得到一定程度的缓解，但无法根本解决数据库单机瓶颈。数据访问代理提供灵活的数据拆分机制，代码侵入性低，可以非常方便地实现数据的水平拆分与扩容，从而从根本上解决数据库单机瓶颈的问题。

金融级数据容灾

基于蚂蚁集团内部多年的金融级数据容灾场景，数据访问代理针对不同业务场景提供了多种机房级数据容灾解决方案，能够保障数据的稳定性与业务的连续性，为企业发展保驾护航。

数据库流量分配

当系统提供了大量查询服务，且稳定性要求较高，单数据库实例进行读写已经不能满足业务层面的要求。数据访问代理提供了基于规则的流量分发机制，通过部署多个数据库实例（如一写多读）的方式来满足业务需求。

1.5. 基本概念

中文	英文	释义
实例	instance	实例是数据访问代理的集群，相当于一个逻辑的物理数据库实例。实例基于 MySQL 标准接口，作为统一的 SQL 请求入口，所有 SQL 都需要请求给实例进行处理，实例会负责将访问请求发到正确的物理数据库上。

中文	英文	释义
数据库	database	表示数据访问代理的逻辑数据库。以这个作为访问的入口，通过分库分表、读写分离等规则可以请求到后端真实的物理数据库。
数据表	data table	表示数据访问代理的逻辑表，应用访问数据访问代理时 SQL 中的表即数据表。一个逻辑表会对应多个物理表，数据访问代理在路由时，会将逻辑表名替换成物理表名。
逻辑表	logic table	同数据表。
物理数据库	physical database	真实的物理数据库。可能是 RDS（MySQL）、OceanBase、Oracle 等。目前支持 RDS（MySQL） 和 OceanBase 两种物理数据库。
物理数据源	physical data source	同物理数据库。
物理表	physical table	物理库内存放的真实表。
数据库分片	database sharding	表示分库分表中的数据库分库。分库是一个逻辑上的概念，物理上可能是一个物理数据库代表一个“分库”，也可能是多个物理数据库组成一个“分库”，在数据访问代理里面统一概念称分片。
分片	shard	同数据库分片。
数据节点	data node	代理层后端的用户物理数据节点实例，类型可以是 MySQL（RDS）或者 OceanBase。

1.6. 基本原理

1.6.1. 分库分表

数据库访问代理在后端将数据量较大的数据表水平拆分到各个 RDS 数据库中，后端的这些 RDS 数据库被称为分库，分库中的表被称为分表。

拆分后，每个分库负责一份数据的读写操作，从而有效的分散了整体访问压力。在系统扩容时，只需要水平增加分库的数量，并且迁移相关数据，就可以提高数据库访问代理系统的总体容量。

数据拆分

数据库访问代理支持库级拆分，表级拆分和分库分表拆分，通过数据库访问代理 DDL 语句指定，具体操作参见 DDL 语法。

数据访问代理根据指定拆分键的值，采用特定的算法进行计算，然后根据计算结果将数据存储到对应的分库/分表中。拆分键即分库/分表字段，因此分为分库键和分表键。目前，拆分键只支持单个字段。

- 分库键：数据库访问代理根据分库键的值将数据水平拆分到后端的各个 RDS 分库里。键值相同的数据一定位于同一个 RDS 数据库。
- 分表键：每一张逻辑表都可以定义自己的分表键，键值相同的数据一定位于同一个 RDS 数据表。

> **说明**
>
> 在执行带有 WHERE 条件的 UPDATE、DELETE、SELECT 语句时，如果 SQL 语句中没有使用拆分键，或者虽然指定了拆分键但是范围太广，会导致 SQL 语句被分发到所有分库上执行（即全表扫描），且执行结果会在数据库访问代理中进行合并。全表扫描响应较慢，在高并发业务场景中应尽量避免使用。

1.6.2. SQL 路由

在分库分表模式下，数据访问代理会根据拆分键（即拆分字段）以及 SQL 语义把 SQL 语句分发到底层中各个存储数据的分表进行执行。执行结束后，数据访问代理会将从各个分表获取的数据合并，然后返回给用户。本文介绍在分库分表场景中数据访问代理执行 SQL 语句时的路由原理以及数据合并。

有关数据访问代理的数据拆分原理，请参考文档 分库分表。

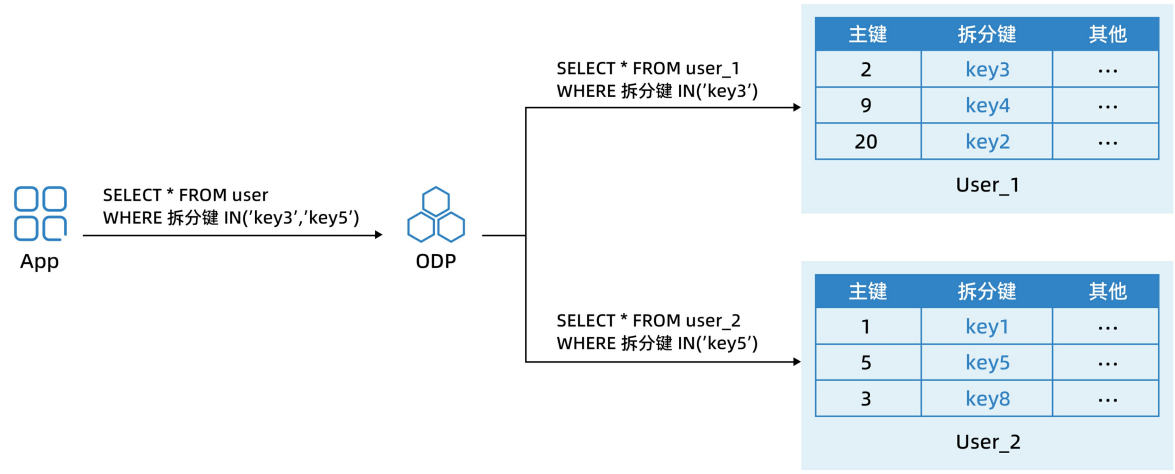
拆分键

分库分表过程中，数据访问代理按照指定的拆分键，采用特定的算法进行计算，然后根据计算结果将数据存储到对应的分表中。拆分键是数据访问代理中数据分布和 SQL 路由的凭证。

主键	拆分键	其他
1	key1	…
3	key2	…
4	key1	…
5	key1	…
9	key2	…
11	key1	…
20	key2	…

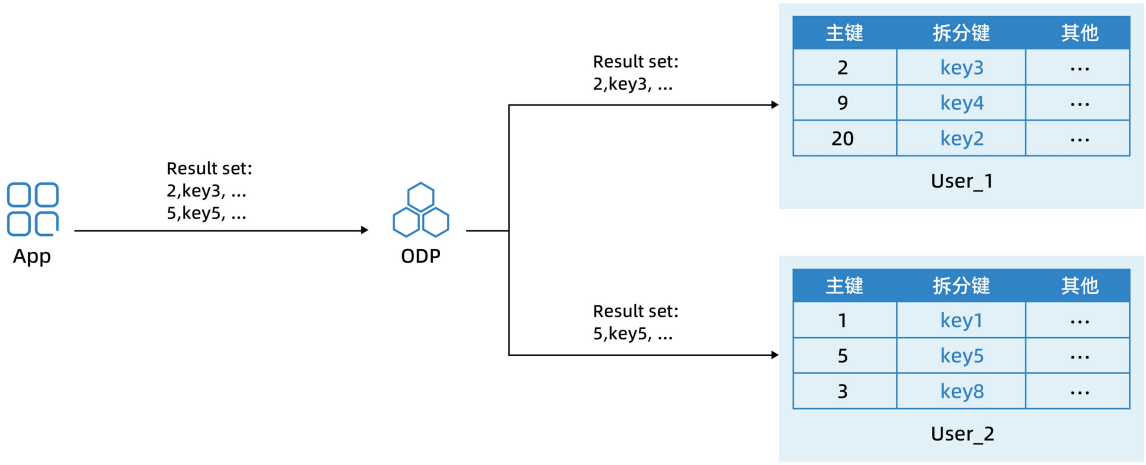
SQL 路由

当用户发起执行 SQL 语句的请求时，数据访问代理会理解 SQL 语句的含义，然后按照拆分键的值和执行策略将 SQL 语句路由到对应分区进行执行，如下图所示：



数据合并

如果一个 SQL 语句被路由到多个分表执行，数据访问代理会将各个分表返回的数据按照原始 SQL 语义进行合并，并将最终结果返回给用户。

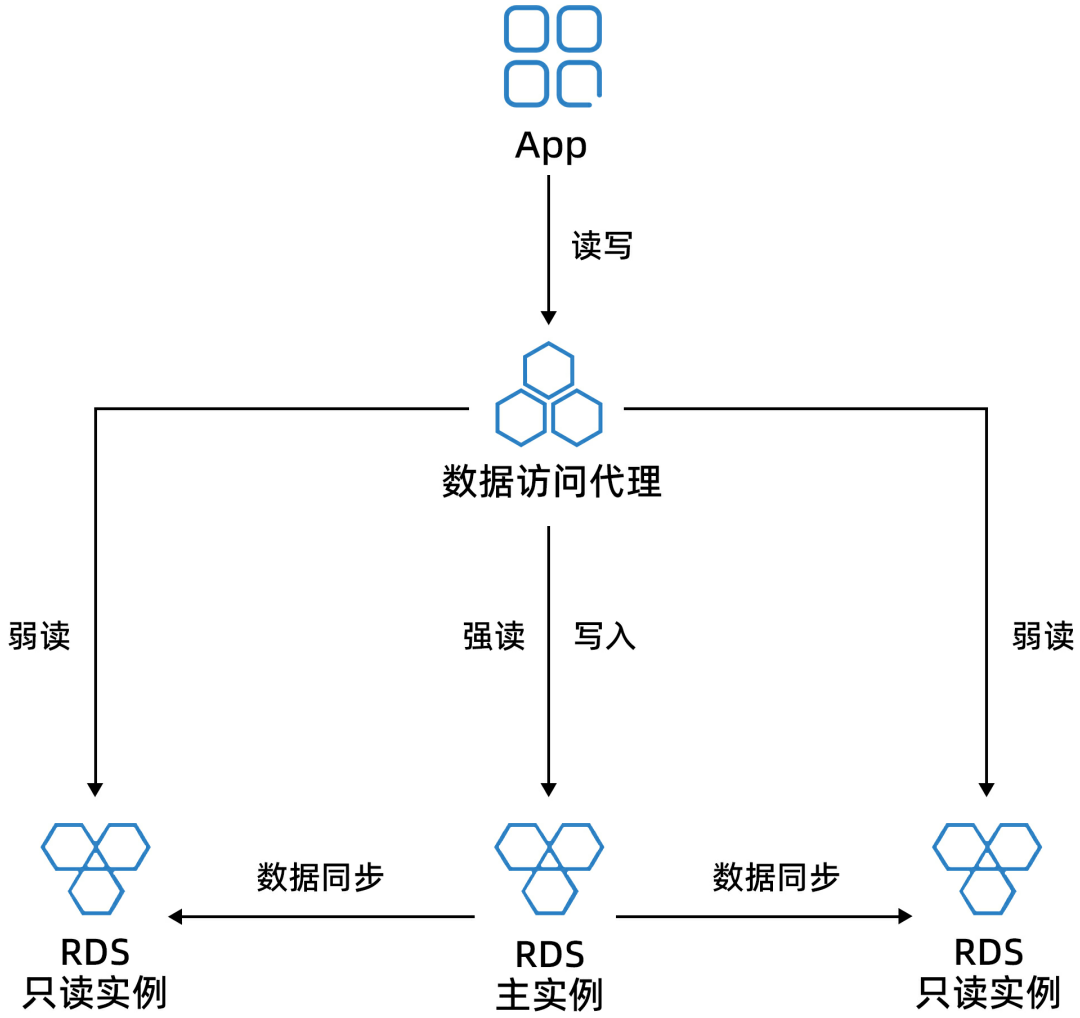


1.6.3. 读写分离

在主实例的读请求较多、读压力较大的时候，可以通过数据访问代理读写分离功能对读流量进行分流，减轻 RDS 主实例的读压力。

数据访问代理的读写分离功能是对应用透明的设计。在不修改任何应用代码的情况下，只需要在数据访问代理控制台中调整读权重，即可将读流量按配置的比例在主 RDS 实例与多个 RDS 只读实例 之间进行分流；写流量则全部到主实例，不做分流。

设置读写分离后，主 RDS 实例读取过程是强读，即实时强一致读，而只读实例上的数据是从主实例上异步复制的，存在毫秒级的延迟，因此只读 RDS 实例读取过程是弱读，属于非强一致性读。在金融级业务场景下，当需要实时、强一致读时，可以通过数据访问代理控制台的 **数据库设置** 页面来关闭读写分离功能，这时数据访问代理仅能通过强读的方式读取主 RDS 实例，保证访问该库的 SQL 语句只在主 RDS 实例上执行。


App
读写
数据访问代理
弱读
强读
写入
弱读
数据同步
数据同步
RDS
只读实例
RDS
主实例
RDS
只读实例

2.快速入门

2.1. 创建实例

数据访问代理 ODP 是蚂蚁集团自主研发的金融级分布式数据库中间件，用于解决海量请求下数据访问的瓶颈及数据库的容灾问题。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。使用本产品前，您需要创建一个数据访问代理实例。

操作步骤

1. 登录 ODP 控制台。
2. 在左侧导航栏单击 **实例**，然后单击 **创建实例**。
3. 选择云游环境，并选择是否创建双机房，然后单击 **确定**。

   新创建的 ODP 实例会出现在实例列表中。

2.2. 添加物理数据节点

数据访问代理实例创建完成后，需要添加用户存在的物理数据节点。物理数据节点可以是阿里云平台上的 RDS 数据库，也可以是 OceanBase 数据库。数据访问代理不仅支持手动添加物理数据节点，也支持导入物理数据节点。

> 重要
>
> 目前 ODP 仅支持 MySQL 5.6、5.7 和 8.0 版本，其他版本暂不支持。使用错误的版本会在创建逻辑库时报错：“所使用的账号并非为高权账号”。

- 添加物理数据节点：在数据访问代理控制台中，手动添加 RDS 或 OceanBase 物理数据节点。需要注意的是，如果被添加的实例开启了白名单功能，您还需要前往对应的 RDS 或 OceanBase 控制台为数据访问代理实例 设置白名单 。
- 导入物理数据节点：在数据访问代理控制台中，导入 RDS 物理数据节点。导入成功后，在对应的 RDS 实例中会自动加入数据访问代理实例的白名单。因此，对于 RDS 实例，推荐使用 **导入物理数据节点** 的方法。

前提条件

请确保您已经在 RDS 或 OceanBase 控制台购买相应的数据库资源，并获取其实例 ID 及实例链接地址，才可以将其添加为数据访问代理数据库的数据存储节点。

方法一：添加物理数据节点

1. 在数据访问代理控制台左侧导航栏中，选择 **运维** > **物理数据节点** > **添加节点**。

2. 在弹出的对话框中，输入相关物理数据节点信息进行绑定。

- **物理节点 ID**：必填，待添加的 RDS 或 OceanBase 节点的实例 ID，需要从其对应产品的控制台中实例详细信息中获取。在专有网络（VPC）下连接 OceanBase 时，物理节点 ID 须填写实例 ID 后缀的数字。

  > **说明**
  >
  > 例如，假设 OceanBase 节点实例 ID 为 `obtest1234`，请填写 `1234`。

- **链接地址**：必填，待添加的 RDS 或 OceanBase 节点的实例链接地址，需要从其对应产品的控制台中实例详细信息中获取。最多 128 个字符。
- **数据库类型**：必填，可选类型为 MySQL 或 OceanBase。如果数据节点为 RDS，则请选择 MySQL。
- **描述**：对于这个物理数据节点的描述信息。
- **网络类型**：需要明确当前物理数据节点所处于的网络类型，需要从其对应产品的控制台中实例详细信息中获取。

- **VPC ID**：必填，如果当前物理数据节点处于专有网络中，则需要指定其 VPC ID，需要从其对应产品的控制台中实例详细信息中获取。

3. 单击 **确定**，即可完成节点添加。

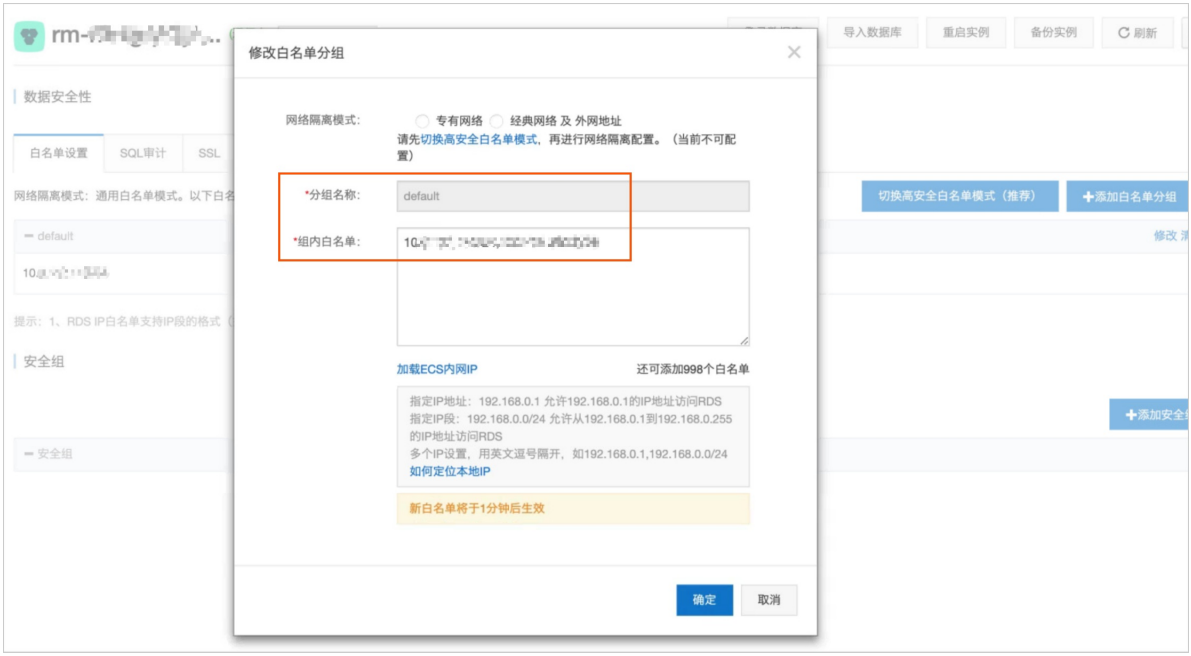
设置白名单

如果被添加的物理数据节点（即 RDS 或 OceanBase 实例）开启了白名单，则需要获取当前数据访问代理实例的 IP/IP 网段，并登录 RDS 或 OceanBase 的控制台界面，在其（RDS 或 OceanBase）白名单中添加数据访问代理的 IP 或 IP 网段。

以 RDS 为例，操作步骤如下：

1. 进入数据访问代理控制台页面，选择 **实例**，单击目标数据访问代理实例名，进入该实例的详情页。
2. 在 **白名单** 标签下，单击 **数据访问代理机器列表** ，获取当前实例所在的 IP（或安全 IP 网段）。

3. 登录 RDS 控制台，单击目标实例的实例名称，进入基本信息页面，单击 **设置白名单**。
4. 在白名单设置页面中，找到目标白名单分组后单击 **修改** 或单击 **+添加白名单分组** 添加新分组。
5. 在新弹出窗口中，确定分组名称，并在 **组内白名单** 填入刚刚获取到的数据访问代理实例的 IP 地址。



6. 单击 **确定**。新白名单将在 1 分钟内生效。生效后，该物理数据节点即可访问和使用。

方法二：导入物理数据节点

1. 在数据访问代理控制台左侧导航栏中，选择 **运维 > 物理数据节点 > 导入节点**。
2. 在弹出的对话框中，在已获取的 RDS 实例列表中，选择目标实例，并单击 **确定**。

导入后，数据访问代理会自动创建物理数据节点，并在对应的 RDS 实例中加入数据访问代理实例的白名单。

2.3. 创建数据访问代理数据库

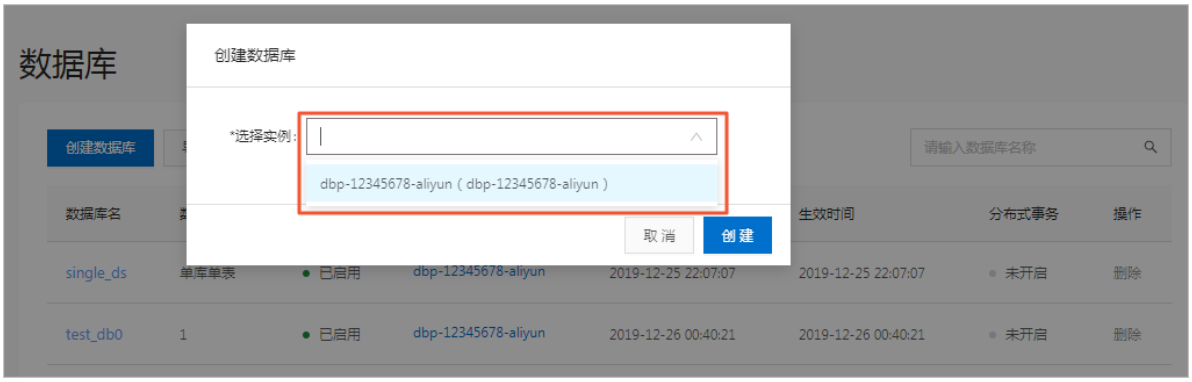
创建完数据访问代理实例后，需要创建数据访问代理数据库。

创建数据访问代理数据库和创建普通 MySQL 数据库有两点不同：

- 创建数据访问代理数据库的操作只能在控制台上进行；
- 创建数据访问代理数据库时，需要选择若干（包含一个）RDS 或 OceanBase 作为数据存储节点（如果没有，需先购买并参见 添加物理数据节点 进行添加）。

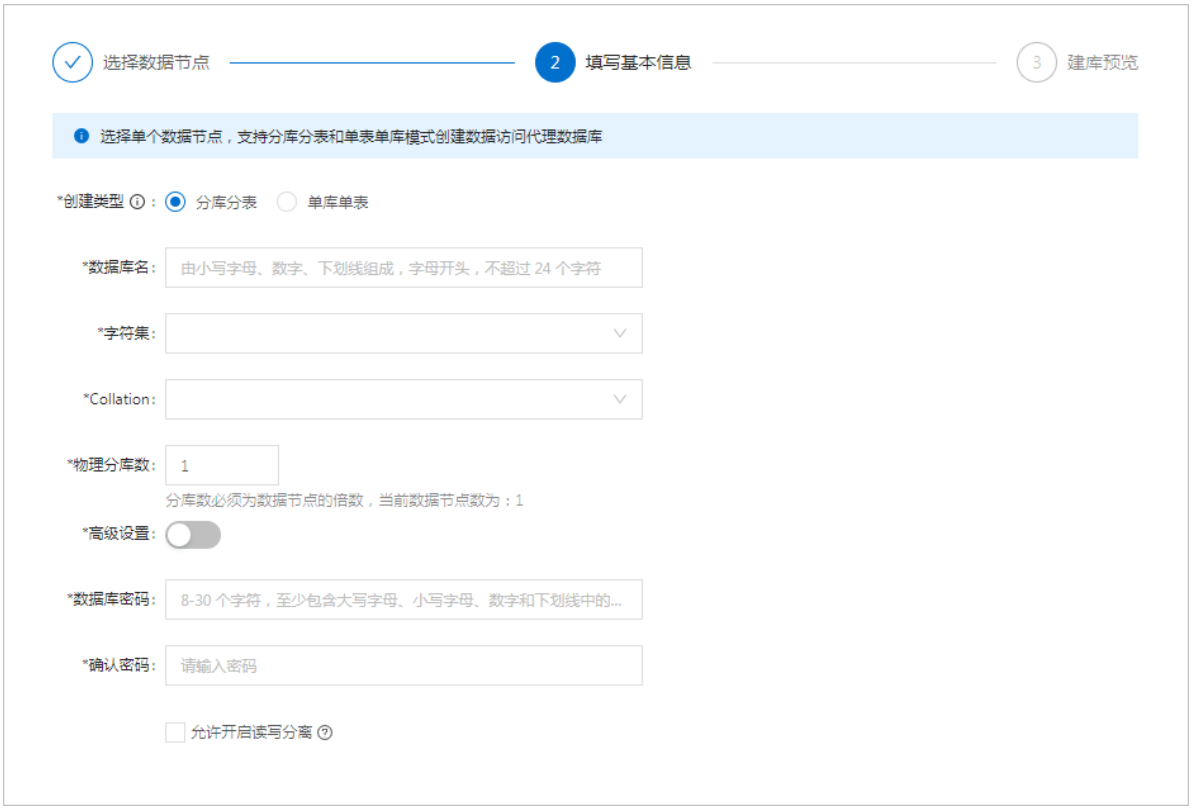
操作步骤

1. 在数据访问代理控制台左侧导航栏，选择 **数据库**，进入数据库列表后，单击 **创建数据库**。
2. 在弹出的 **创建数据库** 窗口中，选择指定的数据访问代理实例来创建数据库，单击 **创建** 进入 **选择数据节点** 页面。



3. 根据需要勾选 RDS 或 OceanBase 作为当前数据库的存储节点，单击 **下一步**。

4. 根据提示，填写或选择基本信息。



- **创建类型**：
  - **分库分表**：则需要设置对应的分库数，分表规则会在创建数据表时指定；
  - **单库单表**：将已有的数据库交由数据访问代理进行代理访问，实现统一管理及读写分离的功能，修改数据库连接串和用户名密码即可，无需进行数据导入或修改代码。
- **数据库名**：由小写字母、数字、下划线组成，以字母开头，不超过 24 个字符。
- **物理分库数**： 物理分库用于均匀承载分片，默认物理分库数等于分片数，如果修改则需保证分片数是物理分库数的整数倍，同时物理分库数也是物理节点的整数倍。
- **高级设置**：

- **分片数**：在分库分表模式下，需要指定分片数。为保证数据库后期的水平扩容，分片数需要能够平均分配在上一步选择的所有物理数据节点上。所以分片数必须为数据节点的倍数。

- **数据库密码**：8~30 个字符，大、小写字母、数字、下划线四者中至少包含三种。
- **允许开启读写分离**：考虑到业务可能对数据强一致性的需求，这里默认不开启读写分离。当开启后，对该库的读访问将会根据节点权重的设置被路由在各个只读节点。

5. 建库预览，确认信息正确。
   - 确认之前输入信息的正确性，包括分库数，分片数等，数据访问代理会根据上一步选择的分库数在指定的物理数据节点上创建物理库。

   - 确认是否自动创建物理数据库：
     - 选择 **是**，则表示将由数据访问代理来在指定的物理数据节点上根据在上一步中指定的规则创建物理分库，这里需要输入各个物理节点的高权限账户以便自动执行库 DDL 创建物理库。
     - 选择 **否**，则表示在指定的物理数据节点中物理数据库已经创建完成，且创建规则符合上一步指定的分库规则，这种情况下就不需要再输入物理数据节点的高权限账户。

6. 信息确认完毕，单击 **创建** 开始创建数据库。
7. 数据库创建完毕后，页面会自动进入新创建数据库的详情页面，单击 **上线** 启用当前数据库。数据库状态会从 **未启用** 变为 **已启用**。

2.4. 创建数据访问代理数据表

创建完数据访问代理数据库后，与普通单机数据库一样，数据访问代理也需要创建数据表。对于设置为“分库分表”模式的数据访问代理数据库来说，在创建数据表时需要制定其相应的分表规则，之后应用通过连接数据访问代理可以透明访问后端的数据表，由数据访问代理根据设定的分表规则进行路由操作。

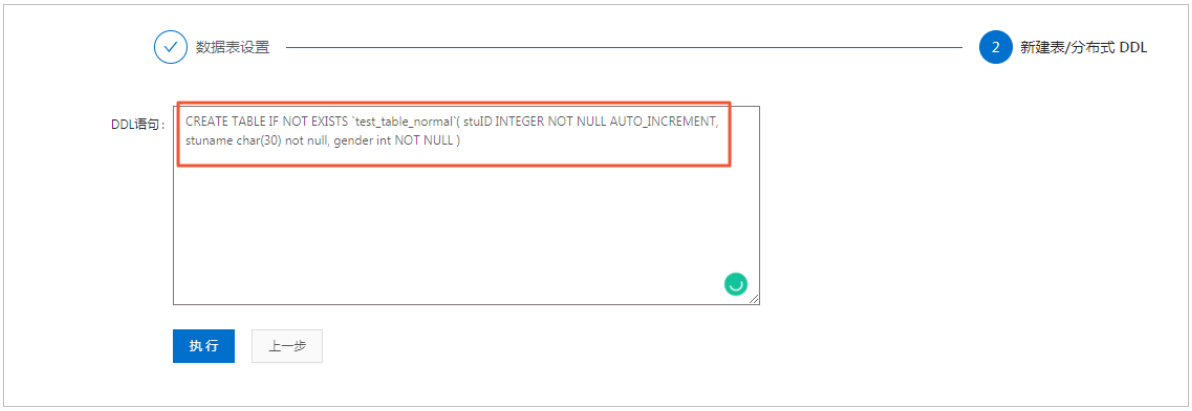
操作步骤

1. 在数据访问代理控制台左侧菜单栏选择 **数据库**，进入数据库列表。

2. 在列表中，单击需要建表的数据库名称进入该数据库基本信息页，单击右侧页面下方的 **新增数据表**。
3. 进入创建数据表向导，填写或选择各项信息，并单击 **下一步**。

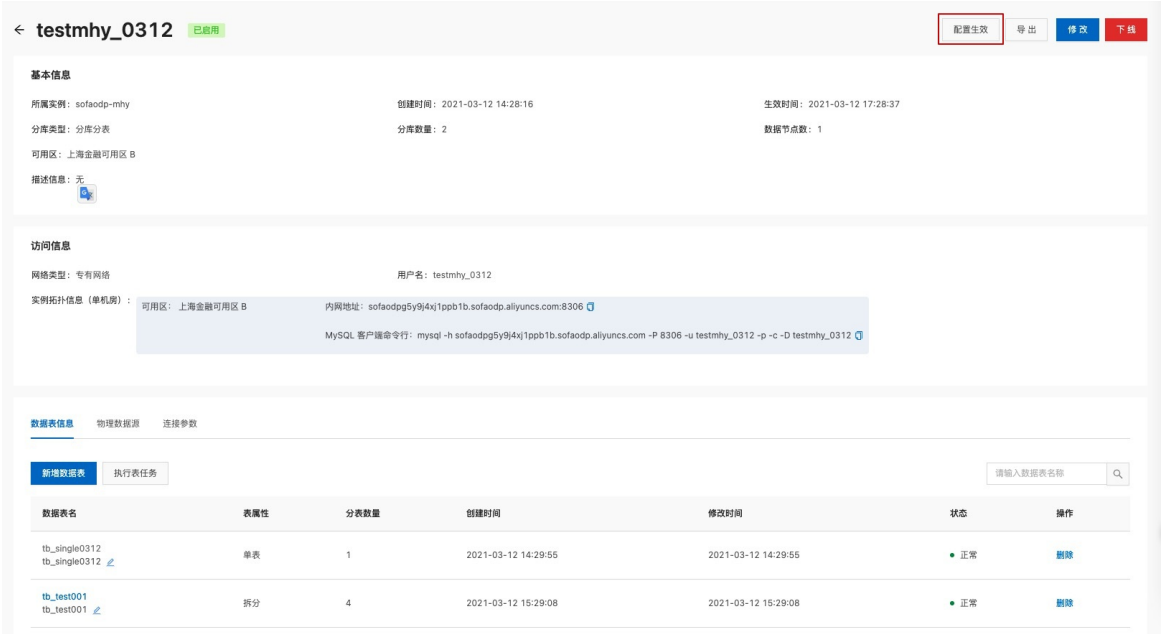
   - 表属性：
     - 选择 **单表** 则无需设置拆分的路由规则。
     - 选择 **拆分表** 后需要指定对应的分表数以及路由规则。
   - 分表总数：所指定的分表数会被均分在当前数据库的各个物理库中。
   - 分表规则：
     - 在拆分表（分表）模式下数据访问代理提供上层应用透明的数据表访问模式，所以这里需要设置分表路由规则，数据访问代理会对应用的 SQL 进行自动路由。
     - Hash 取模：数据访问代理会以指定的路由字段与数据库数量进行取模散列。
     - 字符串截取：数据访问代理会以指定的路由字段中指定范围的字符串来进行散列。
     - 自身位：指定路由字段数值来进行直接路由。
     - 自定义：如果上述路由操作无法满足需求，用户也可以直接定义数据路由表达式，数据访问代理会根据指定的路由字段以及自定义表达式来进行路由。自定义分表规则支持 Groovy 表达式，更多详情参见 自定义分表规则。
   - 现在创建物理表：选中时点击 **下一步** 会根据设置的分表数进行物理表的创建。在没有选中的情况下点击 **下一步** 会完成数据表（逻辑表）的创建，但是此时还没有物理表，需要手动进行关联。
   - 高级设置：可以进行分库规则的自定义设置，规则同“分表规则”。
4. 执行 DDL 语句创建数据表（如果上一步选择了 **现在创建物理表**）。这里可以通过直接输入 DDL 语句来进行表 DDL 操作，单击 **执行** 触发，比如输入：

```
 CREATE TABLE IF NOT EXISTS `test_table_normal`( stuID INTEGER NOT NULL AUTO_INCREMENT, s
tuname char(30)notnull, gender int NOT NULL )
```



> **说明**
>
> 执行详细信息可以在控制台左侧的 **运维 > 任务管理** 中的任务列表进行查询。

5. 查看并应用创建后的数据表。
   i. 在左侧菜单栏选择 **数据库**，进入数据库列表。
   ii. 找到新创建数据表所在的数据库，单击数据库名进入详情页面。
   iii. 在下方的 **数据表信息** 标签页中可以看到新创建的数据表。这时新创建的数据表尚未生效，客户端无法进行查询或者其他操作，需要单击右上方的 **配置生效** 来生效当前的数据表变更。



数据表的状态有 **正常**、**创建中**、**创建失败**、**删除中**、**删除失败** 四种。

> **说明**
>
> 单击 **配置生效** 只会将状态为 **正常** 的数据表配置生效。

   iv. 单击 **确定** 后，新创建的数据表（包括所有该数据库下的变更）均会生效。

2.5. 连接数据访问代理

在控制台创建数据访问代理实例、创建数据访问代理数据库、创建表之后，需要通过连接数据访问代理进行后续的数据库操作。

操作步骤

1. 在数据访问代理控制台左侧菜单栏，选择 **数据库** 进入数据库列表。
2. 找到需要连接的数据库，单击名称进入数据库详情页面。
3. 在详情页中，**访问信息** 下方的 **MySQL 客户端命令行** 即为该数据库的连接信息。

4. 获取到连接信息后，可通过 **第三方工具** 或者 **程序代码** 两种方式进行连接。
   - **第三方工具**数据访问代理遵循 MySQL 标准交互协议，所以支持第三方工具的连接和使用。

     > **说明**
     >
     > 数据访问代理完全兼容 MySQL 官方命令行客户端（版本 5.1+）。数据访问代理不支持 MySQL 的历史版本（如 3.x、4.x 等版本）的指令和不常用指令，因此数据访问代理只承诺第三方 GUI 客户端可执行基础的数据库操作，包括数据的增删改查和 DDL 操作。

   - **程序代码**主要通过 MySQL 官方驱动或者第三方符合 MySQL 官方交互协议的程序进行连接。
     - 数据访问代理支持的客户端工具：
       - MySQL 命令行（推荐）
       - MySQL Workbench（推荐）
       - SQLyog
       - Sequel Pro
       - Navicat for MySQL

- 数据访问代理支持的程序驱动：
  - JDBC Driver for MySQL (Connector/J)

```
//JDBC
Class.forName("com.mysql.jdbc.Driver");
Connection conn =DriverManager.getConnection("jdbc:mysql://127.0.0.1:1234/sample_schema","sample_user","sample_password");
//...
conn.close();
```

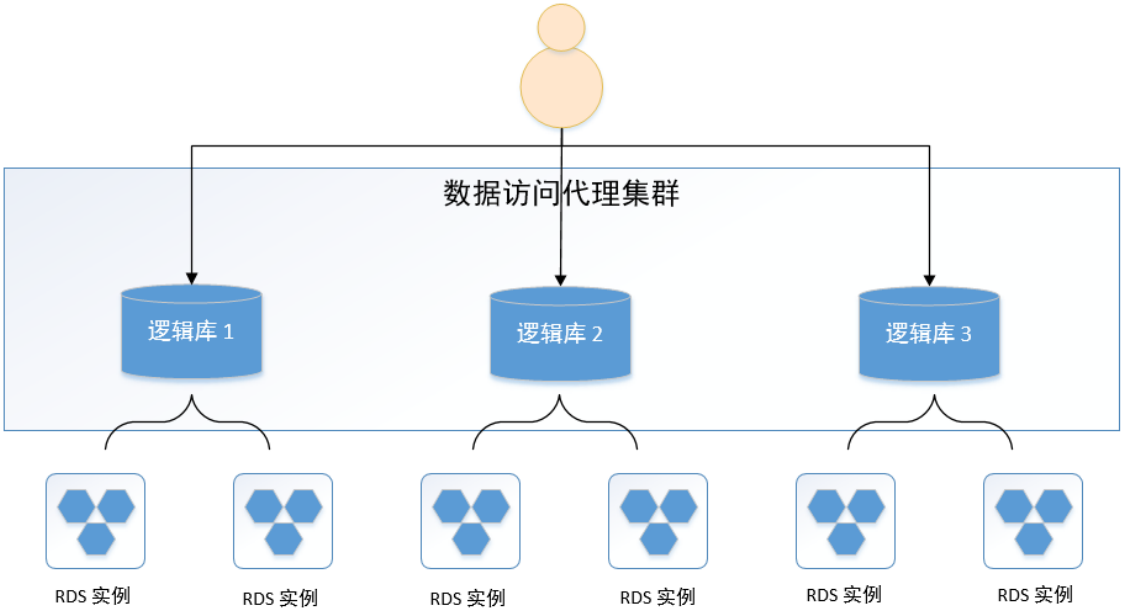
  - Python Driver for MySQL (Connector/Python)
  - C++ Driver for MySQL (Connector/C++)
  - C Driver for MySQL (Connector/C)
  - ADO.NET Driver for MySQL (Connector/NET)
  - ODBC Driver for MySQL (Connector/ODBC)
  - PHP Drivers for MySQL (mysqli, ext/mysqli, PDO_MYSQL, PHP_MYSQLND)
  - Perl Driver for MySQL (DBD::mysql)
  - Ruby Driver for MySQL (ruby-mysql)

3.实例管理

3.1. 实例介绍

数据访问代理实例在物理上是由多个数据访问代理服务器节点组成的分布式集群。数据访问代理的数据库是逻辑概念，只包含元信息，具体数据由后端连接的物理库存储。目前，只提供 **专享实例** 类型的数据访问代理实例。

用户专享的数据访问代理集群的架构图如下所示：



专享的数据访问代理实例旨在满足金融级用户的业务需求，通常用于以下场景：

- 资源隔离，数据安全性要求高；
- 业务压力大，对数据库性能要求高；
- 业务压力存在阶段性变化，需要动态变更配置来适配不同场景的业务压力。

3.2. 创建实例

数据访问代理 ODP 是蚂蚁集团自主研发的金融级分布式数据库中间件，用于解决海量请求下数据访问的瓶颈及数据库的容灾问题。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。使用本产品前，您需要创建一个数据访问代理实例。

操作步骤

1. 登录 ODP 控制台。
2. 在左侧导航栏单击 **实例**，然后单击 **创建实例**。
3. 选择云游环境，并选择是否创建双机房，然后单击 **确定**。

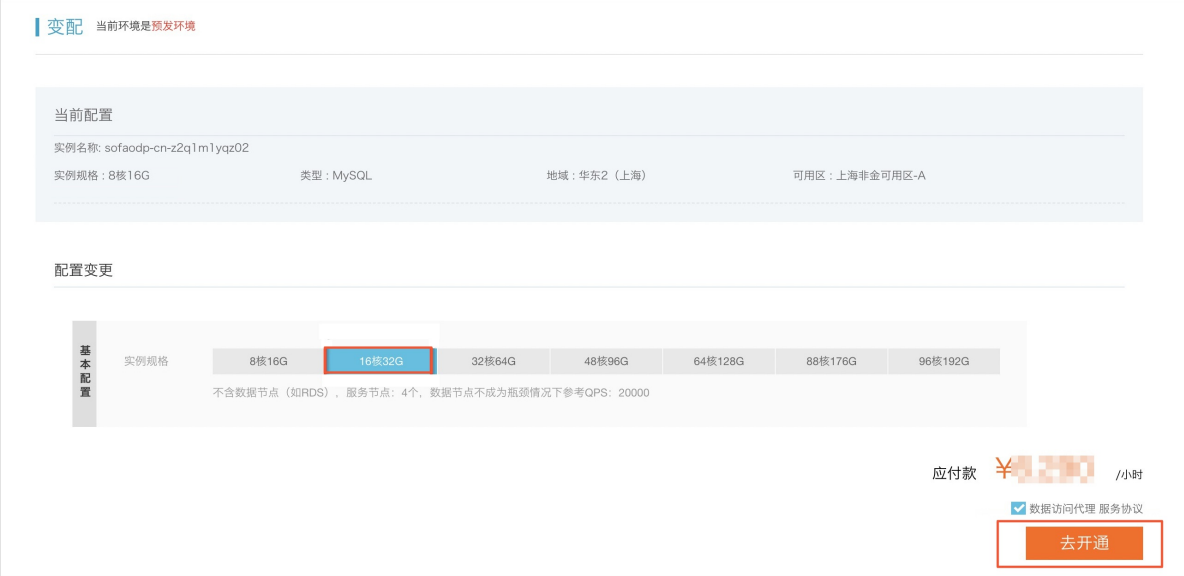
   新创建的 ODP 实例会出现在实例列表中。

3.3. 变配实例

数据访问代理支持实例变配，即修改实例规格，扩缩容节点数，可在业务无损的情况下提升服务能力。

操作步骤

1. 进入数据访问代理控制台，单击左侧导航栏上的 **实例** 后，进入数据访问代理实例列表。
2. 实例列表中，找到待变配的实例，单击操作列的 **变配** 按钮。
3. 在 **变配** 页面，选择您想要更改的目标实例规格，勾选 **数据访问代理服务协议**，单击 **去开通**。



4. 返回实例列表，可看到该实例的状态变为 **变配中**。

> **说明**
>
> 实例处于变配中状态不会影响现有数据访问代理的服务能力。

5. 几分钟后，即可完成变配，状态变为 **运行中**，同时配置规格将发生相应变化。

3.4. 释放实例

本文介绍如何释放正在使用中的数据访问代理实例。

前置条件

释放实例前，确保该实例中所有已创建的（逻辑）数据库均已被删除。

操作步骤

1. 进入数据访问代理控制台，单击左侧导航栏上的 **实例** 后，进入数据访问代理实例列表。
2. 实例列表中，选择指定的实例，单击 **释放**，确认后即可释放该实例。

> **说明**
>
> 实例释放后，将不再产生费用。 执行释放操作后，数据访问代理实例的所有数据将无法找回。

3.5. 外网地址

数据访问代理实例创建成功后，会自动创建一个内网地址。您还可以根据业务需要自行选择申请公网访问，创建公网 SLB 和公网域名。

申请外网地址

1. 在实例的详情页中，单击右上角的 **申请外网地址**。

2. 开通后，即可在实例的 **基本信息** 中获取具体的公网地址及端口号，如下图所示。

释放外网地址

如需停止提供公网访问，单击实例详情页右上角的 **释放外网地址**，并确定即可，如下图所示。

说明

释放后，公网 SLB 与域名将被释放，无法再访问，请谨慎操作。

4.权限管理

4.1. 使用 IAM 账号管理权限

4.1.1. 打开 ODP 权限开关

IAM（Identity and Access Management）是基于蚂蚁科技多年发展，并结合业内其他厂商的方案，孵化出的一套通用、灵活的身份管理、认证及访问控制解决方案。数据访问代理 ODP 已接入 IAM，您只需要打开 ODP 的权限开关，即可通过 IAM 管理内部用户的权限。

IAM 服务器部署完成后，您可以在 odpconsole 的元数据库中，通过如下命令开启 ODP 权限：

```
insert into zdalproxy_config (
gmt_create,
gmt_modified,
env_tenant,
env_mode,
config_key,
config_value,
description
)values(
  now(),
  now(),
  "-",
  "-",
  "global_auth_check_with_iam_in_private",
  "true",
  "IAM权限管理"
)
```

4.1.2. 创建 ODP 角色

权限管理主要体现在对角色的管理，角色是若干权限的集合，是权限分配的载体。本文介绍如何创建 ODP 权限角色。

操作步骤

您可以根据实际需要生成用于 ODP 服务相关权限的角色，例如 ODP 管理员、ODP 成员等。操作步骤如下：

1. 登录 IAM 控制台。
2. 在左侧导航栏选择 **账号管理 > 权限管理**。
3. 单击 **角色** 页签，然后单击 **新增角色**。
4. 在基本设置中输入角色名称和描述，然后单击 **下一步**。
5. 在选择权限步骤中选择角色需要的权限。

   您可通过以下任一方式选择角色需要的权限：

- 在权限列表中，选中 ODP 相关权限。

- 在 **根据已有角色选取** 下拉列表选择一个或多个已有的角色，并以该角色为基础，通过增减权限来创建自定义角色。
- 在页面右上方的权限搜索栏中，输入要查询的 ODP 权限，搜索并选择权限。

  权限选择完成后，您还可以单击页面右下方的 **查看已选权限**，浏览已选的权限。

6. 确认无误后，单击 **确定**。

4.1.3. 创建租户成员并授权

拥有用户管理权限的成员（如空间管理员）可以在管理控制台中管理租户成员，并根据角色自定义各成员的操作权限。

新增成员

1. 登录 IAM 控制台。
2. 在左侧导航栏选择 **账号管理** > **账号管理**。
3. 单击 **租户成员** 页签，然后单击 **新增成员**。
4. 在 **新增成员** 对话框输入新成员的登录邮箱、昵称、姓名、手机号和座机号。

   其中座机号为选填项。
5. 单击 **确定**。

为成员授权

新增成员仅拥有观察者权限，您需要使用空间管理员或拥有成员管理权限的用户可对租户成员进行授权，使该成员拥有角色对应的权限。操作步骤如下：

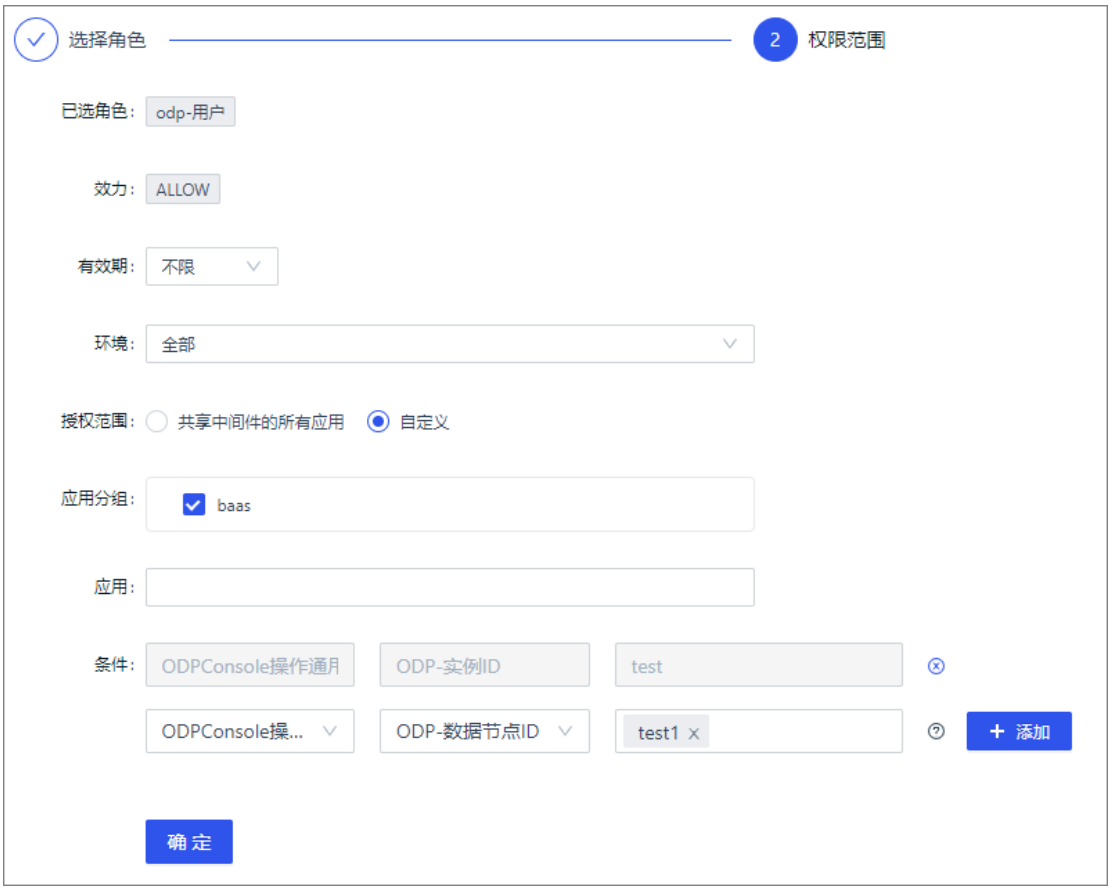
1. 在 **租户成员** 页签，单击目标成员右侧 **授权**。
2. 在 **添加授权** 页面，从角色列表中选择要赋予成员的角色，然后单击 **下一步**。

   您可以根据需要为该成员选择合适的 ODP 角色。您可以按照角色名或权限搜索角色。
3. 设置权限范围。

   您可以为成员设置权限的生效范围，例如只针对指定环境、应用、条件等生效。
   - **有效期**：可以指定权限生效的时间。可以指定为不限、1 天、2 天、3 天、7 天、14 天、30 天或自定义时间。
   - **环境**：可以指定权限生效的环境，该权限将只在指定环境中生效。
   - **授权范围**：
     - **共享中间件所有应用**：针对中间件所有应用生效。

- **自定义**：自定义权限生效的应用。
  - **应用分组**：选择应用分组，权限将针对分组内所有应用生效。
  - **应用**：选择指定应用，权限将只针对目标应用生效。
  - **条件**：通过筛选条件，进一步指定权限生效的资源或服务。例如设置权限仅针对 ODP 的指定实例生效。

    单击 **添加** 可添加多个条件。



4. 单击 **确定**。

4.2. 使用 ODP 账号管理权限

4.2.1. 概述

通过数据访问代理的账号管理功能，您可以创建具有不同权限的账号，实现对数据库访问的权限控制，例如只读、读写、DDL 和 DML。您还可以根据需要随时修改权限、重置密码等。

账号类型

数据访问代理服务支持两种类型的账号：高级账号和普通账号。它们之间的区别如下：

区别类型	高级账号	普通账号
权限范围	拥有读写权限，详见 账号权限。	分为四种权限类型：只读、读写、DDL、DML，详见 账号权限 。
创建方式	创建数据库时系统自动生成，详见 创建和管理账号。	需在控制台手动创建，操作指导详见 创建和管理账号。
管理操作	可修改密码，操作指导详见 创建和管理账号。	可修改密码、修改权限、删除账号，操作指导详见 创建和管理账号。

账号权限

高级账号和普通账号的具体操作权限如下：

账号类型	权限类型	权限				
高级账号	-	SELECT	INSERT	UPDATE	DELETE	CREATE
		DROP	RELOAD	PROCESS	REFERENCES	INDEX
		ALTER	CREATE TEMPORARY TABLES	LOCK TABLES	EXECUTE	REPLICATION SLAVE
		REPLICATION CLIENT	CREATE VIEW	SHOW VIEW	CREATE ROUTINE	ALTER ROUTINE
		CREATE USER	EVENT	TRIGGER	-	-
	只读	SELECT	LOCK TABLES	SHOW VIEW	PROCESS	REPLICATION SLAVE
		REPLICATION	CLIENT	-	-	-
		SELECT	INSERT	UPDATE	DELETE	CREATE

账号类型	权限类型	权限				
普通账号	读写	DROP	REFERENCES	INDEX	ALTER	CREATE TEMPORARY TABLES
		LOCK TABLES	EXECUTE	CREATE VIEW	SHOW VIEW	CREATE ROUTINE
		ALTER ROUTINE	EVENT	TRIGGER	PROCESS	REPLICATION SLAVE
		REPLICATION	CLIENT	-	-	-
	DDL	CREATE	DROP	INDEX	ALTER	CREATE TEMPORARY TABLES
		LOCK TABLES	CREATE VIEW	SHOW VIEW	CREATE ROUTINE	ALTER ROUTINE
		PROCESS	REPLICATION SLAVE	REPLICATION CLIENT	-	-
	DML	SELECT	INSERT	UPDATE	DELETE	CREATE TEMPORARY TABLES
		LOCK TABLES	EXECUTE	SHOW VIEW	EVENT	TRIGGER
		PROCESS	REPLICATION SLAVE	REPLICATION CLIENT	-	-

4.2.2. 创建和管理账号

本文介绍如何为数据访问代理实例创建账号、修改账号密码和权限，以及删除账号。

创建账号

创建普通账号

1. 登录数据访问代理控制台，单击左侧导航栏的 **实例**。
2. 在实例列表中，单击目标实例名，进入该实例的详情页面。
3. 单击 **账号管理** 页签，然后单击 **创建账号** 按钮。
4. 在 **创建数据库账号** 对话框，填写账号名称和密码、选择需要授权的数据库并设置相应的权限，以及填写备注信息（可选）。

   关于各权限类型包含的具体操作权限，请参见 账号权限。
5. 单击 **确定** 按钮。

创建高级账号

高级账号无法手动创建，只能由系统自动生成。

当您创建数据库时，系统会同时为该数据库生成一个同名的高级账号，即该高级账号的名称就是该数据库的名称；初始密码是您创建数据库时设置的密码，之后可以修改。

要查看高级账号信息，从控制台进入 **实例** > **账号管理**。在 **账号类型** 一列显示 **高级** 的账号即为高级账号。

修改账号密码

1. 登录数据访问代理控制台，单击左侧导航栏的 **实例**。
2. 在实例列表中，单击目标实例名，进入该实例的详情页面。
3. 单击 **账号管理** 页签，找到目标账号，单击其 **操作** 列的 **修改密码**。
4. 在 **修改密码** 对话框，输入新密码，然后单击 **确定** 按钮。

修改账号权限

您只能对普通账号能进行修改权限的操作。在修改权限时，您可以更改该账号关联的数据库，并更改相应的权限类型。

1. 登录数据访问代理控制台，单击左侧导航栏的 **实例**。
2. 在实例列表中，单击目标实例名，进入该实例的详情页面。
3. 单击 **账号管理** 页签，找到目标账号，单击其 **操作** 列的 **修改权限**。
4. 在 **修改权限** 对话框，根据需要更改授权数据库和相应的权限类型，然后单击 **确定** 按钮。

删除账号

您只能删除普通账号。高级账号不允许被删除。删除普通账号前，请确保应用程序中无该账号的连接和操作，否则会引起应用程序报错。

1. 登录数据访问代理控制台，单击左侧导航栏的 **实例**。
2. 在实例列表中，单击目标实例名，进入该实例的详情页面。
3. 单击 **账号管理** 页签，找到目标账号，单击其 **操作** 列的 **删除**。
4. 在弹出的对话框中单击 **确定删除**。

5.数据库管理

5.1. 数据库管理概述

数据库访问代理的数据库是一个逻辑上的概念，类似于 MySQL 中的数据库，它为前端应用提供了一个完整的数据库使用接口。在物理存储上，数据访问代理底层存储由一个或者多个 RDS/OceanBase 实例组成，一个数据访问代理数据库由一个或多个底层 RDS/OceanBase 提供的数据库组成。

针对单个数据访问代理实例，数据库管理包括创建数据库、查看数据库详情、配置读写分离、数据表管理、分库管理、白名单设置、数据导入、只读实例管理、平滑扩容、查看监控信息等功能。

5.2. 创建数据库

在数据库使用上，数据访问代理提供与 MySQL 相同的使用体验，但是数据访问代理创建数据库的流程有所不同，需要在控制台上完成，具体参见 创建数据库。

5.3. 查看数据库详情

您可以通过数据访问代理控制台查看数据库具体信息。

操作步骤

1. 进入数据访问代理控制台，单击左侧导航栏上的 **数据库**，进入数据库列表页面，可查看数据库概览信息。
2. 在数据库列表中，选择需要查看的数据库，单击数据库名称进入数据库详细信息页面。

在此页面可以查看数据库基本信息、访问信息、数据表信息、物理数据源和连接参数信息。

其中，访问信息中的命令行连接地址是指登录到数据库的连接地址。由于数据访问代理完全兼容 MySQL 协议，因此通过 MySQL 客户端可以操作数据访问代理数据库。将 **客户端命令行** 复制粘贴到操作系统终端中，输入密码即可登录使用数据访问代理数据库。

> **说明**
>
> - 一些旧版 MySQL 客户端对用户名长度有限制，不能大于 16 个字符。数据访问代理使用的库名和用户名相同，建库时库名超过 16 个字符，则会报错。
> - 如果使用 MySQL 客户端，在使用 HINT 时，务必在命令行中添加 `-c` 参数。数据访问代理的 HINT 是通过注释实现（类似 `/*...*/`）。如果没有加 `-c` 参数，会丢失注释，从而导致数据访问代理的 HINT 丢失。

5.4. 数据库上下线与配置生效

数据访问代理提供了数据库上线与下线的切换能力。当数据库状态为上线时，当前数据库可以被正常使用；当数据库状态切换为下线时，则当前数据库无法被应用连接使用。在一些应用变更或升级的场景下，旧的数据库不再被使用，但是考虑到新版本可能会带来的问题，旧版本数据库依旧需要保留一段时间，以便在紧急回滚时可以开启继续使用。数据访问代理的上/下线切换特性方便您在紧急回滚时快速开启旧版数据库以供使用。

数据库上线与下线

如上图所示，当前数据库状态为 **已启用**，单击 **下线** 按钮可以将数据库状态切换为 **未启用**。

数据库配置生效

当前数据访问代理提供了简单的数据库版本管理功能。对于在指定数据库中所做的修改，包括参数变更、数据表的增删改，点击数据库详情页面的 **配置生效** 按钮后，这些变更才会真正生效。这一特性主要用于那些需要对数据进行多处变更操作，且这些操作需要同时生效，否则会影响到业务使用的场景。

生效时间 指示上一次配置生效的时间点，在此之后所做的变更则需要单击 **配置生效** 按钮方可真正生效，被客户端识别。

5.5. 查看分库信息

您可以通过数据访问代理控制台查看数据库的分库分表信息以及后端各个物理分库名与所在的物理数据源（RDS 或 OceanBase）。

操作步骤如下：

1. 进入数据访问代理控制台，单击左侧导航栏上的 **数据库** 后，可在右侧的数据库列表页面查看数据库相关信息。
2. 在数据库列表中，单击指定的数据库进入数据库详细信息页，然后单击页面下方的 **物理数据源** 标签，可查看当前数据库所对应的所有分库信息，以及各个分库所属的物理节点（RDS 或 OceanBase）ID。

5.6. 管理数据表

您可以通过数据访问代理控制台进行数据表管理，包括创建数据表，查看、修改数据表设置。

创建数据表

- 通过控制台创建数据表

  具体操作，请参见 创建数据表。

- 通过命令行创建数据表
  - 单库单表支持 drop table 、 alter add 、alter drop、create 的 DDL 语法。

- 建表的 Sharding 语句如下：

Sharding 语句的详细说明，请参见 Sharding 功能介绍。

查看数据表设置

在数据表创建成功后，您可以通过数据访问代理控制台来查看数据表详情，具体步骤如下：

1. 进入数据访问代理控制台，单击左侧导航栏上的 **数据库** 后，在右侧的数据库列表中单击数据表所在的数据库进入数据库详细信息页，即可看到当前数据库中所有的数据表列表。
2. 单击指定的数据表名进入详情页面，该页面展示了当前数据表的路由规则，如下图所示。有关路由规则的具体参数说明，参见 创建数据表 中的路由规则部分。

修改数据表设置

您可以对数据表详情页中的路由规则进行修改，包括添加、修改、删除设置项。

操作步骤如下：

1. 在数据表详情页中，单击 **路由规则** 右侧的 **编辑** 按钮，进入路由规则编辑页面。
2. 根据需要添加新的设置项、修改或删除已有的设置项。
   - 添加设置项：在空白的路由字段输入框中输入新的路由字段，并选择相应的路由模式，然后单击 **添加** 按钮，此时新增的路由字段会被保存，同时出现一行空白项供您继续添加。
   - 修改设置项：选择要修改的设置项，直接在该设置项的路由字段输入框中输入需要的路由字段，并选择相应的路由模式即可。

- 删除设置项：选择要删除的设置项，单击该设置项右侧的 **删除** 按钮即可。

3. 单击 **保存** 按钮保存前面所做的编辑。若要放弃编辑，则单击 **取消** 即可。

分库分表规则说明

分表规则详细说明如下：

- **Hash 取模**：要求字段类型为数字，将数值对分库/分表数量进行取模，得到的值作为分库位/分表位。如分表规则为 Hash 取模，分表总数为 100，值为 230，则分表位计算结果为 `230 % 100 = 30` 。
- **字符串截断（MySQL 风格）**：要求字段类型为字符串，将值通过 MySQL 的 substr 函数进行截取，第一个参数为开始位置，从第 1 位开始计算，第二个参数为长度；截取之后的值转为数字类型，再对分库/分表数量进行取模，得到的值作为分库位/分表位。
  - 如分表规则为字符串截断（MySQL 风格），分表数量为 100，第一个参数设置为 1，第二个参数设置为 2，值为 12345，那么分表位计算结果为 `((int)substr('12345', 1, 2)) % 100 = ((int)'12') % 100 = 12 % 100 = 12` 。
  - 如分表规则为字符串截断（MySQL 风格），分表数量为 100，第一个参数设置为 -2，第二个参数设置为 2，值为 12345，那么分表位计算结果为 `((int)substr('12345', -2, 2)) % 100 = ((int)'45') % 100 = 45 % 100 = 45` 。
- **自身位**：要求字段类型为数字，将值直接作为分表位/分表位。
  - 如分表规则为自身位，值为 23，则分表位计算结果为 23。

    > **重要**
    > 在计算分库规则时，如果不填写分库规则，那么会生成一个默认的分库规则，其计算结果为 `分表规则的计算结果 / （分表数/分库数）` 。

  - 如分表规则计算结果为 20，分库数为 10，分表数为 100，那么相同规则下分库规则计算结果为 `20 / ( 100 / 10) = 2` 。

5.7. Sharding 功能介绍

Sharding 是把数据库横向扩展到多个物理节点上的一种有效的方式。如果将一个数据库当作一块大玻璃，将块玻璃打碎的过程就叫 Sharding（可以翻译为分片，也称为分库），每一小块都称为数据库的碎片（DatabaseShard）。

分库分表规则

- Hash 取模：将数值对分库/分表数量进行取模，得到的值作为分库位/分表位。如分表规则为 Hash 取模，分表总数为 100，值为 230，则分表位计算结果为：`230 % 100 = 30` 。字段类型为数字。
- 字符串截断（MySQL 风格）：将值通过 MySQL 的 substr 函数进行截取，第一个参数为开始位置，从第 1 位开始计算，第二个参数为长度。将截取之后的值转为数字类型，再对分库/分表数量进行取模，得到的值作为分库位/分表位。字段类型为字符串。
  - 如分表规则为字符串截断（MySQL 风格），分表数量为 100，第一个参数设置为 1，第二个参数设置为 2，值为 12345，则分表位计算结果为：`((int)substr('12345', 1, 2)) % 100 = ((int)'12') % 100 = 12 % 100 = 12` 。
  - 如分表规则为字符串截断（MySQL 风格），分表数量为 100，第一个参数设置为 -2，第二个参数设置为 2，值为 12345，则分表位计算结果为：`((int)substr('12345', -2, 2)) % 100 = ((int)'45') % 100 = 45 % 100 = 45` 。

- 自身位：将值直接作为分表位/分表位。字段类型为数字。
  - 如分表规则为自身位，值为 23，则分表位计算结果为 23。

    > **重要**
    >
    > 在计算分库规则时，如果不填写分库规则，则会生成一个默认的分库规则，其计算结果为：`分表规则的计算结果 / （分表数/分库数）`。

  - 如分表规则计算结果为 20，分库数为 10，分表数为 100，那么相同规则下分库规则计算结果为：`20 / ( 100 / 10) = 2`。

Sharding 语法和使用限制

下面简单介绍 Sharding 语句的通用语法，以及 Sharding 语句在单库单表和分库分表等不同情况下的使用限制。

语法

```
CREATE [TEMPORARY] TABLE tbl_name[(create_definition,...)][table_options] [dbshard by hash([id,str]) shards N] [tbshard by hash([id,str]) shards N]
```

使用限制

- 单库单表：在单库单表中执行 Sharding 语句创建分库分表时，会被拦截。

  > **说明**
  >
  > 单库单表中执行 Sharding 语句，不带 shards 参数时，创建出来的是单表，符合逻辑，因此不会被拦截。

- 分库分表

- 一个分库
  - 无论使用的是 tbshard 子句还是 dbshard 子句，带 shards 参数且 `shards=1` 时，均会出现“ERROR 7400 (HY000): create table failed, caused by: sharding rules”报错，创建失败。

    > **说明**
    >
    > 如果需要在一个分库条件下，创建分库不分表，可通过控制台手动创建。详情请参见 创建数据访问代理数据表。

  - 使用 tbshard 子句时，不带 shards 参数，将创建单表。

    ```
    create table xx (name varchar(20) not null) tbshard by hash (substr(name , -2, 2));
    ```

    代码说明：
    - 分表规则是字符串截取，具体规则请参见 分库分表规则。
    - 执行语句，成功创建单表。
  - 使用 tbshard 子句时，带 shards 参数且 shards=n（n大于1），将创建多个分表。

    ```
    create table xx (name varchar(20) not null) tbshard by hash (substr(name , -2, 2)) shards 4;
    ```

    代码说明：
    - 分表规则是字符串截取，具体规则请参见 分库分表规则。
    - 执行语句，成功创建多个分表。
  - 使用 dbshard 子句时，不带 shards 参数，将创建单表。

    ```
    CREATE TABLE  xxx ( id int(10) primary key , name varchar(30) not null ) dbshard by hash (id);
    ```

    代码说明：
    - 分表规则是 Hash 取模 id，具体规则请参见 分库分表规则。
    - 执行语句，成功创建单表。
  - 使用 dbshard 子句时，带 shards 参数且 shards=n（n大于1），将创建多个分表。

    ```
    CREATE TABLE  xxx ( id int(10) primary key , name varchar(30) not null ) dbshard by hash (id) shards 2;
    ```

    代码说明：
    - 分表规则是 Hash 取模 ID，具体规则请参见 分库分表规则。
    - 执行语句，成功创建多个分表。
- 多个分库（以两个分库为例）

- 使用 tbshard 子句时，不带 shards 参数，将创建单表。

```
create table xx (name varchar(20) not null) tbshard by hash (substr(name , -2, 2));
```

代码说明：

- 分表规则是字符串截取，具体规则请参见 分库分表规则。
- 执行语句，成功创建单表。

- 使用 tbshard 子句时，带 shards 参数且参数 shards 的值等于分库数，则成功创建分表。

```
create table xx (name varchar(20) not null) tbshard by hash (substr(name , -2, 2)) sh
ards 2;
```

代码说明：

- 分表规则是字符串截取，具体规则请参见 分库分表规则。
- 执行语句，成功创建分表。

- 使用 tbshard 子句时，带 shards 参数且参数 shards 的值为分库的整数倍，则成功创建整数倍分表。

```
create table xx (name varchar(20) not null) tbshard by hash (substr(name , -2, 2)) sh
ards 4;
```

代码说明：

- 分表规则是字符串截取，具体规则请参见 分库分表规则。
- 执行语句，成功创建整数倍分表。

- 使用 tbshard 子句时，带 shards 参数且参数 shards 的值不是分库的整数倍，则创建失败，语句执行报错。

```
create table xx (name varchar(20) not null) tbshard by hash (substr(name , -2, 2)) sh
ards 3;
```

代码说明：

- 分表规则是字符串截取，具体规则请参见 分库分表规则。
- 语句执行报错为：ERROR 1149 (HY000): Table shards illegal。

  如果执行 Sharding 语句时，shards 的数值不规范，会出现“ERROR 1149 (HY000): Table shards illegal”报错 。之后再执行其他命令时，会出现延迟。

  例如，执行 Sharding 语句报错后，执行命令 `show tables` 会出现“ERROR 7400 (HY000): create table failed, caused by: table count mod group count must be zero”报错。之后执行命令 `select 1` ，展示的却是 `show tables` 的结果。

  如果遇到了执行异常的情况，可断开 Server 后重新连接 Server。

- 使用 dbshard 子句时，不带 shards 参数，将创建单表。

```
CREATE TABLE  xxx ( id int(10) primary key , name varchar(30) not null ) dbshard by h
ash (id);
```

代码说明：

- 分表规则是 Hash 取模 ID，具体规则请参见 分库分表规则。
- 执行语句，成功创建多个分表。

- 使用 dbshard 子句时，带 shards 参数且参数 shards 的值等于分库数，则将创建分库不分表。

```
CREATE TABLE  xxx ( id int(10) primary key , name varchar(30) not null ) dbshard by h
ash (id) shards 2;
```

代码说明：

- 分表规则是 Hash 取模 ID，具体规则请参见 分库分表规则。
- 执行语句，成功创建分库不分表。

- 使用 dbshard 子句时，带 shards 参数且参数 shards 的值为分库的整数倍，则成功创建整数倍分表。

```
CREATE TABLE  xxx ( id int(10) primary key , name varchar(30) not null ) dbshard by h
ash (id) shards 6;
```

代码说明：

- 分表规则是 Hash 取模 ID，具体规则请参见 分库分表规则。
- 执行语句，成功创建整数倍分表。

- 使用 dbshard 子句时，带 shards 参数且参数 shards 的值不是分库的整数倍，则创建失败，语句执行报错。

```
CREATE TABLE  xxx ( id int(10) primary key , name varchar(30) not null ) dbshard by h
ash (id) shards 5;
```

代码说明：

- 分表规则是 Hash 取模 ID，具体规则请参见 分库分表规则。
- 语句执行报错为：ERROR 1149 (HY000): Table shards illegal。

注意事项

- 当两个实例下的逻辑库共用一个节点时，在控制台中可以删除逻辑库下面的逻辑表。如果一个实例进行了删除逻辑表的操作（同时删除物理库），另一个实例的逻辑表不会消失。但是在操作时，对于另一边的实例 来说，虽然逻辑表没有消失，但物理表已经被删除了，这个逻辑表也不可用。操作人员需要注意自己的操作行为。
- 使用 Sharding 语句中的 dbshard 子句，参数 shards 的值是分库数的整数倍时，创建出来的分表通过 `show topology from` 命令查询出来的信息和在物理库中的展示信息不统一：在物理库中的展示信息是分库分表；在 server 中通过命令查询出来的信息则是分库不分表。

5.8. 设置白名单

数据访问代理提供访问控制功能，只有 IP 在白名单中的客户端才可以访问。同时，数据访问代理允许您查看当前数据访问代理实例所在的 IP（或安全 IP 网段），方便您对自己所管理的后端数据节点实例（RDS 或 OceanBase）进行白名单的设置。

获取数据访问代理实例所在的 IP（或安全 IP 网段）

1. 进入数据访问代理控制台，单击左侧导航栏上的 **实例**。
2. 在右侧的实例列表中，单击指定的数据访问代理实例名，进入实例详情页面。
3. 单击 **白名单** 标签，再单击 **数据访问代理机器列表** 可以查看当前实例所在的 IP（或安全 IP 网段）。

添加客户端 IP 至数据访问代理白名单

在实例详情页面中（如上图所示），单击 **设置白名单** 标签，即可为当前数据访问代理设置（或更新）受信客户端的 IP 或 IP 网段信息，保证访问安全。

IP 地址支持全 IP、 CIDR 模式和 `*` 号占位符模式。当配置为 `*.*.*.*` 或空时，表示不设置 IP 访问限制。

5.9. 管理物理数据库连接参数

应用数据源配置包含了数据库所有连接参数的管理，在 **连接参数** 标签中，主要有两个部分：

默认连接数

1. 勾选需要调整连接数的物理数据库名。
2. 在 **连接数** 输入框中输入最小、最大连接数。
3. 按回车键预览效果，调整至合适的效果后，单击页面底部的 **保存配置**。

其他参数

数据访问代理可以通过调整参数来控制对物理数据库的操作，配置如下图：

参数说明：

- **connectionProperties**：MySQL 中的参数。
- **blockingTimeoutMillis**：获取连接的最大超时时间。
- **idleTimeoutMinutes**：连接空闲时间，超过该时间会被剔除出连接池。
- **prefill**：是否预热最小连接数。

注意事项

以上所有操作执行完毕后，您需要点击页面顶部的 **配置生效** 按钮，数据库访问代理实例会在 3 分钟内生效配置。

5.10. 导入导出数据库

为了能在不同环境中快速复制、迁移或备份数据，数据访问代理提供数据库导出、导入功能，并可通过导入数据库来更新数据表信息和连接参数。

导出数据库

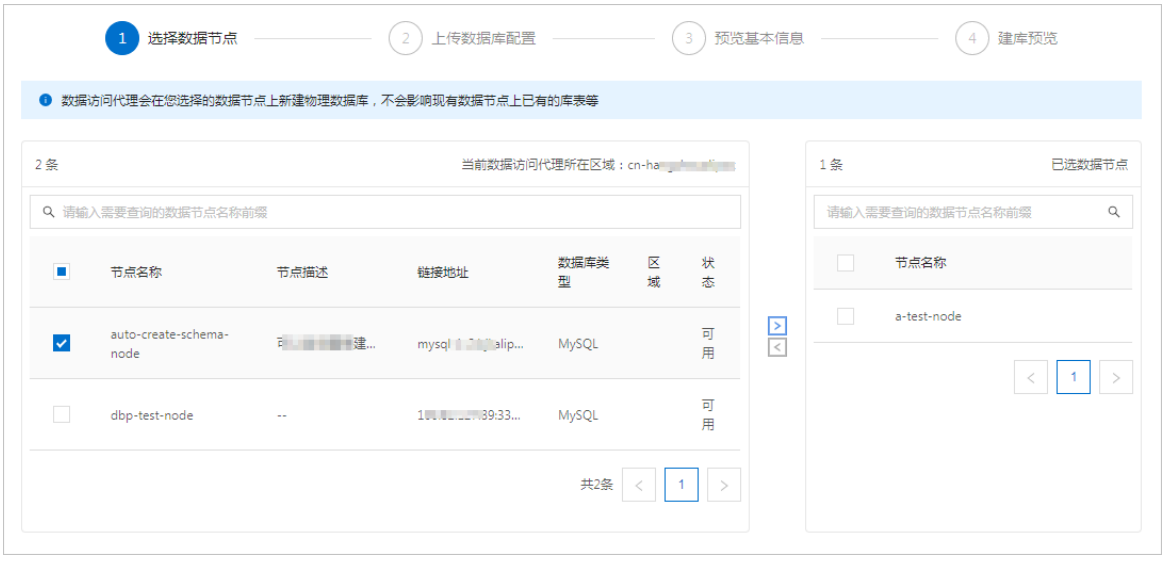
1. 进入数据库详情页，有以下两种方式：
   - 数据访问代理 > **实例** > 选取需要导出的数据库所在的实例 > 单击数据库名称，进入数据库详情页。
   - 数据访问代理 > **数据库** > 单击数据库名称，进入数据库详情页。
2. 单击页面右上方的 **导出** 按钮。

3. 单击 **确定**。
4. 数据库信息文件将以 `.zip` 格式下载至本地。

导入数据库

> **说明：**
> - 导出的数据库文件（.zip），可以在不同的实例下导入。
> - 同一实例下不能导入相同名称的数据库文件（.zip）。
> - 导出的数据库文件（.zip），解压修改文件中的数据库名后，可再压缩导入到同一个实例下。

1. 进入 **导入数据库** 向导页：
   - 数据访问代理 > **实例** > 选取需要导入数据库的实例 > **导入数据库**；
   - 数据访问代理 > **数据库** > **导入数据库** > 选择需要导入数据库的实例 > **导入**。
2. **导入数据库** 向导页共有四个步骤：
   i. 选择数据节点。勾选需要导入的数据节点，单击 **添加** 按钮，并单击 **下一步**。



ii. 上传数据库配置。单击 **上传文件**，选取数据访问代理导出到本地的数据库文件（.zip），并单击 **下一步**。

iii. 预览基本信息。数据访问代理会自动分析上传的配置文件，读取相应的数据库信息，您无法对各参数进行修改，单击 **下一步**。

iv. 建库预览。在此步骤中，您可以选择 **是否自动创建物理数据库**。

- 如果选择 **是**，则数据访问代理将为您自动创建物理数据库。您需要输入各数据节点的高权限账户用户名与密码。若各节点的高权账户相同，可以勾选 **为所有数据节点使用相同的账户**，那么只需要输入一次高权限用户名与密码。
- 如果选择 **否**，表示物理数据库由您手动创建。勾选 **为所有数据节点使用相同的账户**，即可输入一次物理数据库的账户用户名及密码。

v. 单击 **完成**，开始导入数据。

3. 数据库导入完成后，页面跳转至数据库列表，您可以看到新导入的数据库。

5.11. 小表广播

业务中存在一些配置表，存储重要的配置，读多写少。在实际业务查询中，很多业务表会和配置表进行联合数据查询。但在数据库水平拆分后，配置表是无法拆分的。因此，数据访问代理提供了小表广播功能，支持配置表同步至目标数据库的所有分库。

配置表，即广播表，是在每个物理库中都存在的一个单表，没有后缀。广播表的更新会在物理库主库操作，然后同步到所有分库。

广播任务列表

在 **实例详情 > 小表广播** 页面，您可以查看当前实例下所有的小表广播任务及其概览信息。

- **表名**：待同步配置表的名称。
- **源数据库**：待同步表所在的逻辑数据库。
- **目标数据库**：待同步表的目标端数据库。
- **任务状态**：广播任务的执行状态，包括 **创建中**、**广播表创建失败**、**运行中**、**运行异常** 与 **已废弃**。
- **创建时间**：广播任务创建的时间。
- **修改时间**：最近一次操作的时间。
- **操作**：支持 **废弃**、**重试建表** 与 **刷新状态**。

如当前实例的广播表任务过多，您可以通过在列表右上角的搜索框，通过搜索 **表名**、**源库** 或 **目标库**（支持模糊搜索），快速找到想要的广播任务。

广播任务操作

对于不同状态的广播任务，您可以进行相应的人工干预操作。

操作	说明	适用状态
废弃	废弃当前的广播任务，直接终止该任务	• 广播表创建失败 • 运行中 • 运行异常
重试建表	进入建表 DDL 任务的详情页，查看具体任务详情，可进行重试操作	• 广播表创建失败
刷新状态	获取广播任务的最新运行状态	• 运行中 • 运行异常

创建小表广播

1. 进入数据访问代理控制台页面，在左侧导航栏中，选择 **实例**。
2. 在实例列表中，选择想要配置广播表的实例，单击其实例名，进入详情页。
3. 选择 **小表广播** 页签，单击 **创建广播任务**。
4. 在创建页面，选择或输入以下信息：

- **源库**：选择当前实例下的数据库作为源端数据库。
- **待同步表名**：输入源库中要作为广播表的的表名。单击 **+添加**，可添加更多待同步表。至少需要有一个待同步表。
- **目标库**：选择当前实例下的数据库作为目标数据库。

> **警告**
>
> 待同步表必须是源库中已经存在的表。 待同步表必须是一个单表，不支持拆分表。 待同步表必须是没有正在运行的广播任务，已创建广播任务的表（不含废弃的表）不可再次作为同步源。 待同步的目标数据库中不能已经存在同名表。 目前不支持广播表的结构变更的同步，如后续有结构变更，需手动同步目标表。 目前仅支持 RDS 数据库。

5. 单击 **创建**，即可返回小表广播列表页，显示该广播任务正在创建中。

5.12. SQL 拦截

数据访问代理提供了 SQL 拦截功能，可以防止业务危险的 SQL 导致数据被误删除或者无索引的查询使数据库和数据访问代理性能变差，导致业务请求不流畅等问题。

操作步骤

1. 进入数据访问代理控制台页面，左侧导航栏选择 **实例**。
2. 在实例列表中，找到您想要进行 SQL 拦截的数据库所在的实例，单击其名称，进入详情页。
3. 在该实例的详情页中，切换至 **SQL 拦截** 页签，单击 **创建拦截规则**。

4. 在新弹出窗口中，配置以下信息：
   - 数据库名称：从下拉菜单中选择要进行 SQL 拦截的目标数据库。
   - 名称：输入拦截规则的名称，仅支持数字、字母、下划线、中划线、64 个字符以内。
5. 单击 **确定**，完成拦截规则创建。
6. 返回 SQL 拦截规则列表，找到刚刚创建的拦截规则名称，单击 **编辑**。
7. 在新页面中，单击 **添加**，开始添加需要拦截的 SQL 模板。目前仅支持基于 SQL 模板的拦截。

8. 规则添加完成后，单击 **保存**。

9. 拦截规则保存后，您可以尝试执行符合上述添加的 SQL 模板的语句，该语句将被拦截，如下图所示。

```
mysql> select * from user where user_id = 123456789;
Empty set (0.02 sec)

mysql> select * from user where age > 10;
ERROR 7204 (HY000):  select * from user where age > 10  was intercepted
mysql>
```

6.物理数据节点

6.1. 物理节点管理概述

数据访问代理实例底层存储是由一个或者多个 RDS/OceanBase 实例节点组成，其管理的数据库也是由这些底层节点提供的数据库组成。因此，在使用数据访问代理进行数据库的管理之前，您需要通过数据访问代理控制台添加已购买的数据节点，这些数据节点可以是 RDS 或 OceanBase 实例，使数据访问代理可以接入并管控这些数据节点。

当数据访问代理接入数据节点后，您可以通过数据访问代理控制台直观地看到这些数据节点的运行状态，并且可以通过创建逻辑库的建库规则，灵活地创建物理数据库。也可以通过表 DDL 来创建、删除、修改数据表。

6.2. 查看物理数据节点

您可以通过数据访问代理控制台查看数据访问代理的数据节点相关信息。

物理数据节点列表

进入数据访问代理控制台，单击左侧导航栏上的 **运维** > **物理数据节点** 后，可在右侧的物理数据节点页面查看当前系统中所有的物理数据节点。

物理数据节点详情

单击指定的数据节点 ID，进入该数据节点的详情页面，可以看到该节点的基本信息以及在该节点上创建的物理数据库（分库）。

6.3. 添加与管理物理数据节点

您可以通过数据访问代理控制台管理物理数据节点（RDS 或 OceanBase）。

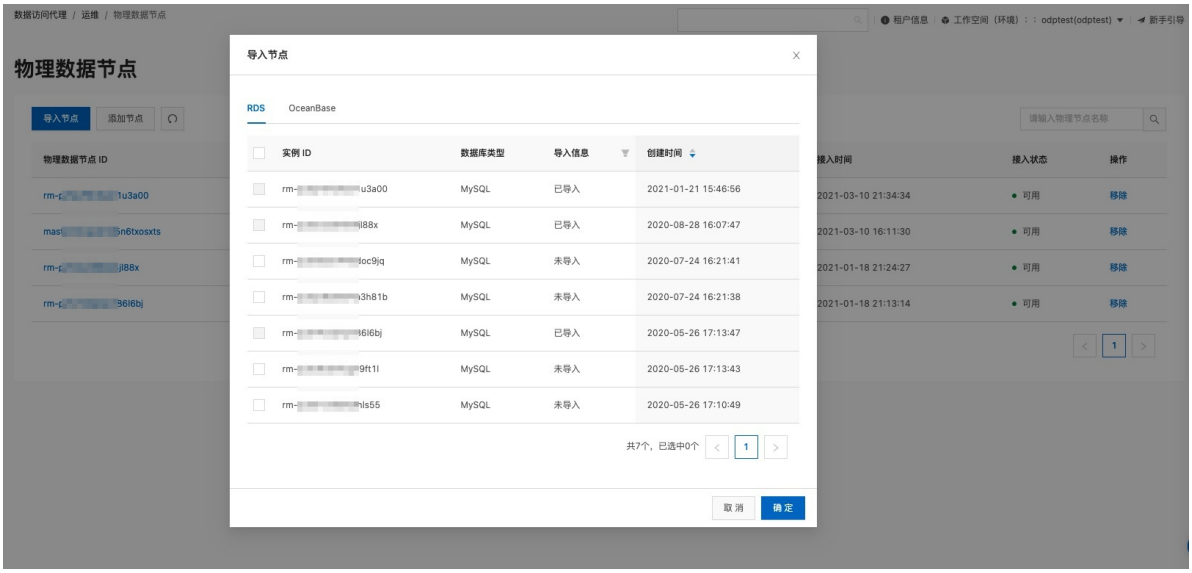
添加节点

在数据访问代理控制台，您可以添加已购买的物理数据节点（RDS 或 OceanBase）进行管理，具体操作参见添加物理数据节点。

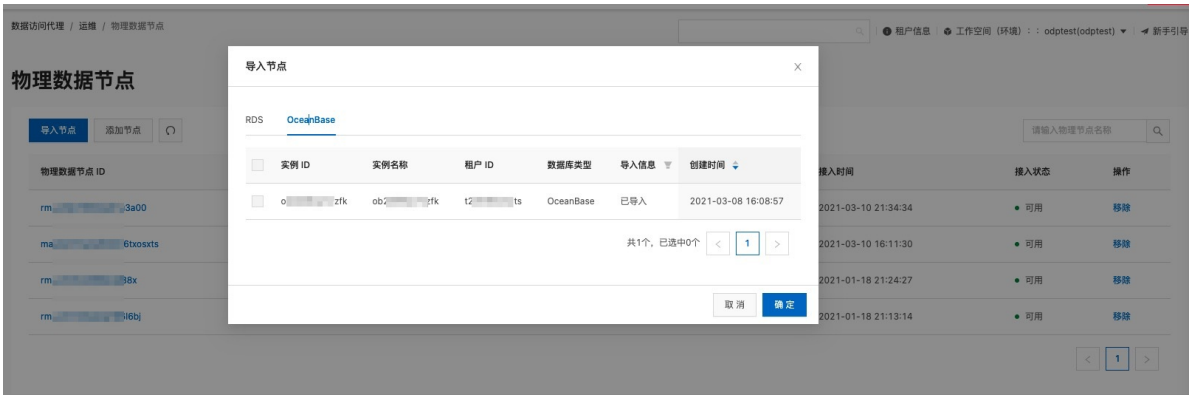
导入节点

数据访问代理不仅支持只能支持手动创建数据节点，还支持自动导入 RDS 实例。操作步骤如下：

1. 在数据访问代理控制台左侧导航栏中，选择 **运维 > 物理数据节点 > 导入节点**。
2. 在弹出的对话框中，选择节点类型（RDS 或 OceanBase）。



3. 在已获取的实例列表中，选择目标实例，并单击 **确定**。



> **说明**
>
> 导入 OceanBase 数据库实例时，账号需要被授 AliyunSofaOdpDefaultRole 角色权限，授权步骤参见资源授权。

导入后，数据访问代理会自动创建物理数据节点，并在对应的 RDS 或 OceanBase 实例中加入数据访问代理实例的白名单。

修改节点信息

您可以通过控制台来修改已添加的物理数据节点的部分信息。目前可以更新的物理数据节点信息包括：

- **节点描述**：增加或修改当前物理数据节点的描述。
- **用户名/密码**：如果接入的物理数据节点的高权访问账户被修改（例如，管理员通过 RDS 控制台修改了 RDS 节点的高权限账号密码），则需要将更改后的账户信息同步到数据访问代理。错误的用户名/密码会导致数据访问代理无法正常接入物理数据节点，也无法进行物理分库/分表创建等操作。您可以前往 **数据库 > 数据库详情页 > 物理数据源页签**，单击 **设置物理数据库账户** 修改账户信息。

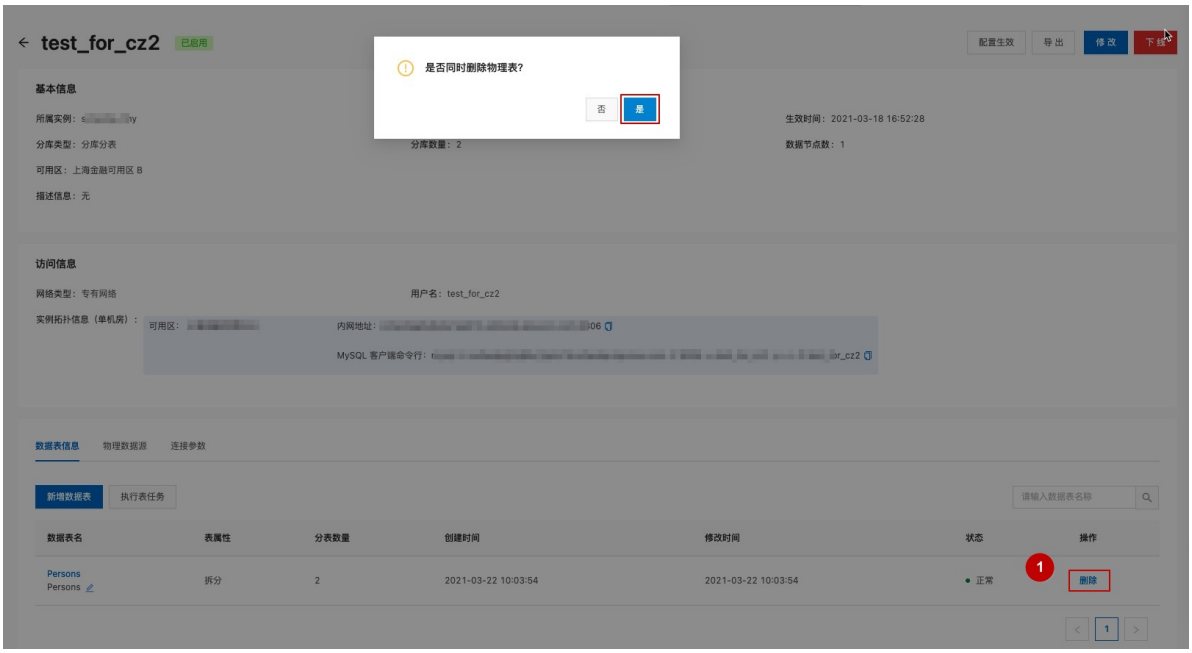
删除节点

您可以通过控制台来移除已添加的物理数据节点。移除后，数据访问代理无法继续管理该物理数据节点，但是该数据节点依然存在，且您可以在其对应产品的控制台上继续进行访问和控制，比如 RDS 控制台。

在移除物理数据节点时，数据访问代理会进行相应的检测，如果依然有（逻辑）数据库/数据表与当前物理数据节点存在关联，会生成一个删除对应物理数据表/数据库的任务。

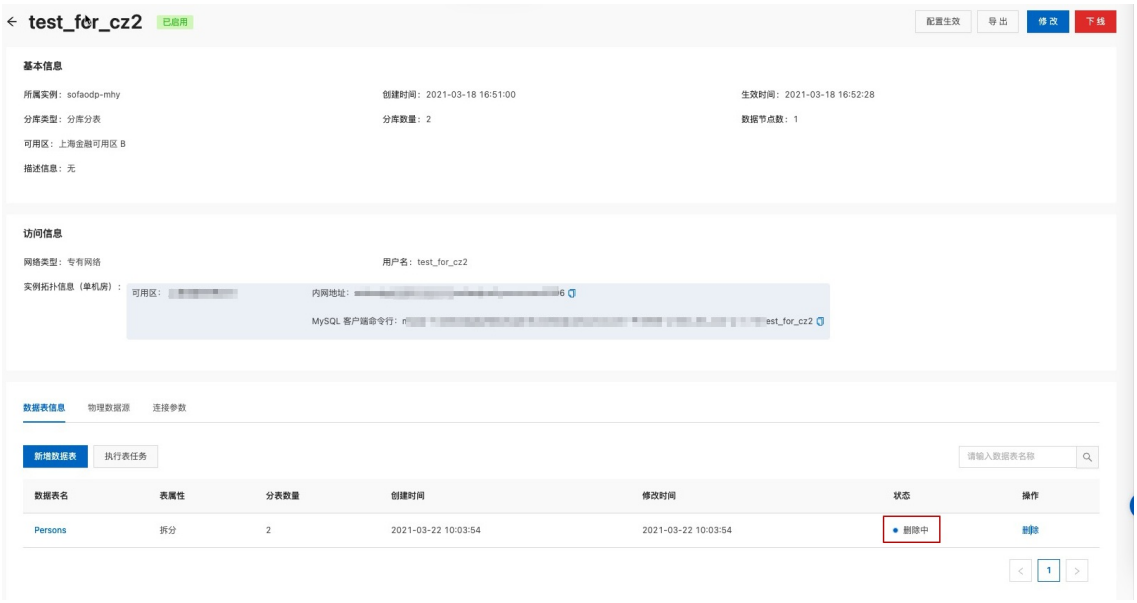
1. 在 **数据访问代理 > 数据库** 页面，单击 **数据表信息** 页签，然后单击操作列的 **删除**。

   弹出 **是否同时删除物理表？** 选择框。

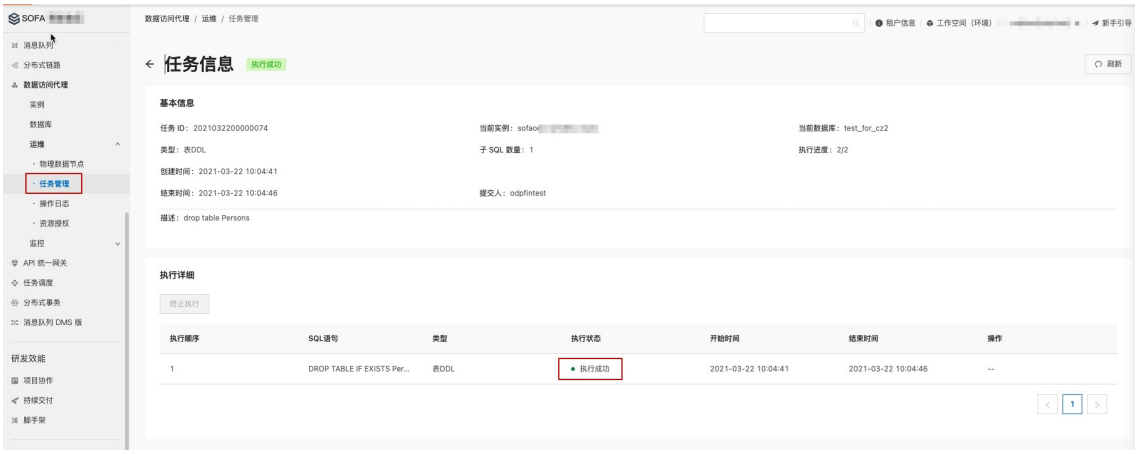


2. 选择是否同时删除物理表。

- 单击 **否**，只删除数据表，不删除对应的物理表。
- 单击 **是**。
  - 表状态变成 **删除中**。



- 在 **运维 > 任务管理** 下新增一条删除对应物理数据表的任务。



7.任务管理

7.1. 任务管理概述

数据访问代理会通过触发 DDL 任务来进行数据库与数据表的管理操作，比如创建、删除数据库，创建、删除数据表等。这些 DDL 任务既可以在创建物理分库前由数据访问代理自动触发，也可以由您在创建数据表时通过传入 SQL 文件或者直接输入 DDL 语句来执行。

在金融级的数据库设计场景中，可能会存在大量的数据库分库，执行 DDL 操作意味着会在多个数据节点以及大量的物理数据库中执行任务，不仅耗时较长，还可能会因为硬件故障等因素导致执行失败，从而需要进行人工干预。

因此，数据访问代理提供一整套针对 DDL 任务的管理体系与界面，方便您直观地对执行过的 DDL 进行审计操作，您也可以在执行 DDL 任务时查看当前的执行进度，出现问题时快速定位到执行出错的位置与详细信息。

数据访问代理中的任务管理模块针对执行出错的 DDL 任务提供出错处理机制。您可以根据实际情况选择重试或者跳过出错的 SQL 语句继续执行后续的操作。

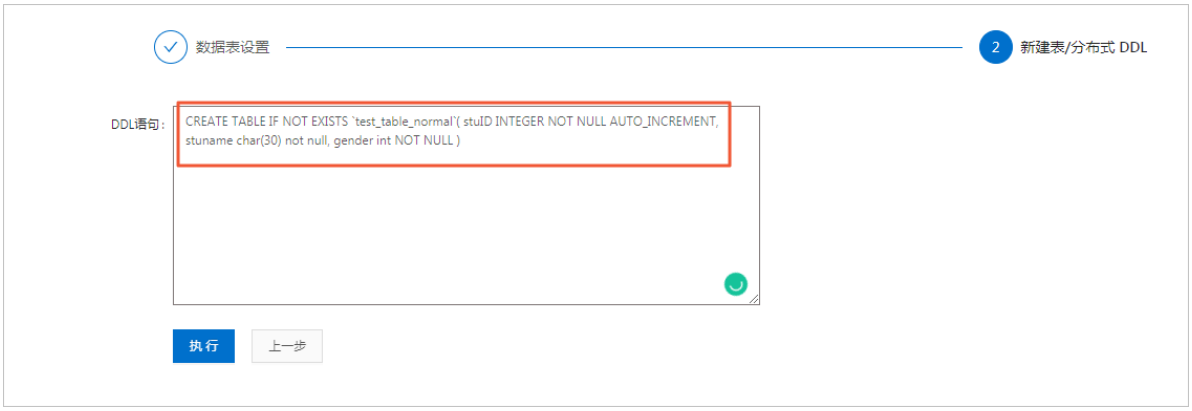
7.2. 创建 DDL 任务

DDL 任务用来进行数据库和数据表的管理操作。您既可以通过 DDL 任务创建数据表，也可以直接指定数据库执行 DDL 任务来进行物理数据表的变更。

通过 DDL 任务创建数据表

操作步骤如下：

1. 进入数据访问代理控制台，单击左侧导航栏上的 **数据库**，查看数据库列表。
2. 在列表中，单击指定的数据库进入数据库详情页面。
3. 单击 **新增数据表** 标签，进入 **创建数据表** 页面。
4. 选择 **上传 SQL 语句** 或 **输入 DDL 语句** 模式，单击 **执行** 触发 DDL 任务。

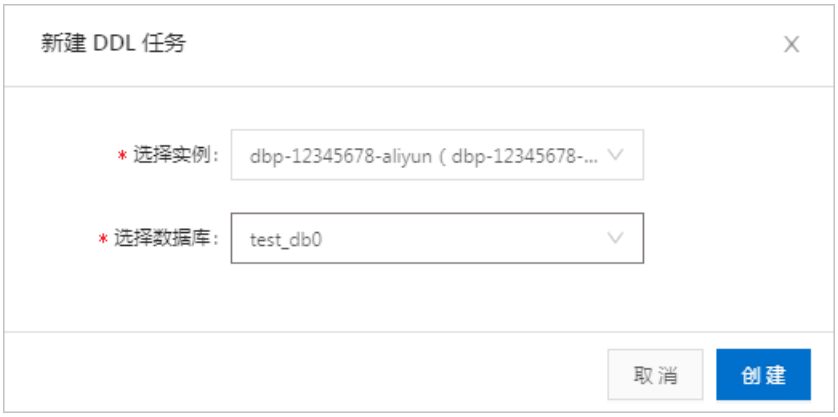


指定数据库执行 DDL 任务进行物理数据表的变更

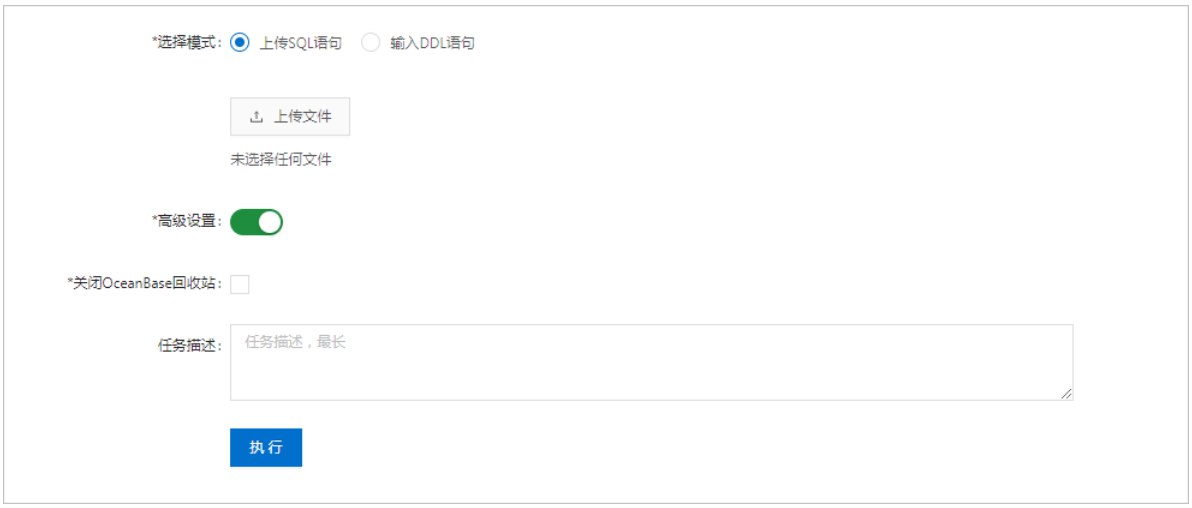
操作步骤如下：

1. 进入数据访问代理控制台，单击左侧导航栏上的 **运维** > **任务管理**。
2. 在任务管理页面，单击 **新建 DDL 任务**。

3. 选择要执行 DDL 的数据访问代理实例与数据库，单击 **创建**。



4. 选择 **上传 SQL 语句** 或 **输入 DDL 语句** 模式后，单击 **执行** 触发 DDL 任务。



7.3. 管理 DDL 任务

通过数据访问代理中的任务管理模块，能直观地对执行过的 DDL 任务进行审计操作，在执行 DDL 任务时查看当前的执行进度，并在出现问题时快速定位到执行出错的位置与详细信息。

任务管理模块针对执行出错的 DDL 任务提供出错处理机制。您可以根据实际情况选择重试或者跳过出错的 SQL 语句继续执行后续的操作。

查看任务

在数据访问代理控制台中，单击左侧导航栏上的 **运维** > **任务管理** > **DDL 任务**，进入 DDL 任务列表。

任务类型

- 待完成任务（正在执行的 DDL 任务）
- 已完成任务（已经执行完成的 DDL 任务）

待完成任务

待完成任务包括以下状态的任务：

- **执行中**：该条 SQL 正在指定的所有表库中执行。
- **队列中**：任务刚被提交，或在处于错误等待状态后被触发继续执行。任务需要排队等待被任务管理系统进行调度执行。
- **错误等待**：该条 SQL 在执行中出现错误，等待用户选择操作。

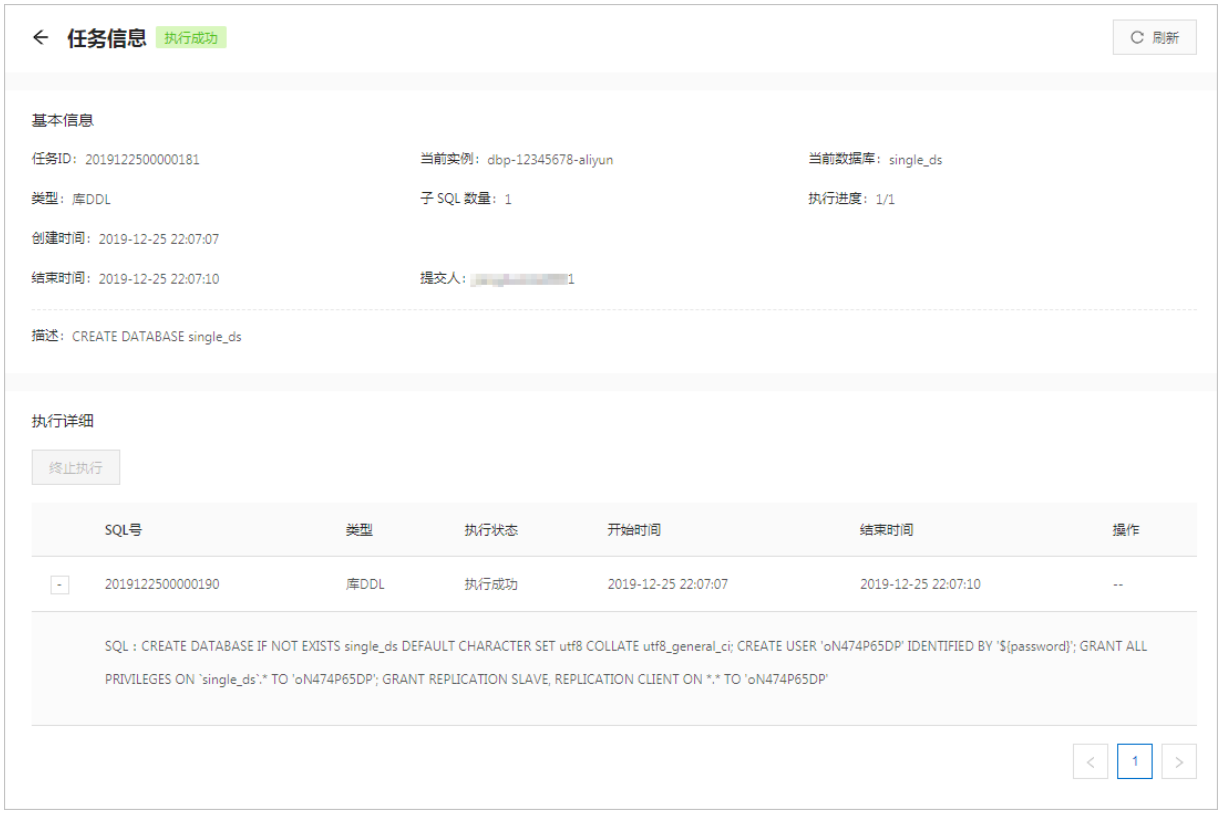
已完成任务

已完成任务包括以下状态的任务：

- **完成**：该条 SQL 在指定的所有库表中执行完毕（含出错跳过）。
- **终断**：该条 SQL 被用户终断执行。

任务详情

点击任务列表中的任务 ID 进入任务详情页面，查看该任务的详细信息与执行状态，包括当前所属的数据访问代理实例、执行表 DDL 所在的数据库以及 DDL 任务执行的详细进度，如下图所示：



处理执行出错的任务

对于运行时出现错误的 DDL 任务，任务详情中会高亮显示当前出错的信息，同时 DDL 任务会暂停执行。您可以定位到出错的位置，并查看当前 SQL 语句以及出错的原因以及所在位置（物理节点、分库、分表等信息）。

对于执行出错的 DDL 任务，提供 **重试** 和 **跳过** 两个选项。

- **重试**：对于某些因配置导致的错误，可以使用管理员账号在后台对数据节点（或物理数据库）进行变更，然后选择 **重试**，则当前 SQL 会被重新调度执行。
- **跳过**：选择 **跳过**，则当前 SQL 会被跳过，错误也会被忽略，下一条语句将被执行。

单击 **终止执行**，则当前整个 DDL 任务会被直接终止，不再继续执行。

7.4. DDL 语法

在使用数据访问代理时，您可以通过 DDL 语句完成建库建表等操作。

- DDL 基本语句
- Sharding DDL 语句

DDL 基本语句

- **创建数据表**

```
CREATE TABLE IF NOT EXISTS `table_name`(
    id   INTEGER NOT NULL PRIMARY KEY,
    name  VARCHAR(20) NOT NULL,
    age   INTEGER NOT NULL
);
```

- **删除数据表**

```
  DROP TABLE table_name;
```

- **修改数据表**

```
/* 修改字段名 */
  ALTER TABLE table_name CHANGE name name1 VARCHAR(255);

/* 修改字段类型 */
  ALTER TABLE table_name MODIFY name1 VARCHAR(64);

/* 添加字段（默认添加至数据表末尾）*/
  ALTER TABLE table_name ADD COLUMN address VARCHAR(20) DEFAULT NULL;

/* 添加主键 */
  ALTER TABLE table_name ADD PRIMARY KEY (id);

/* 添加唯一约束 */
  ALTER TABLE table_name ADD UNIQUE uni_name1 (name1);
```

- **添加索引**

```
/* 创建索引 */
  CREATE INDEX idx_addr ON table_name (address);

/* 使用 ALTER TABLE 语句添加索引 */
  ALTER TABLE table_name ADD INDEX idx_addr (address);

/* 创建唯一索引 */
  CREATE UNIQUE INDEX uni_addr ON table_name (address);
```

- **删除索引**

```
/* 删除索引 */
  DROP INDEX idx_addr ON table_name;

/* 使用 ALTER TABLE 语句删除索引 */
  ALTER TABLE table_name DROP INDEX idx_addr;
```

使用限制

当您在数据访问代理中使用 DDL 语句时，请注意以下限制：

- CREATE TABLE 语句不支持 FOREIGN KEY 约束。
- COMMENT 关键字不支持分号（;）。

Sharding DDL 语句

数据库访问代理在控制台的任务管理与服务端上均支持了 Sharding DDL 功能，即带有拆分规则的建库建表语句。

创建数据库

- 新建库：

```
CREATE DATABASE IF NOT EXISTS my_db  -- MYSQL 原生
default character set utf8  -- MYSQL 原生
[SHARDS 100]--扩展：【通用】分库数属性
```

- 复用已有库：

```
CREATE DATABASE tradequery_db EXTENDS trade_db;
```

创建数据表

- 当需要创建的表数量较少时，您可以在控制台页面中填写表名、分表数量、选择分库分表拆分键和规则，再输入建表语句完成逻辑表和物理表的创建。
- 当需要创建的表数量较多时，您可以通过下面的 SQL 语句完成逻辑表和物理表的创建，提高建表的效率。

```
CREATE TABLE [IF NOT EXISTS] `table_name`(
    id int(11) not null,
    user_id varchar(32) not null,
    age int(11) not null
)
DBSHARD BY HASH (col)
TBSHARD BY HASH (dol)
SHARDS NUMS  --分表数量
```

例如：

```
CREATE TABLE IF NOT EXISTS `user`(
    id int(11) not null,
    user_id varchar(32) not null,
    age int(11) not null
)
[ DBSHARD BY HASH (user_id) ] --可选，根据字段 user_id 进行分库
TBSHARD BY HASH (user_id)  --根据字段 user_id 进行分表
SHARDS 100  --分 100 个表
```

> **说明**
>
> 如果表均匀分布在物理库上，DBSHARD 可以省略，DBP 会自动根据 TBSHARD 计算出分库规则。

单拆分列

- **字符串截断**：字符串截断是数据访问代理应用最多的拆分规则，可以配合 `substr(str,pos,len)` 函数一起使用。

```
CREATE TABLE [IF NOT EXISTS] `table_name`(
id int(11) not null,
user_id varchar(32) not null,
age int(11) not null
)
[ DBSHARD BY HASH (substr(user_id,-4,2)) ] --可选
TBSHARD BY HASH (substr(user_id,-4,2))
SHARDS 100
```

- **HASH**

```
CREATE TABLE [IF NOT EXISTS] `table_name`(
id int(11) not null,
user_id varchar(32) not null,
age int(11) not null
)
DBSHARD BY HASH (id)
TBSHARD BY HASH (id)
SHARDS 100
```

> 说明
>
> - 使用 HASH 函数拆分，拆分键必须为整数类型。
> - 拆分列必须要包含在建表语句中。

- **TOINT**

```
CREATE TABLE [IF NOT EXISTS] `table_name`(
id int(11) not null,
user_id varchar(32) not null,
age int(11) not null
)
DBSHARD BY TOINT(id)
TBSHARD BY TOINT(id)
SHARDS 100
```

7.5. SELECT 语法

在使用数据访问代理时，您可以通过 SELECT 语句从一张或多张表中查询数据。

基本语法

```
[HINT]SELECT
[ALL]
    select_expr [, select_expr ...]
[FROM table_references
[WHERE where_condition]
[GROUP BY {col_name | expr}
[ORDER BY {col_name | expr}
[ASC | DESC],...]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
[FOR UPDATE]
```

语法说明：

- `HINT`：表示数据访问代理自定义的 HINT 语法，可以用 HINT 指定具体的分库分表查询。更多有关 HINT 的信息，参见 自定义 HINT。
- `select_expr`：表示 SELECT 语句要查询的列或者聚合函数表达式，可以使用 `*`。
- `table_references`：表示从哪些表中查询数据，支持单表及多张表。

- `WHERE`：表示查询条件，从表中获取满足 `where_condition` 的行。若没有条件，需要使用 HINT 指定具体的分库分表或全表扫描，否则 SQL 会报错。
- `GROUP BY`：表示聚合条件，可以是具体的列名或者表达式，暂不支持 HAVING 子句。
- `ORDER BY`：表示排序条件，可以是具体的列名或者表达式，可以自定升序 `ASC`（默认）或者降序 `DESC`。
- `LIMIT`：表示输出的 OFFSET 和行数。如果没有指定 OFFSET，则表示从 0 行开始输出，输出 `row_count` 数量的行。
- `FOR UPDATE`：表示对查询的行加上排他锁，阻止并发修改，或阻止在某些事务隔离级别时的并发读取。在全表扫描下，`FOR UPDATE` 暂不支持全局排他锁。

使用限制

在数据访问代理中使用 SELECT 语句进行数据查询时，您需要注意以下使用限制。

普通查询

- 分库分表查询条件要带上拆分键，否则要使用 HINT 指定分库分表或全表扫描。
- 带拆分键查询仅支持 `=` 和 `in`，不支持范围查询。
- `LIMIT` 若有两个参数，第一个参数表示返回第一行的偏移量，第二个参数表示返回的行数。若仅有一个参数，则表示返回的行数，默认偏移量为 0。

聚合查询

- 全表的聚合查询，需要显示使用 HINT，例如 `/*+DBP: $ROUTE={SCAN_ALL()}*/ SQL`。
- 数据访问代理目前支持 `COUNT`、`SUM`、`AVG`、`MAX`、`MIN` 五种聚合函数。暂不支持聚合函数中使用表达式，其中 `AVG` 仅支持数值类型。
- 同时使用 `GROUP BY` 和 `ORDER BY` 时，需尽量保证 `ORDER BY` 后面的表达式或者字段与 `GROUP BY` 保持一致，否则会占用较多内存计算，影响性能。例如，在下面的语句中，`ORDER BY` 后面的字段与 `GROUP BY` 不一致，容易影响性能。

```
SELECT * FROM trade_order GROUP BY trade_no ORDER BY trade_time
```

JOIN 查询

- 全表的 JOIN 查询需要显示使用 HINT，例如 `/*+DBP: $ROUTE={SCAN_ALL()}*/ SQL`。
- 数据访问代理目前支持业务表和 广播表 的聚合查询，例如：

```
SELECT * FROM trade_order o JOIN config c on o.org_id = c.org_id WHERE order_id IN (1,2)
```

- 数据访问代理仅支持拆分维度相同的业务表联合查询。相同拆分维度指的是表拆分键相同、所属同一个逻辑库并且数量相同。例如，在下面的语句中，`trade_order` 和 `order_item` 这两张表的拆分维度必须相同。

```
SELECT * FROM trade_order o JOIN order_item i ON o.order_id=i.order_id  WHERE order_id IN (1,2);
```

子查询

仅单库单表类型数据库支持子查询。

8.数据访问代理连接器

8.1. 数据访问代理连接器概述

数据访问代理连接器基于标准的 JDBC 接口实现，能代理标准的数据库连接池（如 DBCP、Druid、c3p0），对数据库进行增删查改。您可以使用该连接器连接数据访问代理服务器，执行数据库操作。

数据访问代理连接器主要支持以下功能：

- 数据库密码加密解密
- SQL 链路追踪
- 分库分表路由指定
- 拦截器扩展

数据库密码加解密

数据库密码为敏感信息，建议在配置文件中使用加密后的密文以提高安全性。数据访问代理连接器提供 `SecuritySpec` 接口来集成用户自定义的加解密算法 JAR 包，从而实现对自定义加解密算法的支持。

SQL 链路追踪

将 `traceId` 通过 MySQL 的自定义 HINT 语句传给数据访问代理服务器，从而打通应用到服务器的链路，输出 SQL 执行日志，轻松通过链路追踪应用，快速定位 SQL 执行耗时。

分库分表路由指定

可通过自定义 HINT 语句，指定 SQL 语句访问某一特定的分库分表。

拦截器扩展

数据访问代理连接器具有灵活的扩展机制，可通过实现拦截器 (Interceptor) 接口，提供更多的扩展功能。

8.2. 配置连接器

本文介绍如何在现有的应用工程中配置数据访问代理连接器。

操作步骤

> **注意**：下列各步骤中的配置仅供参考，您需要根据实际情况进行必要的修改。

1、在工程根目录的 `pom.xml` 文件中，根据需要添加以下 Maven 依赖：

```
<dependency>
      <groupId>com.alipay.sofa</groupId>
      <artifactId>dbp-connector-java</artifactId>
      <version>1.0.9</version>
</dependency>
```

2、在 Spring 配置文件中，增加连接器配置和数据源连接池 bean，如下所示：

```
<!-- 增加连接器配置 -->
 <bean id="simpleDataSourceConnector" class="com.alipay.sofa.dbp.DbpDataSource" init-method
="init">
    <property name="delegate" ref="simpleDataSource"/>
    <property name="dbpInstanceId" value="${yourODPInstanceId}"/>
    <property name="appName" value="${yourAppName}"/>
    <property name="database" value="${yourDatabaseName}"/>
 </bean>
 <!-- 连接器配置结束 -->
 <!-- 数据源连接池配置 -->
<bean id="simpleDataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="ini
t">
<property name="driverClassName" value="com.mysql.jdbc.Driver"/>
<!-- ODP实例地址，形如sofaodpojiss8o73o5la.sofaodp.aliyuncs.com      -->
<property name="url" value="jdbc:mysql://${yourODPConnectionURL}:8306/${yourDatabaseName}"/
>
<property name="username" value="${user}"/>
<property name="password" value="${password}"/>
</bean>
<!-- 数据源连接池配置结束 -->
```

密码解密

数据库密码是比较敏感的信息，在配置文件中推荐使用加密后的密文提高安全性；dbp-connector 中提供了 SecuritySpec 接口实现密码的加解密的功能，您需要实现这个接口自定义加解密的算法。

```
<!-- 增加连接器配置 -->
 <bean id="simpleDataSourceConnector" class="com.alipay.sofa.dbp.DbpDataSource" init-method
="init">
    <property name="delegate" ref="simpleDataSource"/>
    <property name="dbpInstanceId" value="${yourODPInstanceId}"/>
    <property name="appName" value="${yourAppName}"/>
    <property name="database" value="${yourDatabaseName}"/>
    <!-- 增加密码解密配置 -->
    <property name="securitySpec" ref="securitySpecImpl"/>
 </bean>
 <!-- 连接器配置结束 -->
 <!-- 数据源连接池配置 -->
<bean id="simpleDataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="ini
t">
      <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
      <property name="url" value="jdbc:mysql://${yourODPConnectionURL}:8306/${yourDatabaseN
ame}"/>
      <property name="username" value="${user}"/>
      <property name="password" value="${password}"/>
</bean>
<!-- 数据源连接池配置结束 -->

 <!-- 自定义密码解密实现，实现接口 com.alipay.sofa.extds.SecuritySpec-->
<bean id="securitySpecImpl" class="x.x.x.SecuritySpecImpl"/>
<!-- 密码解密结束-->
```

支持 SQL 链路追踪

dbp-connector 支持 SQL 链路追踪功能，dbp-connector 会将 traceId 通过 MySQL 的自定义 HINT 语法传给 dbpserver，从而打通应用到 dbpserver 的链路，方便通过链路追踪应用快速定位 SQL 执行耗时。

HINT语法格式如下：

```
/*+DBP: $SYS={TRACE(0a0fe91c151497435345910091964 9#0.1)}*/select * from test
```

HINT语法格式说明：

```
/*+DBP: $SYS={TRACE(TraceId#RpcId)}*/select * from test
```

dbp-connector 会将 SQL 执行信息包括 trace 信息打印在本地日志 `sql-digest.log` 中。默认情况下，执行时间小于 3ms 且执行成功的 SQL 按照 10/1 的比例抽样打印；执行失败和执行时间大于 3ms 的 SQL 全量打印。

日志格式如下：

```
2018-06-2023:42:10.280,testApp,0a41928115295093299891004690 09,0.1,testDbpInstanceId,dbpclie
nt_db,select*from mars,success,289ms,274ms,15ms,DBP,11.239.141.253:8306,main
```

日志格式说明：

```
日志打印时间,AppName,TraceId,RpcId,DBP,dbp实例id,schema,SQL,执行结果（success/failed),执行耗时（
ms）,链接建立时间,数据库执行时间,DBP,dbp实例Ip,当前线程名
```

```
<!-- 增加连接器配置 -->
 <bean id="simpleDataSourceConnector" class="com.alipay.sofa.dbp.DbpDataSource" init-method
="init">
    <property name="delegate" ref="simpleDataSource"/>
    <property name="dbpInstanceId" value="${yourODPInstanceId}"/>
    <property name="appName" value="${yourAppName}"/>
    <property name="database" value="${yourDatabaseName}"/>
    <!-- 增加链路追踪配置 -->
    <property name="clientTracer" ref="clientTracer"/>
 </bean>
 <!-- 连接器配置结束 -->
 <!-- 数据源连接池配置 -->
<bean id="simpleDataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="ini
t">
         <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
         <property name="url" value="jdbc:mysql://${yourODPConnectionURL}:8306/${yourDataba
seName}"/>
         <property name="username" value="${user}"/>
         <property name="password" value="${password}"/>
</bean>
<!-- 数据源连接池配置结束 -->

 <bean id="clientTracer" class="com.alipay.sofa.dbp.DbpClientTracer">
       <!-- 设置客户端日志才样的阈值，默认3ms（3ms内成功的sql按10%采样），设置为0 全部打印 -->
       <property name="sampleThreshold" value="3"/>
</bean>
```

阿里云双机房配置

要求 dbp-connector 的版本号为 1.1.1 及以上。ODP 阿里云实例的域名格式为：`阿里云 ODP 实例 ID(去掉中划线)+zone 按中划线分割的最后一部分.[public.]sofaodp.aliyuncs.com`，如 `sofaodpojiss8o73o5la.sofaodp.aliyuncs.com`。

```
<!-- 增加连接器配置 -->
 <bean id="simpleDataSourceConnector" class="com.alipay.sofa.dbp.DbpDataSource" init-method
="init">
    <property name="delegate" ref="simpleDataSource"/>
    <property name="dbpInstanceId" value="${yourODPInstanceId}"/>
    <property name="appName" value="${yourAppName}"/>
    <property name="database" value="${yourDatabaseName}"/>
 </bean>
 <!-- 连接器配置结束 -->
 <!-- 数据源连接池配置 -->
<bean id="simpleDataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="ini
t">
     <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
     <!-- 形如jdbc:mysql://sofaodp-ojiss8o73o5l@zone@.[public.]sofaodp.aliyuncs.com:8306/tes
tdb -->
     <property name="url" value="jdbc:mysql://${yourODPInstanceId}@zone@.sofaodp.aliyuncs.c
om:8306/${yourDatabaseName}"/>
     <property name="username" value="${user}"/>
     <property name="password" value="${password}"/>
</bean>
<!-- 数据源连接池配置结束 -->
```

专有云双机房配置

修改 Spring 配置文件如下：

```
<!-- vip寻址-->
<bean id="dbpDiscovery" class="com.alipay.sofa.dbp.discovery.DbpDiscovery"/>

<!-- dbp-connector代理-->
<bean id="delegatingDataSource" class="com.alipay.sofa.dbp.DbpDataSource" init-method="init
">
      <property name="delegate" ref="simpleDataSource"/>
      <property name="appName" value="${yourAppName}"/>
      <property name="database" value="${yourDatabase}"/>
      <property name="dbpInstanceId" value="${yourDbpInstanceId}"/>
      <property name="clientTracer" ref="clientTracer"/>
</bean>

<!-- 连接池配置，以druid为例 -->
<bean id="simpleDataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="ini
t">
     <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
     <property name="url" value="jdbc:mysql://${yourDbpInstanceId}:8306/${yourDatabase}"/>
     <property name="username" value="${username}"/>
     <property name="password" value="${password}"/>
</bean>

<bean id="clientTracer" class="com.alipay.sofa.dbp.DbpClientTracer"/>
```

8.3. 使用说明

本文介绍如何使用数据访问代理连接器来进行 SQL 链路追踪、指定分库分表路由，以及 拦截器扩展。

SQL 链路追踪

使用以下 HINT 语句，根据 `traceId` 和 `RpcId` 对 SQL 数据库进行追踪：

```
/*+DBP: $SYS={TRACE(TraceId#RpcId)}*/select*from{table_name}
```

HINT 语句示例：

```
/*+DBP: $SYS={TRACE(0a0fe91c151497435345910091964 9#0.1)}*/select*from test
```

数据访问代理连接器会将包括追踪信息在内的 SQL 执行信息输出到本地日志 `dbp-connector-digest.log` 中。

- 默认情况下，对于执行时间小于 3ms 且执行成功的 SQL 命令，按照 10/1 的比例抽样打印；
- 对于执行失败或执行时间大于 3ms 的 SQL 命令，全量打印。

输出的日志格式如下：

```
日志输出时间,应用名称,traceId,RpcId,SQL 语句,执行结果（success/failed),执行耗时（ms）,连接建立时间,数据库执行时间,当前线程名
```

输出的日志示例如下：

```
2018-01-0318:12:33.541,xxx,0a0fe91c151497435345910091964 9,0.1,select*from test,success,81ms,77ms,4ms,main
```

分库分表路由指定

使用以下 HINT 语句，指定 SQL 访问某一特定的分库分表：

```
/*+DBP: $ROUTE={GROUP_ID(分库位),TABLE_NAME(物理表名)}*/ SQL 语句
```

HINT 语法格式说明：

- `GROUP_ID`：指定分库位，等同于 `dbRule` 返回的结果；
- `TABLE_NAME`：指定物理表名。

另外，数据访问代理连接器提供了拦截器，当您在工程代码中指定分库分表后，在 SQL 语句执行前自动拼接路由 HINT 语句。当您在脚本中定义以下字段后：

```
RouteParameters routeParameters =RouteCondition.newRouteParameters();
routeParameters.setGroupId("分库位");
routeParameters.setTargetTables(Collections.singletonList("物理表名"));
cn = dbpDataSource.getConnection();
ps = cn.prepareStatement("select * from xxxx");
```

数据访问代理连接器会将其自动转换为以下 HINT 语句：

```
/*+DBP: $ROUTE={GROUP_ID(分库位),TABLE_NAME(物理表名)}*/select*from xxxx
```

拦截器扩展

您可通过实现拦截器接口获得更多扩展功能。配置方法如下：

1. 在 Spring 的配置文件中，添加以下 bean（以 `sqlCountInterceptor` 类为例）：

```
<bean id="simpleDataSourceConnector" class="com.alipay.sofa.dbp.DbpDataSource" init-metho
d="init">
<property name="delegate" ref="simpleDataSource"/>
<property name="securitySpec" ref="securitySpecImpl"/>
<property name="interceptors">
<list>
<ref bean="sqlCountInterceptor"/>
</list>
</property>
</bean>

<bean id="sqlCountInterceptor" class="com.alipay.sofa.dbp.SqlCountInterceptor"/>
```

2. 为添加的 bean 定义拦截器的代码实现，如下所示：

```
@NotThreadSafe
public class SqlCountInterceptor implements Interceptor {
    private static final Map<String, Integer> sqlCounter = new HashMap<>();

    @Override
    public Object intercept(Chain chain) throws Exception {
        String sql = chain.sql();
        Integer count = sqlCounter.get(sql);
        if (count == null) {
            sqlCounter.put(sql, 1);
        } else {
            count += 1;
            sqlCounter.put(sql, count);
        }
        return chain.proceed();
    }

    public int getSqlExecuteCount(String sql) {
        Integer count = sqlCounter.get(sql);
        if (count == null) {
            return0;
        }
        return count;
    }
}
```

9.自定义 HINT

HINT 作为一种 SQL 补充语法，在关系型数据库中扮演着非常重要的角色。它允许用户通过相关的语法影响 SQL 的执行方式，从对 SQL 进行特殊的优化。

同样，数据访问代理也提供了自定义的 HINT 语法。HINT 使用基本语法：

```
/*+DBP: hint 语句 */ SQL 语句
```

数据访问代理自定义 HINT 基于 MySQL 注释 实现，其 HINT 语句会位于 `/*` 与 `*/` 之间，并且必须以 `+DBP:` 开头，用以识别为数据访问代理的 HINT 语法。HINT 语句是数据访问代理的 HINT 命令，目前仅提供路由相关的 HINT 命令，后续会提供更丰富的 HINT 使用场景。

自定义路由 HINT

分库分表场景下使用数据访问代理时，每一个 SQL 请求语句中都需要存在分库分表字段。在无法获得分库分表字段场景下，如果需要请求指定数据分片、分表执行 SQL 语句时，可以通过自定义路由 HINT 实现。

自定义路由 HINT 基本语法：

```
/*+DBP: $ROUTE={GROUP_ID(分片位),TABLE_NAME(物理表名)}*/ SQL 语句
```

路由 HINT 使用说明：

- `GROUP_ID` 表示指定的分片 ID，数值类型
- `TABLE_NAME` 表示数据所在的物理表名。如果仅存在分库规则，无分表规则的话，`TABLE_NAME` 无需配置

```
mysql>/*+DBP: $ROUTE={GROUP_ID(0),TABLE_NAME(user_00)}*/ SELECT * FROM user WHERE id =1;
+------+------+
| id   | name |
+------+------+
|1| foo  |
+------+------+
1 row inset(0.00 sec)
```

全表扫描 HINT

自定义路由 HINT 需要知道数据的路由结果，如果无法知道数据的路由结果可以通过全表扫描 HINT 来扫描所有的分库分表获取数据。

全表扫描 HINT 基本语法：

```
/*+DBP: $ROUTE={SCAN_ALL()}*/ SQL 语句
```

```
mysql>/*+DBP: $ROUTE={SCAN_ALL()}*/ SELECT * FROM user;
+------+-------+
| id   | name  |
+------+-------+
|1| foo1  |
|2| foo2  |
+------+-------+
1 row inset(0.00 sec)
```

虚拟字段路由 HINT

当进行过数据分片（即采取了分库分表）时，都是基于某一特定字段，或者某两个字段进行拆分。这样在执行所有的数据库访问时都需要提供这些字段的 Value，否则数据路由将报错。然而，有时在执行 INSERT、UPDATE、SELECT 时 SQL 中的参数并不包含分库分表字段，这时我们需要使用虚拟字段路由 HINT 来解决该问题。

添加 SHARIND_KEY 关键字：`$ROUTE={SHARDING_KEY(key1=value1,key2='value2')}` key 为虚拟字段名称，value 为虚拟字段值。

如：

- id='123'，类型为字符串，单引号和双引号均可。

```
/_+DBP: $ROUTE={SHARDING_KEY(id='123')} _/
```

- id=123，类型为数字，真实类型是 java.lang.Integer。这里暂时不处理 long 类型，全都视为integer。

```
/_+DBP: $ROUTE={SHARDING_KEY(id=123)} _/
```

- 多个 sharding key，注意可以两边加入空格。

```
/_+DBP: $ROUTE={SHARDING_KEY(id='123', name  =  'abc'  )} _/
```

注意事项

MySQL 命令行客户端

由于数据访问代理自定义 HINT 是以 MySQL 注释 的形式使用的，在使用 MySQL 官方命令行客户端执行带有数据访问代理自定义 HINT 的 SQL 时，请在登录命令中加上 `-c` 参数。否则，该客户端会将注释语句删掉再发送到服务端执行，导致数据访问代理自定义 HINT 失效。具体请查看 MySQL 官方客户端命令。

全表扫描

- 要避免全表扫描对后端数据库的压力，请尽量减少全表扫描 HINT 使用。如需使用，也请加上合理的查询过滤条件。
- 全表扫描会读取多个表的结果做合并，如果使用 `LIMIT start,offset` 得到的结果无法保证正确性。所以，全表扫描 HINT 仅支持 `LIMIT offset` 以及 `LIMIT 0, offset` 语法。
- 全表扫描在超时情况可能会引起流式结果集问题。报错信息如下：

```
Streaming result set com.mysql.jdbc.RowDataDynamic is still active.No statements may be i
ssued when any streaming result sets are open and in use on a given connection.Ensure tha
t you have called .close() on any active streaming result sets before attempting more que
ries.
```

此时，您可以在 ODP 连接参数的 connectionProperties 中添加 `clobberStreamingResults=true` 配置，并使其生效，即可解决该问题。

10.分布式序列

数据访问代理提供了生成分布式环境下的分布式唯一序列（Sequence）的能力，该序列有全局唯一、全局递增的特性，常用于分库分表下的主键、业务主键生成的场景。

> 重要
>
> 数据访问代理分布式序列功能是基于数据库实现，如果需要使用该功能，需要在业务数据库中创建 `dbp_sequence` 表。

普通序列

数据访问代理的分布式序列功能提供了类 Oracle 语法的 SQL 语句，`seq_name.nextval`，其中 `seq_name` 是任意字符串，一般是一张逻辑表使用同一个 `seq_name`，基于单库单表的 `dbp_sequence` 表实现。使用如下：

```
SELECT order_seq.nextval FROM dual
```

业务序列

如前面提到的，分布式序列功能基于数据库表实现，`dbp_sequence` 可以部署成单库单表模式，同样也可以部署成分库分表模式，分库分表模式下有如下优点：

- 提升 Sequence 表的读写能力。
- 提升 Sequence 表的可用性，无单点故障。
- 通过将 Sequence 表和业务数据表部署在一起，保持数据拆分规则一致，方便生成业务主键。

数据访问代理的分布式序列在 `nextVal` 语法基础下，扩展了更多业务型的字段，在获得 Sequence 值之外，还可以获得更多分库分表相关的信息，开发者可以根据实际场景灵活组装业务的序列号。使用如下：

```
SELECT
  order_seq.timestamp, order_seq.dbtimestamp, order_seq.groupid,
  order_seq.tableid, order_seq.nextval
FROM dual
WHERE sharding_col =?
```

其中各项字段的含义：

- `timestamp`：获取 Sequence 的时间戳，该时间是机器时间。
- `dbtimestamp`：获取 Sequence 数据库的时间，使用该命令的话，数据访问代理会实时向物理数据库请求当前时间。
- `groupid`：获取 Sequence 所在分片的 ID。
- `tableid`：获取 Sequence 所在分表的 ID。
- `sharding_col`：获取 Sequence 的分库分表字段，该字段是虚拟并不真实存在，用以方便数据访问代理计算分库分表规则。

配置分表规则

在数据库中添加 `dbp_sequence` 表的分表规则，以下是以 100 个分片 100 个分表，以 `sharding_col` 字段 hash 规则为示例：

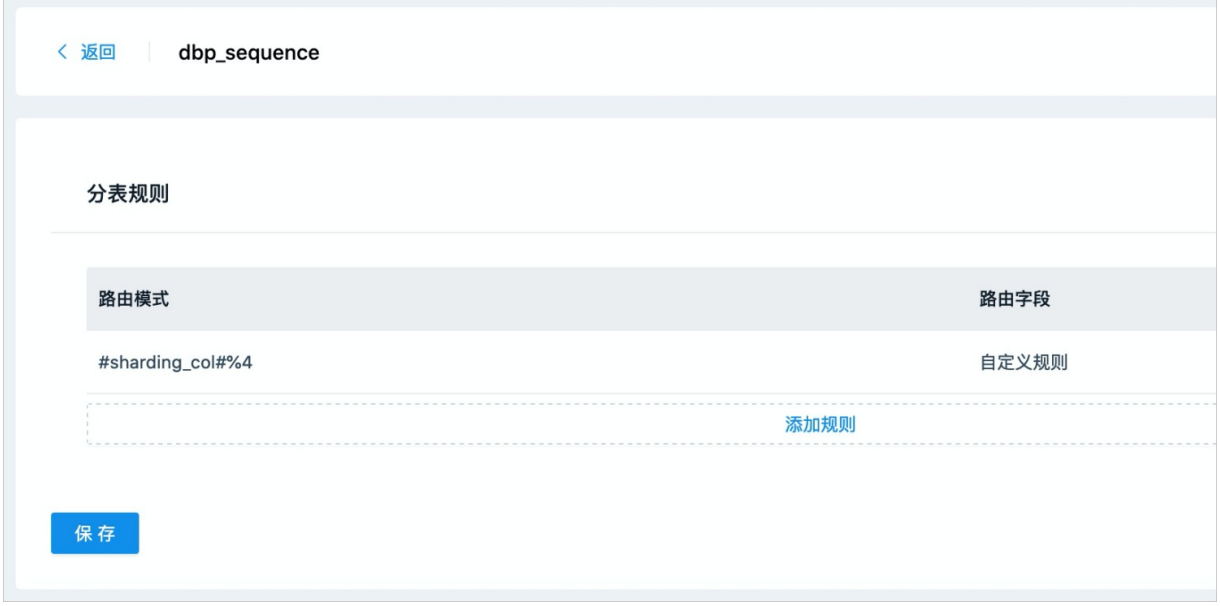
全局序列

通过配置分布式序列，获取全局序列的核心步骤如下：

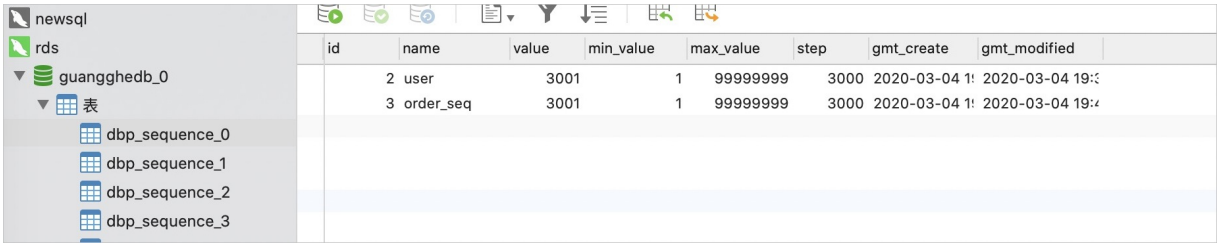
1. 在 DB 中创建 `dbp_sequence` 表。
   - 建议您在数据访问代理中创建逻辑表 `dbp_sequence` 时，让数据访问代理去创建 DB 中的 `dbp_sequence` 表（数据访问代理会按照分表规则，自动创建相应的表）。
   - 假如是分表模式，则添加的分表必须以 `sharding_col` 为分表字段，例如 `#sharding_col#%4`，注意这个字段不能变。
2. 在客户端使用 `SELECT user.nextval FROM dual WHERE sharding_col = xxxx` 去获取 sequence。
   - 其中，`dual` 和 `sharding_col` 不能变。
   - `user` 代表客户的逻辑表，会被数据访问代理填充到 `dbp_sequence` 表的 `name` 字段里面去，表示该 sequence 是为 `user` 这个表产生的。但是，这里没有强相关性，只是为了程序中好控制唯一性。

数据访问代理上的逻辑表，示例如下：

`dbp_sequence` 的分表规则，示例如下：



物理数据库中的表和内容示例如下：



> **说明**
>
> 表中内容是客户端运行 SQL 后自动创建的，其中 SQL 是用来获取 sequence 的。

获取 sequence 的 SQL 代码，示例如下：

```
//          query sequence
String selectSql = "SELECT user.nextval FROM dual WHERE sharding_col = ?";
PreparedStatement queryPreparedStatement = connection.prepareStatement(selectSql);
queryPreparedStatement.setInt( parameterIndex: 1,  x: 1100);
ResultSet resultSet = queryPreparedStatement.executeQuery();
while (resultSet.next()) {
    System.err.print("  id:" + resultSet.getInt( columnIndex: 1) + ", ");
}
```

注意事项

- 分库分表场景下，`nextval` 是每个分片/分表独立递增，不同分片/分表的 `nextval` 会有重复的情况。所以请拼接其他信息用以区分，如：`groupid` 。
- `nextval` 不保证严格递增，且也有上限，默认 `nextval` 在 1 ~ 99999999 中重复循环。
- 单条 SQL 中不支持多个 Sequence 名字。

附录

dbp_sequence 建表语句

```
CREATE TABLE dbp_sequence (
`id` INT AUTO_INCREMENT,
`name` VARCHAR(255),
`value` INT,
`min_value` BIGINT,
`max_value` BIGINT,
`step` BIGINT,
`gmt_create` DATETIME,
`gmt_modified` DATETIME,
  PRIMARY KEY (`id`),
  UNIQUE KEY (`name`)
);
```

字段说明：

字段名称	类型	说明
name	VARCHAR	记录对应的业务表名，例如：trade_order。
min_value	BIGINT	Sequence 最小值，用于校验 Seqeunce 不能低于该数值，否则报错，默认为 1。
max_value	BIGINT	Sequence 最大值，当到达最大值以后，将从 min_value 开始重新增加，默认为 99999999。
step	BIGINT	一次获取的 Sequence 区间，默认 10000。
value	INT	当前 Sequence 值。
gmt_create	DATETIME	创建时间。
gmt_modified	DATETIME	修改时间。

11.自定义分表规则

当数据访问代理控制台提供的默认分表规则无法满足需求时，用户可以自定义分表规则。自定义分表规则支持 Groovy 表达式。

假设数据库名为 dbptest，数据表名为 user，分片数设置为 10 个，分表数为 100 个，自定义分表规则示例如下：

示例一

路由字段为 user_id（整型）= 123456

```
Integer.valueOf(String.valueOf(#user_id#).substring(3,5))%100
```

通过上面的分表规则，截取 user_id 对应字符串中序号为 3 至 5 之间（不包括 5 ）的子字符串，转换为整型，并对 100 求模，计算结果为 45。

最终的路由结果为 dbptest_4.user_45。

示例二

路由字段为 user_id（整型）= 123456

```
Integer.valueOf(String.valueOf(#user_id#).substring(String.valueOf(#user_id#).length()-2))%100
```

通过上面的分表规则，截取 user_id 对应字符串的最后两位，转换为整型，并对100求模，计算结果为 56。

最终的路由结果为 dbptest_5.user_56。

示例三

路由字段为 user_id（字符型）= '123456'

```
Integer.valueOf(#user_id#.substring(3,5))%100
```

通过上面的分表规则，截取 user_id 中序号为 3 至 5 之间（不包括 5 ）的子字符串，转换为整型，并对 100 求模，计算结果为 45。

最终的路由结果为 dbptest_4.user_45。

示例四

路由字段为 user_id（字符型）= '123456'

```
Integer.valueOf(#user_id#.substring(#user_id#.length()-2))%100
```

通过上面的分表规则，截取 user_id 的最后两位，转换为整型，并对 100 求模，计算结果为 56。

最终的路由结果为 dbptest_5.user_56。

12.日志说明

客户端日志

数据访问代理连接器 `dbp-connetor` 是一个标准的 JDBC 实现，为用户提供了丰富的客户端扩展功能，例如 traceId 透传、SQL 链路追踪、数据库加密解密等功能。结合 `dbp-connector` ，即可在 `logs/dbp-connector-digest.log` 日志中，获取详细的 SQL 执行信息，包括 trace 信息等。

- **日志格式：**

  日志输出时间应用名称,traceId,RpcId,SQL 语句,执行结果（success/failed）,执行耗时（ms）,连接建立时间,数据库执行时间,当前线程名

- **日志示例：**

```
2019-01-0318:12:33.541,xxx,0a0fe91c151497435345910091964 9,0.1,select*from test,success,81ms,77ms,4ms,main
```

服务端日志

SQL 执行摘要日志

`logs/tracelog/zdal-db-digest.log` 是 SQL 执行摘要日志，是采样打印的。对于所有耗时大于 3ms 且执行失败的语句，会进行全量打印；而耗时小于 3ms 的语句，则会进行抽样打印。

- **日志格式：**

  日志打印时间,DBP实例ID,TraceId,RpcId,数据源名称,逻辑库名称,表名,SQL 类型,SQL,执行结果（success/failed）,执行耗时（ms）,当前线程名,数据库类型,逻辑表名,链接建立时间,数据库执行时间

- **日志示例：**

```
2019-11-2216:40:15.112,Test,0a0fe93a1479804014974100121175,0.3,DataSourceName,LogDatabaseName,tableName,UPDATE,update user set pwd='1'where name='khotyn',success,8ms,main,mysql,logicTableName,1ms,0ms,mark=T&uid=a2&,mark=T&uid=a2&
```

SQL 执行统计日志

`logs/tracelog/zdal-db-stat.log` 是 SQL 执行统计日志，每分钟会针对不同库表 SQL 执行统计信息进行打印。

- **日志格式：**

  日志打印时间,DBP实例ID,数据源名称,逻辑库名称,表名,数据库类型,本段时间内的请求数量,本段时间内的请求总耗时,结果信息（Y/N）,全链路压测标志(T/F)

- **日志示例：**

```
2015-05-1120:03:30.487,Test,DataSourceName,LogDatabaseName,tableName,mysql,2,27,N,F
```

ODP 错误日志

`logs/zdalproxy/common-error.log` 是数据访问代理的错误日志，打印记录了详细的错误信息。

ODP 端到端的慢 SQL 日志

`logs/zdalproxy/sql-digest.log` 是数据访问代理端到端的慢 SQL 日志。对于 ODP 端到端的所有耗时大于 4ms 且执行失败的语句会进行全量打印；而小于 4ms 的日志，则会进行抽样打印。

- **日志格式**：

  SQL 执行时间戳,traceId,rpcId,数据库名,数据表名,SQL 类型,SQL,是否是默认数据源,是否是事务,是否执行成功,错误码,整体执行时间us,入队时间us,队列等待时间us,SQL 解析时间us,获取连接时间us,执行时间us,数据回写时间us

- **日志示例**：

```
2019-01-2419:00:02,046,0afd010915167916020406502174 3,0,cb_batch,SELECT,SELECT VERSION FRO
M spring_batch_JOB_EXECUTION WHERE JOB_EXECUTION_ID=29638,N,Y,Y,0,6830,18,17,14,1,6749,20
```

ODP 端到端的 SQL 统计日志

`logs/zdalproxy/sql-stat.log` 是数据访问代理端到端的 SQL 统计日志。

- **日志格式**：

  日志打印时间,数据库名,SQL 类型,是否执行成功,执行次数,总整体执行时间us,总入队时间us,总队列等待时间us,总SQL 解析时间us,总获取连接时间,总执行时间us,总数据回写时间us

- **日志示例**：

```
2019-01-2410:28:22,888,user,SELECT,Y,24,80758,980,835,1969,122,62506,13117
```

ODP 服务端启动初始化日志

`logs/zdalproxy/zdal-client.log` 是数据访问代理服务端启动初始化日志，记录了整个初始化阶段的启动过程。

SQL 执行耗时监控日志

`logs/zdalproxy/zdal/zdal-monitor.log` 是 SQL 执行监控日志，用以打印异常 SQL 与慢 SQL。

数据库连接池日志

`logs/zdalproxy/zdal/zdal-datasource.log` 是物理数据源连接池统计日志，用以记录物理数据源下数据库连接池的变更。

日志示例：

```
2019-06-2615:05:57,305-[[,,,]]tradecore.physics07[Min:2;Max:4;CanUse:4;Managed:2;Using:0;Ma
xUsed:0;CreateCount:2;DestroyCount:0]
2019-06-2615:05:57,306-[[,,,]]tradecore.physics08[Min:2;Max:4;CanUse:4;Managed:2;Using:0;Ma
xUsed:0;CreateCount:2;DestroyCount:0]
2019-06-2615:05:57,306-[[,,,]]tradecore.physics01[Min:2;Max:4;CanUse:4;Managed:2;Using:0;Ma
xUsed:0;CreateCount:2;DestroyCount:0]
2019-06-2615:05:57,307-[[,,,]]tradecore.physics02[Min:2;Max:4;CanUse:4;Managed:2;Using:0;Ma
xUsed:2;CreateCount:2;DestroyCount:0]
```

日志中涉及的连接参数如下：

- `Min`：连接池连接数最小值。
- `Max`：连接池连接数最大值。

- `CanUse`：可使用连接数。
- `Managed`：当前创建了多少连接。
- `Using`：这个瞬间正在使用多少连接。
- `MaxUsed`：获取连接数的最大并发数。一般是指最多使用过多少个连接数，不会超过 `Max` 连接数。
- `CreateCount`：连接创建总次数。
- `DestroyCount`：连接销毁总次数。

分库分表规则初始化日志

`logs/zdalproxy/zdal/zdal-rule.log` 是数据访问代理分库分表的初始化日志，记录了数据拆分规则的加载情况。

13.资源授权

在数据访问代理控制台上访问阿里云其他资源时，比如 OceanBase/RDS 数据库实例，必须拥有云资源访问权限，即拥有 AliyunSofaOdpDefaultRole 角色。本文将介绍如何对账号授予 AliyunSofaOdpDefaultRole 角色权限。

操作步骤：

1. 在 **数据访问代理** 控制台，单击 **运维** > **资源授权** 进入资源授权页面，单击 **激活授权** 按钮。
2. 单击 **同意授权**。

3. 查看账号是否被授予了 AliyunSofaOdpDefaultRole 角色权限。

   授权成功后，在 RAM 控制台，**RAM 角色管理** 页面，搜索框里输入 AliyunSofaOdpDefaultRole。

   - 下面展示 AliyunSofaOdpDefaultRole 角色，说明账号授予了该角色权限。

   - 下面未展示 AliyunSofaOdpDefaultRole 角色，说明账号未授予该角色权限。

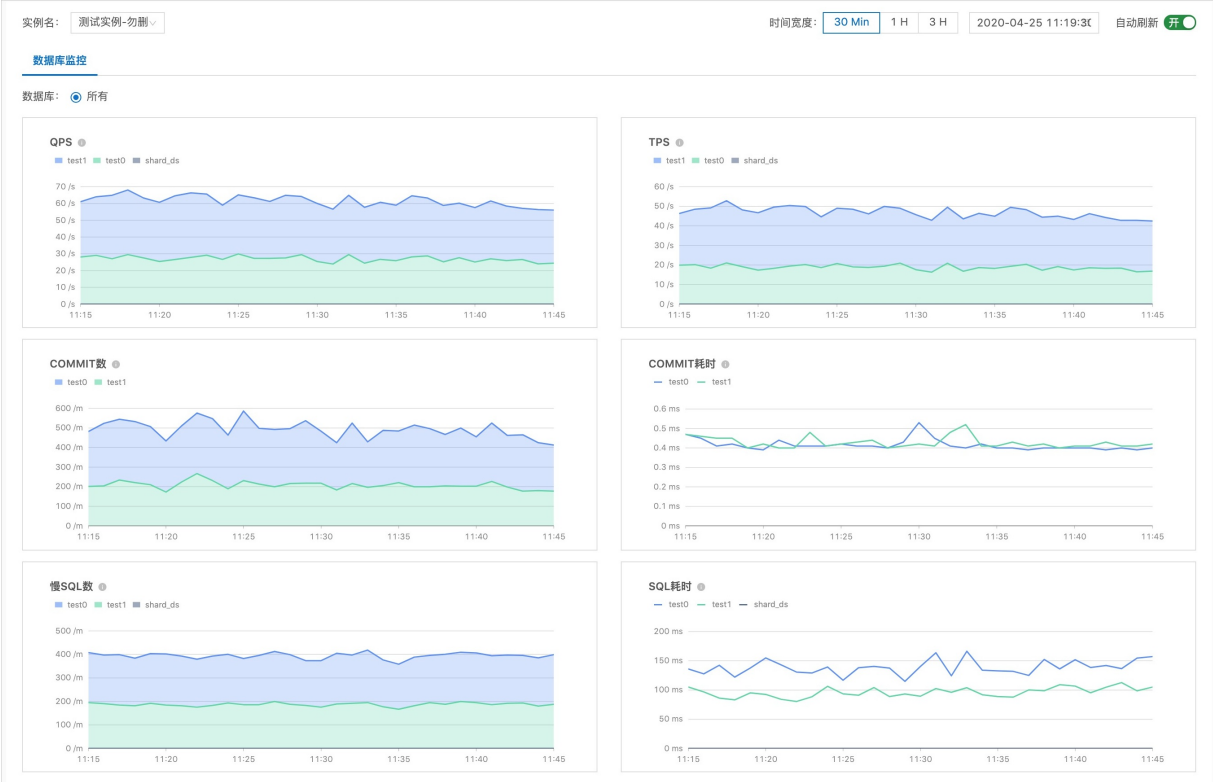
14.监控

14.1. 性能监控

数据访问代理提供了 SQL 性能监控面，帮助业务实时监控业务 SQL 的执行情况。

操作步骤

1. 进入数据访问代理控制台，单击左侧导航栏上的 **监控**。
2. 在监控页面的左上角 **实例名** 的下拉菜单中，选择您想要查看的实例。
3. 在页面的右上角，选择您想要查看的监控数据的具体日期与时间。
4. 选择完毕后，在 **数据库监控** 页签，即可查看以下监控数据：
   - **QPS（Query Per Second）**：每秒执行查询 SQL（SELECT/SEQUENCE）的量。
   - **TPS（Transaction SQL Per Second）**：每秒执行变更 SQL（INSERT/UPDATE/DELETE）的量。
   - **COMMIT 数**：Commit Per Minute，即每分钟显式执行事务提交操作的量。
   - **COMMIT 耗时**：显式提交事务操作的平均耗时。
   - **慢 SQL 数**：每分钟慢 SQL 的数量。耗时超过一定阈值的 SQL 会被定义为慢 SQL，阈值是 500 毫秒，暂时不支持自定义。
   - **SQL 耗时**：SQL 操作的平均耗时。



14.2. 慢 SQL 监控

数据访问代理中，**慢 SQL 监控** 是通过 阿里云日志服务 SLS 收集数据访问代理实例的 SQL 执行日志，将日志存储到用户空间中，在收集到 SQL 数据后，以图表的方式展示出收集到的 SQL 详情。其中，慢 SQL 是指执行耗时大于等于 500ms 的 SQL。慢 SQL 日志最多可以保存 7 天。

前置条件

使用 **慢 SQL 监控** 功能，您需要确保已满足以下条件：

- 已购买了数据访问代理实例，并在该实例下创建了逻辑数据库。具体步骤，参见 购买数据访问代理实例 与 创建数据访问代理数据库。
- 已开通了阿里云的 日志服务 SLS。数据访问代理依赖阿里云的日志服务进行 SQL 审计与分析。

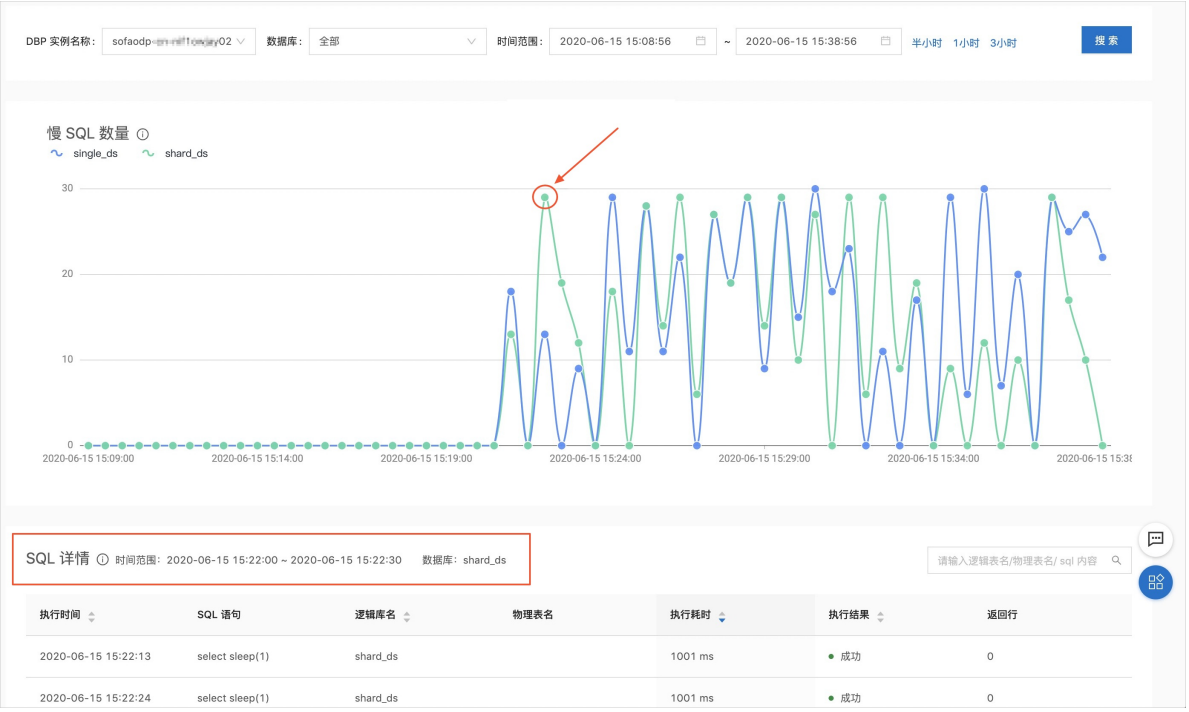
> 重要
>
> - 日志服务 SLS 会对存储的 SQL 日志收取相应的费用。
> - 目前 SQL 审计功能在专有云环境未开放。

使用慢 SQL 监控功能

1. 进入数据访问代理控制台，在左侧导航栏中，选择 **监控 > 慢 SQL 监控**。
2. 在 **慢 SQL 监控** 页面，单击 **SQL 审计设置** 按钮。
3. 在新弹出窗口中，选择数据访问代理实例与该实例下的目标数据库后，单击 **确定**。

4. 设置完成后，当前页面将以图表的形式展示以下两部分 SQL 信息：
   - **慢 SQL 数量**：展示了指定时间范围内的慢 SQL（耗时大于等于 500ms）的统计数据变化图。其中，横坐标为一小段时间（比如 30 秒），纵坐标为该一小段时间内慢 SQL 的总数。
   - **SQL 详情**：展示了指定时间范围内的所有 SQL 及其详情，包括执行时间、逻辑库名、执行耗时等。

5. 在 **慢 SQL 数量** 图中，单击图中任意点，即可查询该逻辑库在该时间段内的 SQL 详情，如下图所示。

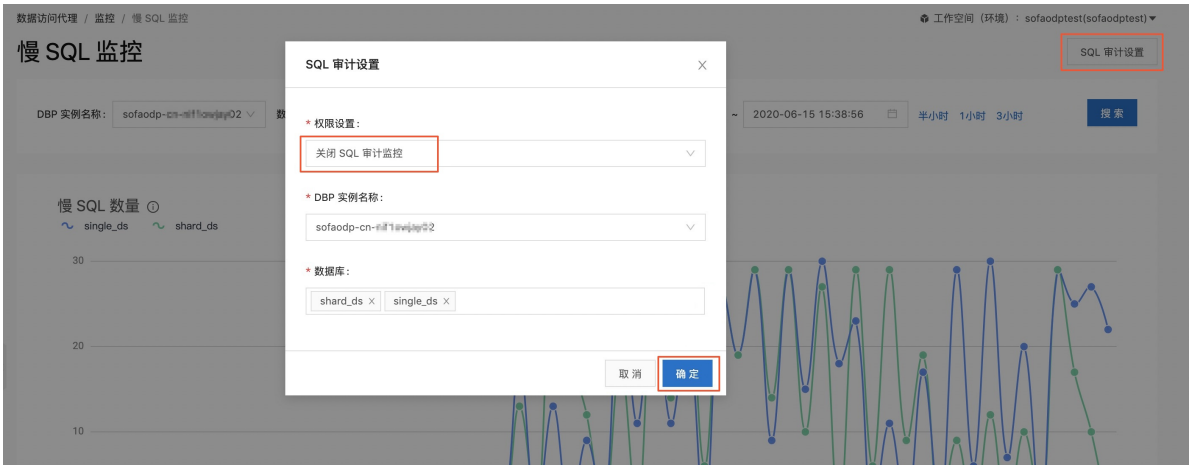


6. 在 **SQL 详情** 列表中，您还可以按照表名、SQL 内容进行模糊搜索，如下图所示。

关闭慢 SQL 监控功能

SQL 日志均会存储在用户空间，日志服务 SLS 会对此收取一定量的费用。因此，如果部分逻辑库不再需要慢 SQL 监控功能，建议将其关闭。操作步骤如下：

1. 在 **慢 SQL 监控** 页面，单击右上角的 **SQL 审计设置** 按钮。
2. 在 **权限设置** 栏中，选择 **关闭 SQL 审计监控** 后，单击 **确定**。



15.最佳实践

15.1. 选择 ODP、RDS 与 OceanBase 的实例规格

数据访问代理（Open Database Proxy，简称 ODP）、RDS 与 OceanBase 的实例规格均按照 CPU 的处理能力、内存容量和磁盘空间等来划分。系统提供多种不同规格的实例供您选择，规格越高代表实例的处理能力越强。

选择 ODP 实例规格

ODP 提供灵活的实例规格供用户选择，不同规格支持的最大 QPS（每秒 SQL 数）均不同。在选择 ODP 实例时，请参考实际的业务场景，评估好业务的最大 QPS 后，根据不同规格支持的最大 QPS 进行购买。

例如：4 核 8 G 规格的 ODP 实例中，只有一台 ECS 机器提供服务，所以建议仅在开发测试阶段使用该规格。对于生产环境，推荐使用 8 核 16 G 及以上的规格，以保障业务的连续性。

> **重要**
>
> ODP 的最大 QPS 估算与实际执行的 SQL 复杂度、真实数据量、数据库表结构和数据库规格相关。本文档的评估结果建立在简单的单条数据查询的基础上，且查询条件中带有拆分键，用于路由到单个分片数据，单条数据容量不超过 1 KB。

选择 RDS 实例规格

需根据当前以及未来几年的业务情况对数据库的规格、磁盘、IOPS、连接数进行估算，选择需要购买的 RDS 实例数量和单个 RDS 实例的规格。更多关于 RDS 实例规格的选择，请参考 RDS 购买指南。

选择 OceanBase 实例规格

系统提供机房级、城市级的容灾方案，因此在 OceanBase 实例规格选择中，除了需要考虑物理规格、磁盘、IOPS、连接数外，您还需考虑选择何种级别（机房级、城市级）的容灾方案，具体的实例规格和容灾方案相关，请联系技术人员或 提交工单 以便进行详细评估。

15.2. 选择分片数

数据访问代理在创建数据库时，提供了分片数的概念。分库分表中的分库是一个逻辑上的概念，物理上可能是一个物理数据库代表一个“分库”，也可能是多个物理数据库组成一个“分库”，在数据访问代理里面统一概念称 “分片”。

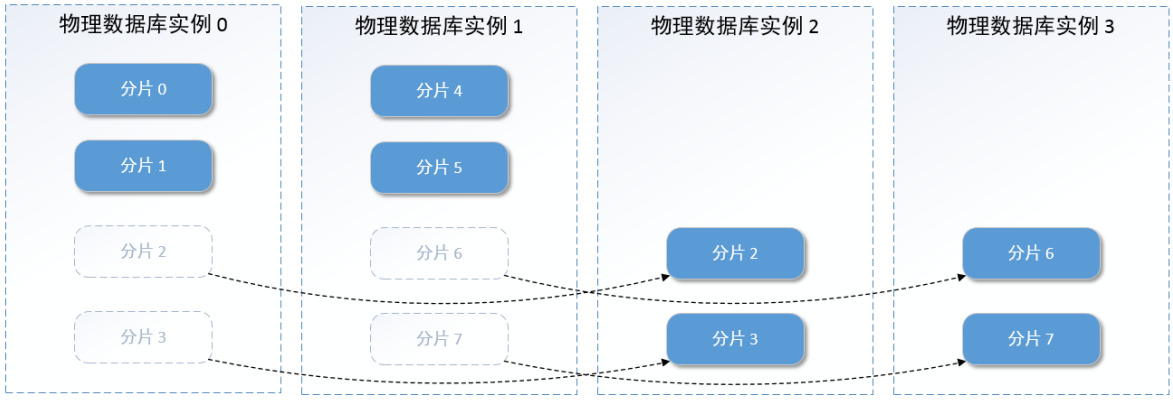
选择分片数原则

分片数决定了数据访问代理数据库数的逻辑最大值，创建后无法修改。一般情况下，建议分片数和分表数保持一致，根据业务 3 到 5 年内的数据量设置一个较大值（如设置为 128），这样在做数据库扩容与迁移的时候，无需修改路由规则，以表为单位做整体的数据迁移操作即可。

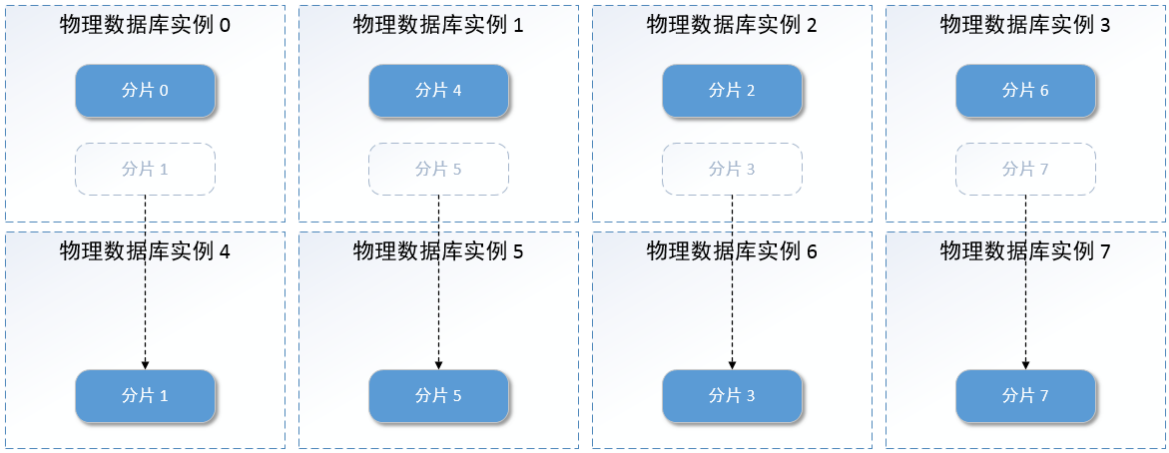
场景分析

如果某用户分片数设置较小（假设设置分片数大小为 8），业务发展前期只购买了 2 个物理数据库实例，分片数为 8 则每个物理数据库实例均匀的存放了 4 个分片，结构如下：

随着业务规模快速发展，两个物理数据库实例已经不能够满足业务要求，该用户决定扩展成 4 台，那么每台物理数据库实例应该存放了 2 个分片，通过数据访问代理进行数据平滑迁移后，结构如下：



后续业务规模继续发展，最终也只能将物理数据库实例扩展为 8 台，通过数据访问代理进行数据平滑迁移后，结构如下：



如果扩展需要超过 8 台的话，数据迁移过程所有数据需要重新分配，涉及到的数据迁移工作会比较复杂，而且对应的分库分表规则脚本、业务代码中的自定义路由 HINT 也需要一起调整，成本较高且无法完全平滑迁移。

所以建议在业务设计之初，为了满足业务 3 到 5 年内的数据库容量，合理的选择一个较大的分片数的值。

15.3. 使用拆分字段

拆分字段即分库/分表字段，是在水平拆分过程中用于生成拆分规则的数据表字段。数据访问代理会根据拆分字段的值将数据表水平拆分到每个物理库实例上的物理分库/分表中。

拆分原则

数据表拆分的首要原则，就是要尽可能找到数据表中的数据在业务逻辑上的主体，并确保大部分（或核心的）数据库操作都是围绕这个主体的数据进行，然后可使用该主体对应的字段作为拆分字段，进行分库分表。

选择业务逻辑主体

业务逻辑上的主体，通常与业务的应用场景相关，下面的一些典型应用场景都有明确的业务逻辑主体，可作为拆分字段：

- 面向用户的金融应用，都是围绕用户维度来做各种操作，那么业务逻辑主体就是用户，可使用用户对应的字段作为拆分字段；
- 侧重于卖家的电商应用，都是围绕卖家维度来进行各种操作，那么业务逻辑主体就是卖家，可使用卖家对应的字段作为拆分字段；
- 游戏类的应用，是围绕玩家维度来做各种操作，那么业务逻辑主体就是玩家，可使用玩家对应的字段作为拆分字段；
- 车联网方面的应用，则是基于车辆信息进行操作，那么业务逻辑主体就是车辆，可使用车辆对应的字段作为拆分字段；
- 税务类的应用，主要是基于纳税人的信息来开展前台业务，那么业务逻辑主体就是纳税人，可使用纳税人对应的字段作为拆分字段。

如果确实找不到合适的业务逻辑主体作为拆分字段，那么可以考虑下面的方法来选择拆分字段：

- 根据数据分布和访问的均衡度来考虑拆分字段，尽量将数据表中的数据相对均匀地分布在不同的物理分库/分表中，适用于大量分析型查询的应用场景（查询并发度大部分能维持为 1）；
- 按照数字（字符串）类型与时间类型字段相结合作为拆分字段，进行分库和分表，适用于日志检索类的应用场景。

> 重要
>
> - 设计拆分字段及拆分规则时，需要注意拆分后数据的均衡性，避免出现数据不均衡从而导致数据热点。
> - 数据访问代理支持多列拆分字段，即有多个拆分字段组成一个拆分规则，但是不推荐使用该方式。如果使用了多列拆分字段，后续的 SQL 执行均需要带上多列查询条件，对业务 SQL 的使用也会比较复杂。

15.4. 配置数据访问代理连接

连接是数据库的重要资源，合理的使用数据库连接资源能使应用程序的资源利用达到最佳状态。

数据访问代理实例内部一共有两种类型的连接，分别是：

- 前端连接：应用程序代码连接数据访问代理实例的连接配置。
- 后端连接：数据访问代理实例连接后端真实物理数据库的连接配置。

前端连接

前端连接的数量理论上仅受限于数据访问代理实例节点可用的内存大小和网络连接数。但在实际的应用场景中，应用程序连接到数据访问代理实例时，通常会管理有限数量的连接来执行请求的操作，并不会维持很高并发量的持久化长连接（例如，数万个并发的长连接），因此可认为数据访问代理实例能接受的前端连接数量是无限制的。

由于前端连接数量不受限制，可以允许有大量空闲连接存在，因此适用于业务端部署应用程序的服务器数量较多，需要同时保持连接到数据访问代理实例的场景。

> **重要**
>
> 虽然前端连接的数量可被认为是无限制的，但从前端连接获取的操作请求是由数据访问代理实例的内部线程通过后端连接实际执行，而内部线程和后端连接的数量有限，因此数据访问代理实例处理请求的整体并发度是有限的。

应用与数据访问代理的连接底层实现是通过 SLB 创建的，而 SLB 有 900 s 的空闲超时机制，即 900 s 内连接没有流量的话，连接会被断开。所以应用使用时可以设置连接池的空闲超时时间小于 900 s，或者通过心跳方式保持连接。

后端连接

数据访问代理实例的每个节点内部都会创建后端连接池，自动管理和维护到 RDS/OceanBase 实例中物理库的后端连接。因此，数据访问代理实例中后端连接池的最大连接数与具体数据库实例支持的最大连接数直接相关。

- 您可参照以下公式来计算数据访问代理实例中后端连接池的最大连接数：

  数据访问代理实例后端连接池的最大连接数 = 向下取整（数据库最大连接总数 / 物理分库数 / 数据访问代理实例节点数），例如某用户搭配购买了如下规格的 RDS 实例和数据访问代理实例：

  - 2 个 RDS 实例，包含 100 个物理分库，规格为 4C16G × 2，最大连接数为 4000 × 2；
  - 1 个数据访问代理专享实例，规格为 16C32G（每 4C8G 为 1 个数据访问代理节点，即该实例包含 4 个数据访问代理节点）。

- 按照以下公式可计算出数据访问代理实例中后端连接池的最大连接数：

  数据访问代理实例后端连接池的最大连接数 = 向下取整（8000 / 100 / 4）= 20

  > **说明**
  >
  > 最大连接数的评估仅是为了知道最大连接数的设置上限，实际日常使用中，为了减轻数据库实例的连接压力，建议留出一定的缓冲余地，使实际最大连接数小于数据库的上限值。

如何调整后端连接配置

请参考使用文档：管理物理数据库连接参数。

15.5. 配置合理的连接参数

数据访问代理服务端内部使用连接池访问物理数据库。合理的连接参数配置不仅有助于提升数据访问性能，而且能够保障数据库的稳定性。

以下是一些常见的服务端连接参数：

- **超时时间**
  - `connectTimeout`：创建数据库连接超时时间。
  - `socketTimeout`：SQL 请求超时时间，操作系统 socket 的超时时间，即一次 SQL 请求如果在该时间内没有返回任何数据，操作系统会抛出 Read timeout 的异常。
  - `txTimeout`：数据访问代理事务超时时间，默认为 10 分钟。该参数是为了保护 ODP 的连接池不会被长事务占满，而影响业务的正常访问。目前该值不能修改，后续提供业务自定义配置。
- `idleTimeoutMinutes`：连接空闲时间，超过该时间则会被剔除出连接池，默认为 12 分钟。
- **其他参数**：其他高级连接参数，可以在 `connectionProperties` 通过 KV 键值对进行配置。
- **连接数量**：
  - `min`：单个分库的最小连接数。如果 `min` > 0，启动时会自动创建 `min` 个连接。
  - `max`：单个分库的最大连接数。

批量任务导入导出配置

当您需要进行批量导入/导出任务时，您可以对以下参数进行配置。

- 设置 `connectionProperties` 中的 `socketTimeout=1000000` 参数，该参数是指指从连接池获取到连接的超时时间，推荐设置为较大值，如 1000000（16 分钟），单位为毫秒。
- 设置 `connectionProperties` 中的 `rewriteBatchedStatements=true` 参数，该参数可以帮您批量执行SQL，需要导入数据的时候推荐设置为 true，这样批量插入时多条 `insert sql` 会被改写为一条。
- 设置 `newConnectionSql` 中的 `set @@ob_query_timeout=2000000000` 和 `set @@ob_trx_timeout=2000000000` 参数，当使用 OceanBase 数据库的时候需要设置这两个值，单位是微秒。

并且此时数据库用户需要使用 odp_migrator 这个账号，当使用此账号时会启用 ODP 内部的流式结果集读取，不会因为单条 SQL 的结果集太大而把 ODP 的内存打爆。

连接池配置

数据库连接池

应用与数据库之间的连接建立和销毁是个大消耗的操作，为了避免每一个请求都进行此操作，于是有了数据库连接池。它的基本原理是连接建立后，先不销毁，而是放回连接池，当有新的请求，不再创建新的连接，而是从连接池中去申请获取已有连接直接使用，这样就避免了频繁的连接创建和销毁。连接池有两个最关键的参数：

- **最小连接数**：连接池中始终维持的最小连接数。
- **最大连接数**：允许创建的最大连接数。

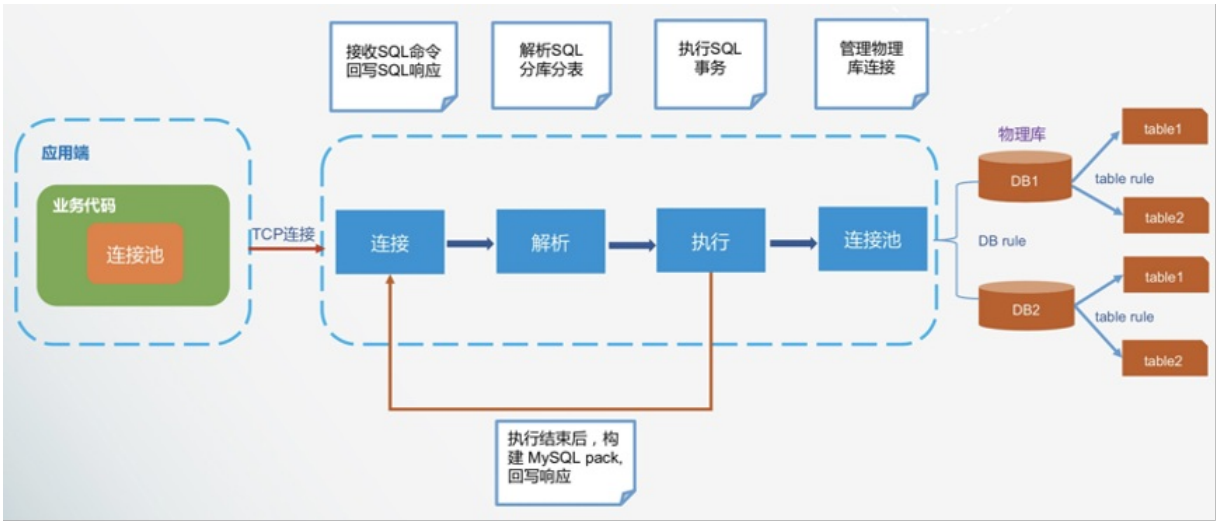
合理的设置这两个参数才能最大化连接池的好处。如果最小连接数设置得过大，容易造成连接过剩；如果最大链接数设置得过大，有可能超过数据库的处理能力。由于连接池是无法感知数据库的运行情况以及负载的，所以建议通过以下两种办法来使得参数值趋于合理：

- **理论计算模型估算基准值**：连接池本质是个排队系统，因此可以根据利特尔法则（Little's Law）来计算连接数。连接池中的连接数（L）= 单位时间内的 SQL 请求量 * SQL 请求耗时（W），最开始可以以此估算一个值作为基准。

- **利用压测和实际运行监控调整**：QPS 和请求耗时需要应用运行时持续监控才能得出更合理的值，同时还需利用压测来模拟大业务量和高并发场景，结合两种情况进行调整得到合理值。

ODP 连接池

在应用使用数据访问代理 ODP 进行分库分表的场景里，整体工作链路如下：



示例

假设业务的预估值是：QPS 是 10000；请求耗时日常是 2ms，最高是 10ms。

根据前面提到的利特尔法则，需要的连接数 10000*0.002=20 个，最高为 10000*0.01=100 个，也就是最小连接数（MIN）20，最大连接数（MAX）100。那么对于应用侧的连接池，假设应用节点个数为 4，连接数则可以设置为 MIN=20/4=5，MAX=100/4=25。

ODP 获取请求后，期望是无缝转发给数据库，所以需要的最小连接数与应用一致最小是 20，最大连接数则还需额外考虑两个因素：

- **分库数量**：考虑到最高耗时，同样最大连接数为 100 个。
- **数据库规格**：根据 10000 QPS，至少需要一个 m1-medium 的 RDS 实例，对应的最大连接数是 4000。

综合考虑，最大连接数一般为基于 QPS 计算出的最大值的两倍，并且必须需要小于数据库规格最大值，因此建议为 100*2=200。由于 ODP 管控界面上的最小最大连接数是对应到的是单个 ODP 节点针对单个库的连接池配置，所以对于 ODP 侧的连接池，假设 ODP 实例节点个数为 2，连接数则可以设置为 MIN=20/10/2=1，MAX=200/10/2=10。

另外，如果业务侧有大量使用 SCAN_ALL Hint 的场景，此时 QPS 在 ODP 侧将会被放大“分库数量”倍，最小、最大连接数也要相应乘以分库数量（考虑性能因素应该尽量规避 SCAN_ALL）。

计算模型

总结下来，我们可以得出连接数的计算模型：

- **应用侧**：

  最小连接数 =（QPS*请求平均耗时）/ 应用节点个数。

  最大连接数 =（QPS*请求最大耗时）/ 应用节点个数。

- **ODP 侧**：

  最小连接数 =（QPS*请求平均耗时）/ ODP 节点个数。

最大连接数 = (QPS*请求最大耗时*2) / ODP 节点个数 （并且需要保证“最大连接数*分库数量*ODP 节点个数” 要小于数据库所支持的最大连接数量）。

> **说明**
>
> 在实际中，应用侧的请求耗时肯定要大于 ODP 侧的请求耗时，因为中间需要经过 ODP 处理，也就是说应用侧的请求耗时需要以应用记录为准，而 ODP 侧的则以 ODP 日志或者数据库日志记录为准。

15.6. 使用应用连接池

将应用和数据库连接进行业务操作，建议使用连接池。如果是 Java 程序，推荐使用 Druid 连接池。

1. 项目工程接入 Druid 示例：

```
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.0.31</version>
</dependency>
```

2. 配置 Druid 示例：

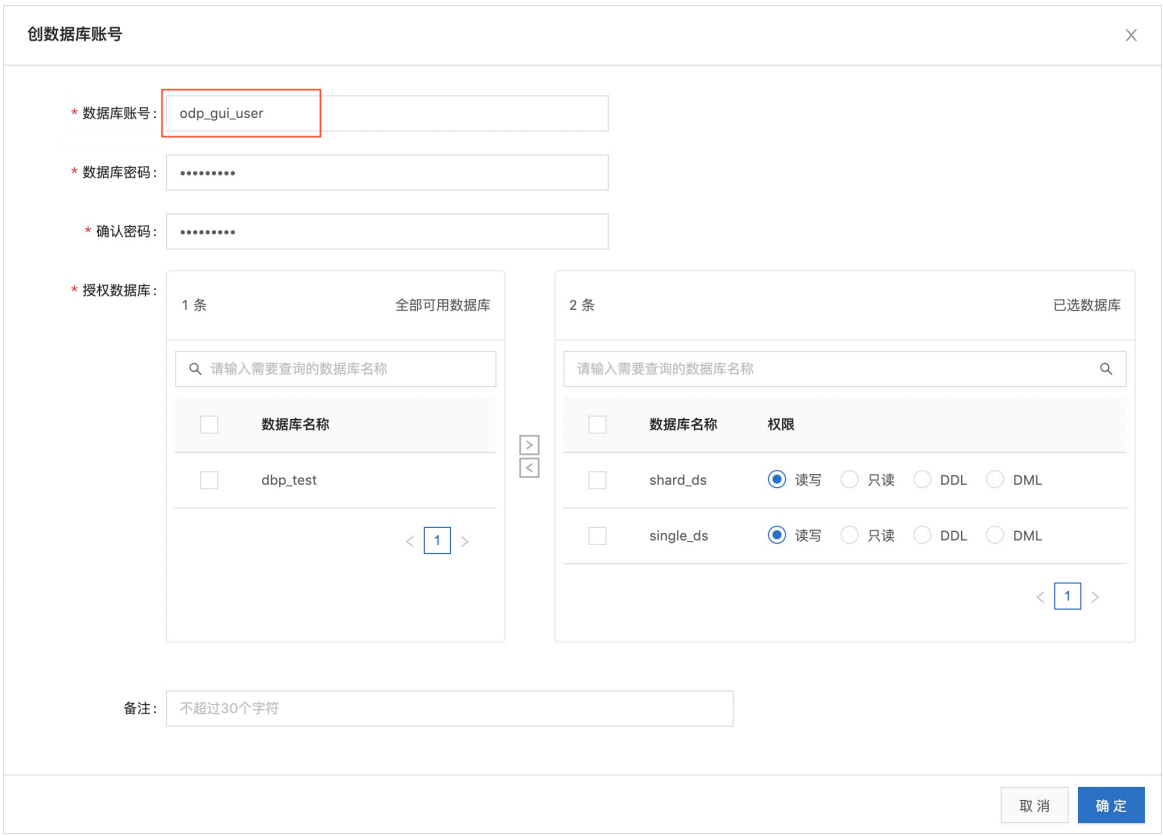
```
<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" d
estroy-method="close">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
    <!-- 基本属性 URL、user、password -->
    <property name="url" value="jdbc:mysql://ip:port/db?socketTimeout=30000&amp;connectTi
meout=3000"/>
    <property name="username" value="{user}"/>
    <property name="password" value="{password}"/>
    <!-- 配置初始化大小、最小、最大 -->
    <property name="maxActive" value="4"/>
    <property name="initialSize" value="2"/>
    <property name="minIdle" value="2"/>
    <!-- 获取连接等待超时的时间，单位是毫秒 -->
    <propertyname="maxWait" value="1000"/>
    <!-- 间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->
    <property name="timeBetweenEvictionRunsMillis" value="60000"/>
    <!-- 一个连接在池中最小空闲的时间，单位是毫秒-->
    <property name="minEvictableIdleTimeMillis" value="300000"/>
    <!-- 检测连接是否可用的 SQL -->
    <property name="validationQuery" value="SELECT 1"/>
    <!-- 是否开启空闲连接检查 -->
    <property name="testWhileIdle" value="true"/>
    <!-- 是否在获取连接前检查连接状态 -->
    <property name="testOnBorrow" value="false"/>
    <!-- 是否在归还连接时检查连接状态 -->
    <property name="testOnReturn" value="false"/>
</bean>
```

15.7. 连接 Navicat 客户端

数据访问代理不仅支持应用程序（如 JDBC）与 MySQL Client 命令行访问，也支持 Navicat 数据库图形工具访问。本文将介绍如何配置数据访问代理与 Navicat 的连接，实现 Navicat 客户端访问。

创建数据库账号

1. 进入数据访问代理控制台，左侧导航栏中，单击 **实例**。
2. 在实例列表中，单击目标实例名，进入该实例的详情页面。
3. 单击 **账号管理** 页签，然后单击 **创建账号** 按钮。
4. 在 **创建数据库账号** 对话框中，配置账号信息：
   - **数据库账号**：账号名称，此处必须填写 `odp_gui_user`。
   - **数据库密码**：为该数据库账号设置一个密码。
   - **授权数据库**：选择需要授权的数据库并设置相应的权限。



5. 单击 **确定**。

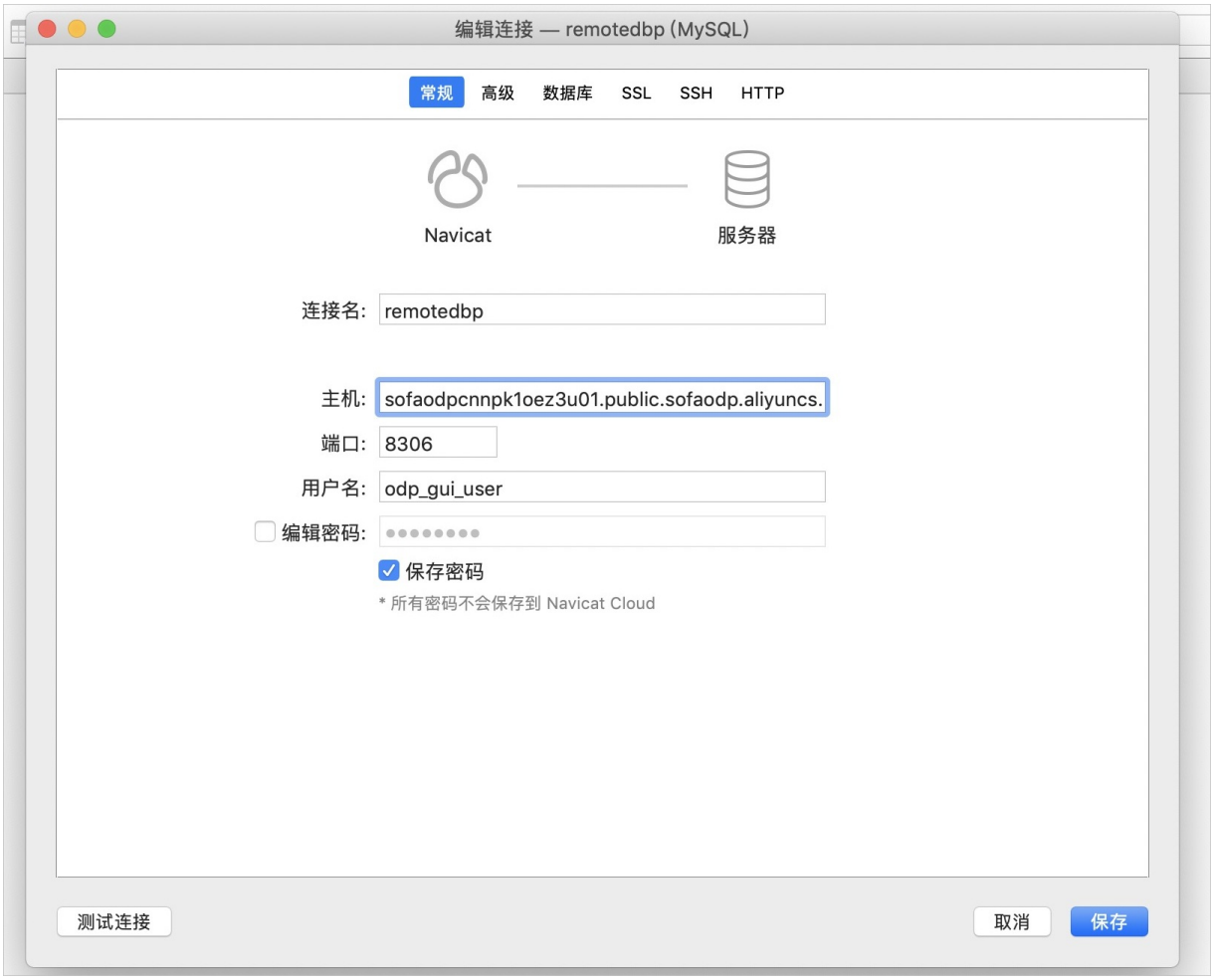
配置 Navicat 连接

打开 Navicat 客户端，配置一个连接。连接信息配置要求如下：

- **连接名**：可根据业务需求，自定义连接名称，如 `remotedbp`。
- **主机**：输入要连接的数据访问代理实例的访问地址，一般是一个公网地址。
- **端口**：输入要连接的数据访问代理实例的访问端口。

> **说明**
>
> 您可以在数据访问代理控制台的实例详情页申请并获取公网地址。详见 公网地址。

- **用户名/密码**：输入上一步创建的数据库账号名（即 `odp_gui_user`）与密码。



功能说明

数据访问代理与 Navicat 完成连接后，在本地 Navicat 客户端中，支持与不支持的操作分别如下：

- 支持访问授权的数据库
- 支持双击打开数据表（单库单表、分库分表均支持）
- 支持查询数据表结构（单库单表、分库分表均支持）
- 支持在 SQL 窗口手动执行查询命令（单库单表、分库分表均支持）
- 支持双击打开数据表后，插入、更新、删除数据（单库单表、分库分表均支持）
- 支持双击打开数据表后，进行排序或筛选（单库单表、分库分表均支持）
- 支持部分 DDL：
  - 不支持库操作
  - 不支持单库单表的所有 DDL

- 分库分表的建表仅支持 sharding DDL
- 支持分库分表的 alter、drop 等

- 不支持视图，包括查看与更新
- 不支持事务
- 不支持清空表
- 不支持库级别的“运行 SQL 文件”

注意事项

在 **双击打开数据表** 与 **SQL 窗口查询** 场景中，如果没有手动加 hint，那么数据访问代理后台都会默认加上一个 `SCAN_ALL` 的 hint，表示 **全表扫描**，以保证在没有分库分表位的查询中有着良好的用户体验。

但需要注意的是，查询时，请勿在数据量很大的时候不加任何筛选条件地进行全表扫描，否则可能会给数据访问代理后台和物理库造成较大压力，引起业务系统查询耗时增长，甚至崩溃。

15.8. 基于 DataX 完成数据访问代理数据迁移

数据访问代理（Open Database Proxy，简称 ODP）通过集成 DataX，支持全量离线静态的数据迁移功能。DataX 是阿里巴巴集团内被广泛使用的离线数据同步工具/平台，实现各种异构数据源之间高效的数据同步。目前，支持的源端数据源类型依赖于 DataX 支持的类型，而目标端仅支持 MySQL 和 OceanBase。

本文将引导您快速完成以下迁移任务：

- 从单库单表 MySQL 迁移至 ODP
- 从 ODP 迁移至其他数据库

单库单表 MySQL 迁移至 ODP

前置条件

您已经开通数据访问代理产品并购买了数据访问代理实例，详见 创建实例。

建库建表

1. 登录数据访问代理控制台页面，选择 **数据库**，单击 **创建数据库**。
2. 在弹出的 **创建数据库** 窗口中，选择实例，单击 **创建**。
3. 进入 **选择数据节点** 页面后，根据需要选择 MySQL 或 OceanBase 的节点，单击 **下一步**。
4. 根据提示，填写或选择数据库的基本信息。详细参数信息，请参见 创建数据访问代理数据库。
5. 数据库创建完成后，进入数据库详情页，单击右侧页面下方的 **新增数据表**。
6. 在 **新增数据表** 页面，输入 **数据表名**，如 `test_migration_user`。
7. 根据需要选择表类型（单表或拆分表），此处以 **拆分表** 为例，并设置 **分表总数** 为 2。
8. 配置分表规则，详细配置方法可参见 创建数据访问代理数据表 和 自定义分表规则。
9. 勾选 **现在创建物理表**，单击 **下一步**。
0. 在 **DDL 语句** 中，输入建表语句，示例如下：

```
CREATE TABLE `test_migration_user`(
     `user_id` varchar(16) NOT NULL,
     `username` varchar(64) NOT NULL,
     `id` bigint(20) NOT NULL AUTO_INCREMENT,
     PRIMARY KEY (`id`),
     UNIQUE KEY `uk_user_id`(`user_id`)
) ENGINE=InnoDB AUTO_INCREMENT=102 DEFAULT CHARSET=utf8mb4;
```

1. 单击 **执行**，完成建表。

创建迁移账户

1. 在数据访问控制台页面，左侧导航栏选择 **实例**。
2. 在实例列表中，找到刚刚创建的数据库所在的实例，单击实例名称，进入实例详情页。
3. 切换至 **账号管理** 页签，单击 **创建账号**。
4. 在弹出的新窗口中，输入数据库账号（必须是 odp_migrator）和账号密码，选择相应的授权数据库。
5. 单击 **确定**，完成创建迁移账户。

配置连接参数

1. 进入数据库详情页，选择页面下方的 **连接参数** 页签。
2. 切换至 **其他参数** 后，在 connectionProperties 属性栏，输入以下属性值：

```
rewriteBatchedStatements=true
clobberStreamingResults=true
```

3. 单击底部 **保存配置** 按钮。

准备 DataX 环境

DataX 是一个异构数据源间数据迁移的中间件，支持各种类型的 reader 和 writer。要实现 ODP 数据迁移，您需要准备 DataX 环境。

1. 单击此处 下载 DataX，并将其解压至本地目录。
2. 单击此处 下载 odpwriter 插件，并将其解压至 `$datax_dir/plugin/writer` 目录下。

有关 DataX 的更多信息，参见 DataX 官方文档。

编辑并执行迁移 Job

1. 进入 `$datax_dir/bin` 目录，编写一份迁移 Job 描述文件。文件示例如下：

```
{
 "job": {
     "setting": {
         "speed": {
                 "channel": 3
         },
         "errorLimit": {
             "record": 0,
             "percentage": 0.02
         }
     },
     "content": [
```

```
        {
            "reader": {
                "name": "mysqlreader",
                "parameter": {
                    "username": "$mysql_usr",
                    "password": "$mysql_pwd",
                    "column": [
                        "user_id",
                        "username"
                    ],
                    "splitPk": "id",
                    "connection": [
                        {
                            "table": [
                                "user"
                            ],
                            "jdbcUrl": [
                                "jdbc:mysql://127.0.0.1:3306/test_migration"
                            ]
                        }
                    ]
                }
            },
            "writer": {
                "name": "odpwriter",
                "parameter": {
                    "writeMode": "replace",
                    "username": "odp_migrator",
                    "password": "$odp_pwd",
                    "column": [
                        "user_id",
                        "username"
                    ],
                    "connection": [
                        {
                            "jdbcUrl": "jdbc:mysql://$dbp_url/test_migration?useUnicode=
true&characterEncoding=utf8",
                            "table": [
                                "test_migration_user"
                            ]
                        }
                    ]
                }
            }
        }
    ]
}
}
```

根据以上 Job 描述文件，该数据迁移任务是将单库单表的 MySQL（`127.0.0.1:3306`）中的`test_migration`库里的`user`表的数据（列`user_id`，`username`）迁移至`$dbp_url`中的`test_migration`库里的`test_migration_user`表（列`user_id`，`username`）。

2. 在`$datax_dir/bin`目录下，执行以下命令，运行数据迁移 Job：

```
python datax.py $yourjob.json
```

3. 命令执行成功，即数据迁移完成。

ODP 迁移至其他数据库

从 ODP 迁移至其他数据库，操作步骤与 从 MySQL 迁移至 ODP 相似。

1. 参考上文，完成 建库建表、创建迁移账户、配置连接参数操作。
2. 按照以下步骤，完成 DataX 环境准备、Job 描述文件的编写与执行。

准备 DataX 环境

1. 单击此处 下载 DataX，并将其解压至本地目录。
2. 单击此处 下载 odpreader 插件，并将其解压至 `$datax_dir/plugin/reader/` 目录下。

执行 Job 描述文件

1. 进入 `$datax_dir/bin` 目录，编写一份迁移 Job 描述文件。文件示例如下：

> **说明**
>
> - Reader 迁移账号也必须是 odp_migrator。
> - 对 Job 描述文件添加一个 connection 的校验，仅允许配置一个 connection，且仅允许配置一张表。

```
{
 "job": {
     "setting": {
         "speed": {
             "channel": 4
             },
         "errorLimit": {
             "record": 0,
             "percentage": 0.02
             }
     },
     "content": [
         {
             "reader": {
                 "name": "odpreader",
                 "parameter": {
                     "username": "odp_migrator",
                     "password": "$odp_pwd",
                     "column": [
                         "user_id",
                         "gmt_create",
                         "gmt_modified",
                         "current_date",
                         "current_time",
                         "current_timestamp",
                         "name",
                         "is_ok",
                         "age",
```

```
                            "salary",
                            "height",
                            "memo",
                            "character_stream",
                            "binary_stream",
                        ],
                        "connection": [
                            {
                                "table": [
                                    "test_migration_user_large"
                                ],
                                "jdbcUrl": [
                                    "jdbc:mysql://$odp_url:8066/test_join?useUnicode=true&ch
aracterEncoding=utf8"
                                ]
                            }
                        ]
                    }
                },
                "writer": {
                    "name": "odpwriter",
                    "parameter": {
                        "writeMode": "replace",
                        "username": "odp_migrator",
                        "password": "$odp_pwd",
                        "column": [
                            "`user_id`",
                            "`gmt_create`",
                            "`gmt_modified`",
                            "`current_date`",
                            "`current_time`",
                            "`current_timestamp`",
                            "`name`",
                            "`is_ok`",
                            "`age`",
                            "`salary`",
                            "`height`",
                            "`memo`",
                            "`character_stream`",
                            "`binary_stream`",
                        ],
                        "batchSize": 2048,
                        "connection": [
                            {
                                "jdbcUrl": "jdbc:mysql://$odp_url:8066/test_join?useUnicode=
true&characterEncoding=utf8",
                                "table": [
                                    "test_migration_user_large_target"
                                ]
                            }
                        ]
                    }
                }
            }
```

```
        ]
    }
}
```

2. 在 `$datax_dir/bin` 目录下，执行如下命令，运行数据迁移 Job。

```
python datax.py $yourjob.json
```

3. 命令执行成功，即数据迁移完成。

注意事项

不推荐使用 `SCAN_ALL` hint + mysql reader 的方式进行数据读取。虽然这种方式也可以对分库分表的数据表进行数据抽取。但其数据拆分方式为 `splitPK`，即全表扫描之后，根据 ID 进行拆分，拆分之后的每个子任务 $from ~ $to 同样需要全表扫描才能取出。这就意味着，任务拆分之后，每个子任务的查询都是一次全表扫描。因此，这种方式性能差、容易超时，而且会对数据访问代理实例产生较大的负载压力，不建议使用。

相比之下，odpreader 的拆分方式是基于逻辑表的分表来实现的，如百库百表下会拆分为 100 个子任务，每个子任务的数据抽取仅会查询单个分表，并不会产生全表扫描。因此，建议您使用 odpreader。

常见问题

DataX 支持的数据类型

DataX 的 MysqlReader 针对 MySQL 类型转换列表如下：

DataX 内部类型	MySQL 数据类型
long	int, tinyint, smallint, mediumint, int, bigint
Double	float, double, decimal
String	varchar, char, tinytext, text, mediumtext, longtext, year
Date	date, datetime, timestamp, time
Boolean	bit, bool
Bytes	tinyblob, mediumblob, blob, longblob, varbinary

说明

- 除上述字段类型外，其他类型均不支持。
- 对于 `tinyint(1)`，DataX 视为整型。
- 对于 `year`，DataX 视为字符串类型。
- 对于 `bit`，DataX 视为未定义行为。

事务与出错重跑

在数据迁移过程中，可能存在部分数据写入失败的情况，此时需要进行重试。由于此时可能已经有部分数据已经完成迁移，重试的时候回出现 Duplicate Key 问题。此时，可以使用 DataX 的 REPLACE/INSERT IGNORE writeMode。

需要注意，REPLACE 语句的行为是如果发现主键或唯一键冲突，会将原来这条数据删除，然后重新插入，这样既可实现比较简单的出错重跑机制。REPLACE 语句可能会导致新的数据记录的自增 id 有变化，如果使用自增 id 进行关联，会出现问题。而 INSERT IGNORE 语句则不会有这个问题，如果发现主键或唯一键冲突，会忽略当前这条 SQL 的插入。

超时问题

SocketTimeout 错误

现象：

出现 SocketTimeout 报错，类似的错误信息如下所示：

```
Thelast packet successfully received from the server was 5,004 milliseconds ago.Thelast pac
ket sent successfully to the server was 5,004 milliseconds ago.
```

解决方案：

1. 登录数据访问代理控制台，进行目标数据库的详情页。
2. 选择页面下方的 **连接参数** 页签，在 **其他参数 > connectionProperties** 属性栏，添加如下属性：

   ```
   socketTimeout=5000（调整为更大的数值，默认为5秒，单位为毫秒）
   ```

3. 单击 **保存配置**。

Timeout 错误

现象：

出现 Timeout 报错，对应的错误码为 4012。

原因：

该错误是 OceanBase 查询超时问题。

解决方案：

您可以通过以下方法处理该错误：

- 在任意一个 column 中加入 hint：`/*+空格QUERY_TIMEOUT(111111111)空格*/ $columnName`，示例如下：

```
"column":[
"/*+ QUERY_TIMEOUT(100) */ user_id",
"username",
"id",
"gmt_create",
],
```

> **说明**
>
> 该超时时间仅影响当前SQL，并不会影响其他SQL。

- 在 newConnectionSql 中添加 `ob_query_timeout` 参数，单位为微秒。

```
set@@ob_query_timeout=比较大的值
```

> **说明**
>
> 该参数配置将影响整个逻辑库。在离线与在线业务混用 ODP 实例的场景中，不推荐使用这种参数配置。可以在直连物理库时执行这条 SQL，检查该超时参数设置为多大值时不会超时，然后在此基础上再加一些 buffer。

Communications link failure 错误

现象：

出现 Communications link failure 报错，错误信息如下所示：

```
Communications link failure

Thelast packet successfully received from the server was 0 milliseconds ago.Thelast packet
sent successfully to the server was 605,920 milliseconds ago.

com.mysql.jdbc.exceptions.jdbc4.CommunicationsException:Communications link failure

Causedby: java.io.EOFException:Cannot read response from server.Expected to read 425 bytes,
read 156 bytes before connection was unexpectedly lost.
```

原因：

该错误是因为数据库达到了 `net_write_timeout` 而主动断连导致的，即数据库回写超时。

解决方案：

数据迁移（流式读取）时默认值为 600 秒，您可以通过设置以下 ODP 连接参数，解决该问题。

```
netTimeoutForStreamingResults=大于600的一个数值，单位是秒
```

15.9. 配置同城双活模式

数据访问代理默认不支持同城双活模式，您需要进行手动配置。本文将介绍如何为数据访问代理配置同城双活模式。

操作步骤

1. 在工程 pom.xml 文件中，添加以下 Maven 依赖。

```
<dependency>
     <groupId>com.alipay.sofa</groupId>
     <artifactId>dbp-connector-java</artifactId>
     <version>1.0.8</version>
</dependency>
```

2. 根据以下示例，修改 Spring 配置文件。

```
<!-- vip寻址-->
<bean id="dbpDiscovery" class="com.alipay.sofa.dbp.discovery.DbpDiscovery"/>

<!-- dbp-connector代理-->
<bean id="delegatingDataSource" class="com.alipay.sofa.dbp.DbpDataSource" init-method="in
it">
    <property name="delegate"ref="simpleDataSource"/>
    <property name="appName" value="${yourAppName}"/>
    <property name="database" value="${yourDatabase}"/>
    <property name="dbpInstanceId" value="${yourDbpInstanceId}"/>
    <property name="clientTracer" ref="clientTracer"/>
</bean>

<!-- 连接池配置，以 druid 为例 -->
<bean id="simpleDataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="i
nit">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql://${yourDbpInstanceId}:8306/${yourDatabase}"/>
    <property name="username" value="${username}"/>
    <property name="password" value="${password}"/>
</bean>

<bean id="clientTracer" class="com.alipay.sofa.dbp.DbpClientTracer"/>
```

> **说明**：目前支持的连接池包括 Druid、DBCP、C3P0、Tomcat。

3. 排除不需要自动寻址的 DataSoure。

```
<!-- 有些数据源需要直连数据库或者类似 spring 的 AbstractRoutingDataSource-->

<bean id="dbpDiscovery" class="com.alipay.sofa.dbp.discovery.DbpDiscovery">
       <property name="excludeDataSources">
            <list>
                 <value>dynamicDataSource</value>
            </list>
       </property>
</bean>
```

4. 如果 Spring 配置文件中配置了事务，事务中的 DataSource 配置需要使用 dbp-connector 代理 DataSource，即第 3 步 Spring 配置文件中的 `delegatingDataSource` 。

15.10. 单元化配置

数据访问代理支持单元化功能。在单元化部署架构下，您需要特别注意数据节点的配置以及配置数据库时各分库对应的账户。

> **说明**
>
> 该功能仅适用于支持 LDC 单元化架构的环境。

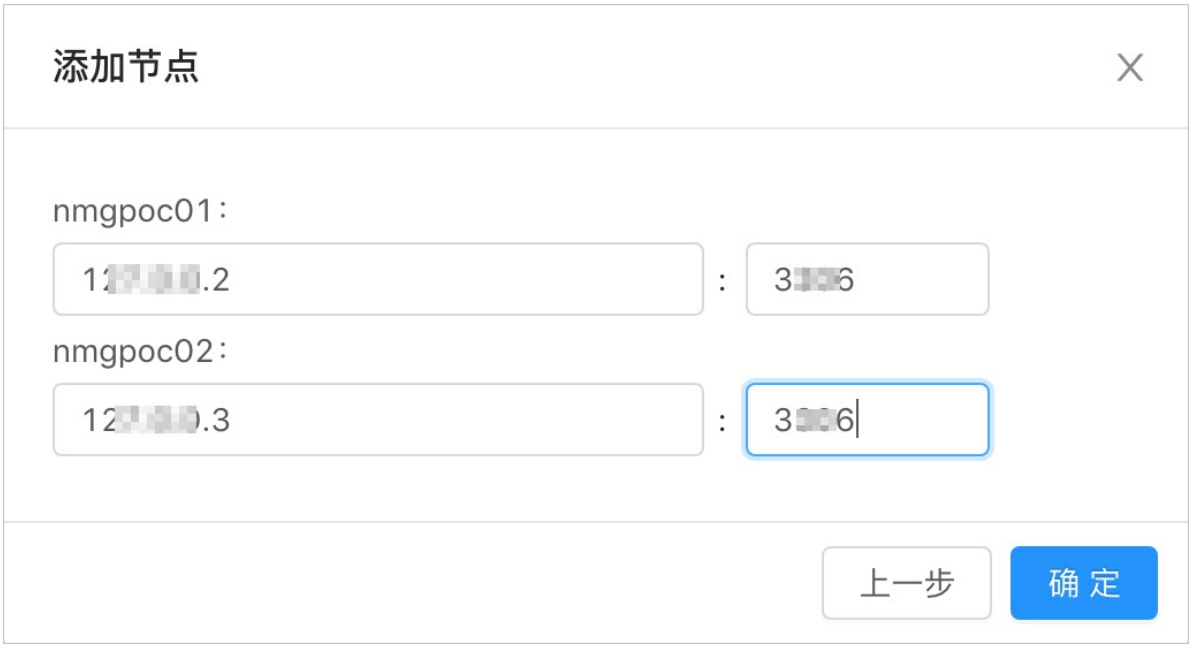
配置数据节点

在数据访问代理配置数据库前，需要添加物理数据库节点，在单元化部署架构下，数据库（OceanBase）会提供三个访问地址。其中两个为只能访问当前机房数据库节点的地址，即 local url。另外一个是可以跨机房访问所有数据库节点的地址，即 global url。

在添加数据节点时，您需要填写以上三个访问地址，操作步骤如下：

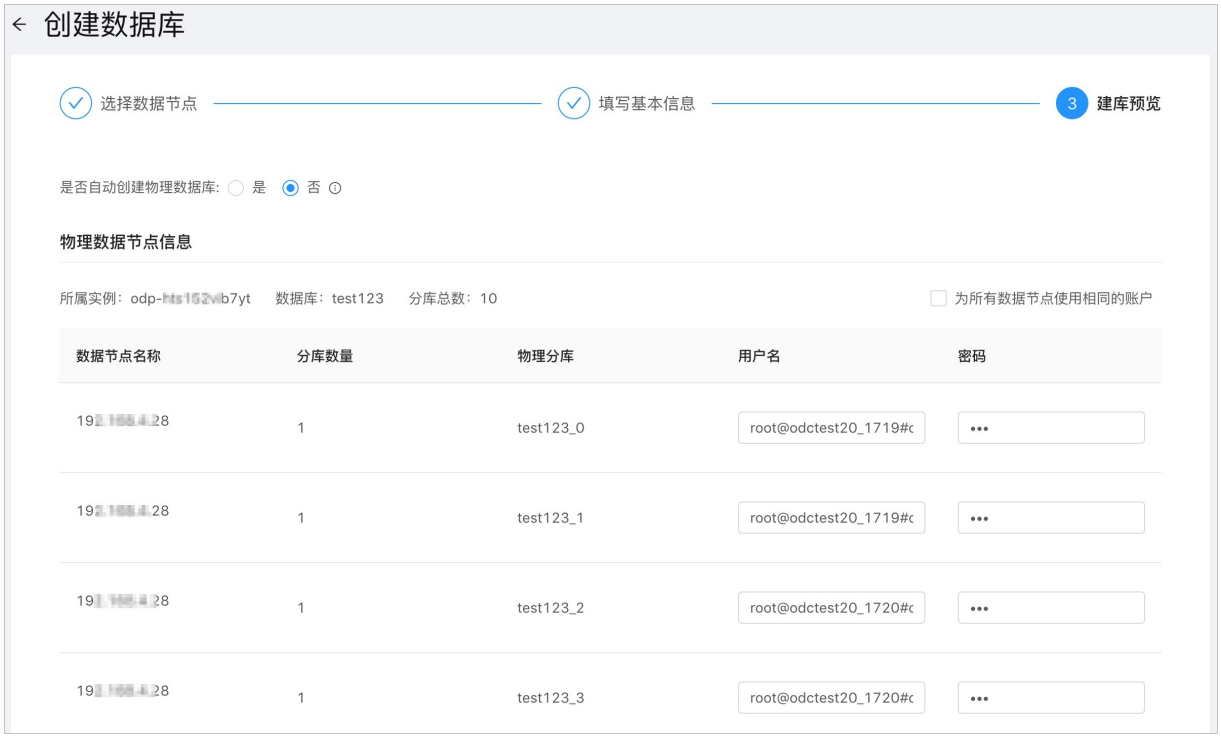
1. 进入数据访问代理控制台页面，左侧导航栏选择 **运维 > 物理数据节点**。
2. 单击 **添加节点**，在新窗口中，添加 global url，单击 **下一步**。

3. 继续添加 local url，单击 **确定**。



配置数据库

在创建配置数据库时，需要根据数据库的单元化规划，填写分库对应的账户，如下图所示。



15.11. 使用双机房 ODP 实例（阿里云版）

本文以搭建阿里云双机房数据访问代理（ODP）实例访问 OceanBase 为例，介绍如何使用阿里云双机房下的 ODP。

> 说明
>
> 本文以上海非金区（华东 2）为例，其他地域的操作步骤类似。

示例背景

部署拓扑：应用部署于上海非金可用区 A 与可用区 C，每个可用区的应用承载 50 个分片的流量（依赖流量调拨）；同时创建部署于可用区 A 与可用区 C 的 ODP 实例（分别映射到可用区 A 与可用区 C 的交换机），应用连接当前可用区内的 ODP 实例。

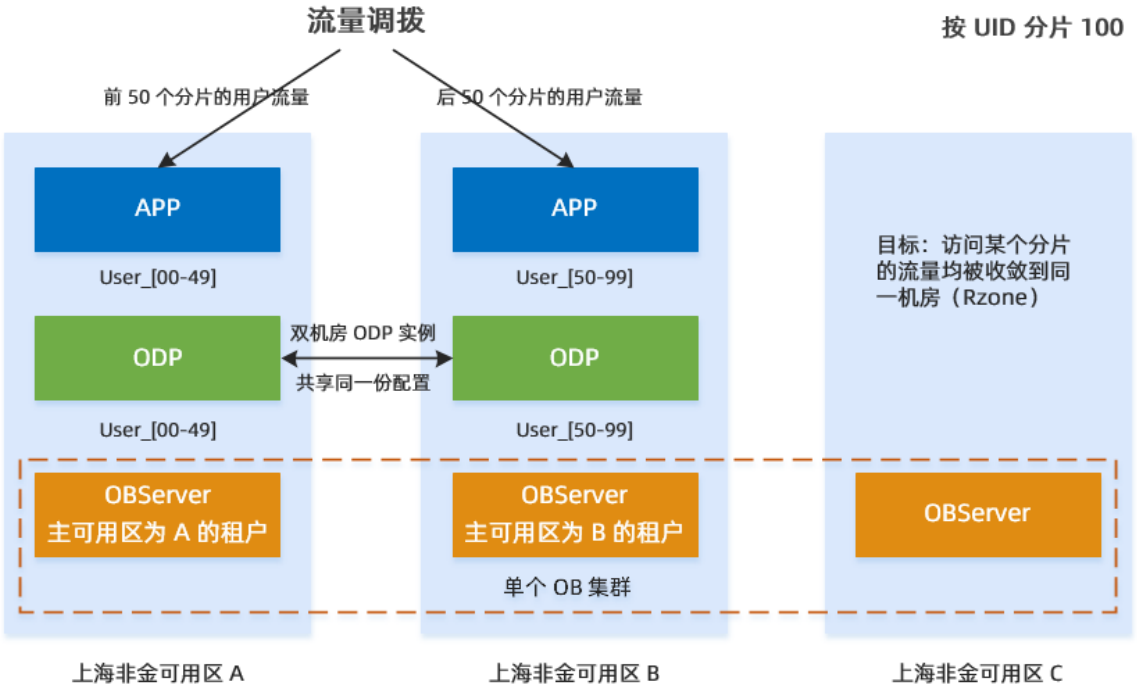
数据拓扑：共分 100 个分片，即百库百表。使用公有云 OceanBase 的单集群双租户，每个租户分配 50 分片。其中，第一个租户的主可用区设置为可用区 A，分配前 50 分片；第二个可用区的主可用区设置为可用区 C，分配后 50 分片。

> 说明
>
> 前 50 分片的 leader 所在可用区 A 宕机后，OB 会将该 50 分片的 leader 切换至可用区 B

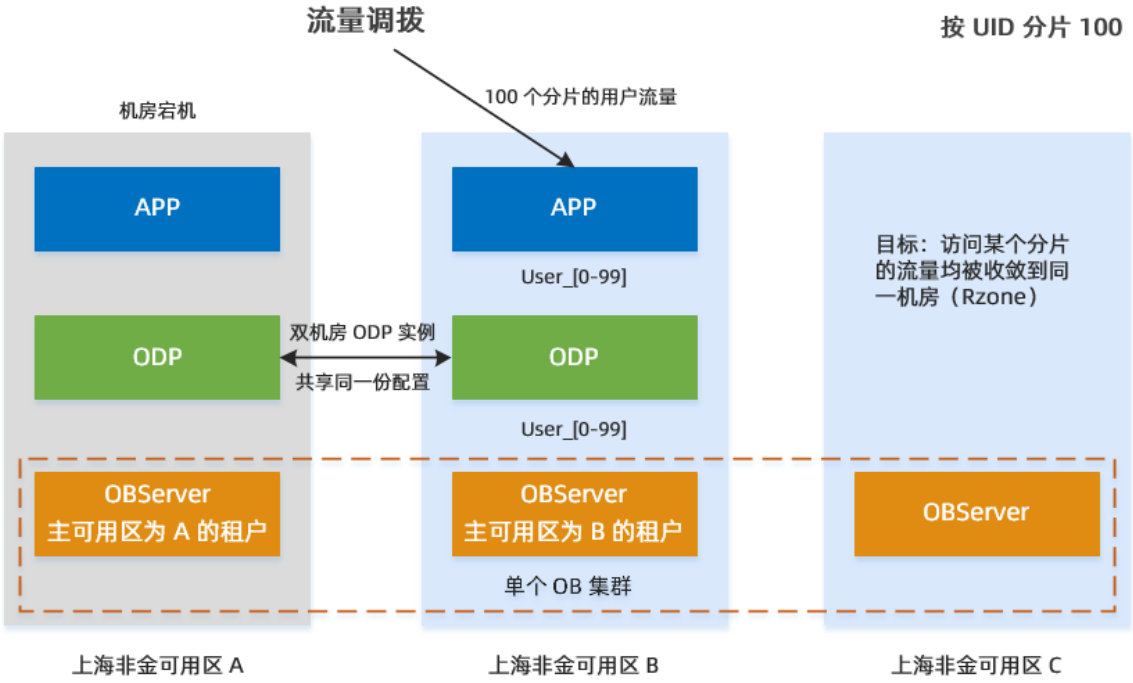
此时，前 50 分片的流量会调拨给 A 可用区内的应用，应用连接当前可用区 A 的 ODP（未产生跨机房），ODP 会连接主可用区在可用区 A 的 OceanBase 租户，访问该租户内的前 50 个分片物理库表（同样未产生跨机房）。

现实情况中，由于上海非金地域中 ODP 实例仅支持可用区 A、C、D，暂时不支持部署在可用区 E、F、G，而 OceanBase 目前仅支持机房 E、F、G 作为主可用区，因此暂时匹配不上，从而出现跨机房访问。

对于三可用区的 OB 集群而言，每个可用区均分布一台 OBServer。在主可用区为 A 的租户中创建的所有分区，其 leader 均为可用区 A；如某个分区的 leader 为可用区 A，则读写该分区时都默认访问位于可用区 A 的 OBServer。



前 50 分片的 leader 所在可用区 A 宕机后，OB 会将该 50 分片的 leader 切换至可用区 B，并且应用流量也会全部调拨到可用区 B。此时，依然可以为 100 个分片提供数据访问服务（OB 多副本机制）。



前置条件

- 开通数据访问代理（ODP）组件服务。
- 在华东 2（上海）地域创建一个专有网络 VPC，并在上海可用区 A 和可用区 C 分别创建一个虚拟交换机 vSwitch。详见 搭建专有网络。

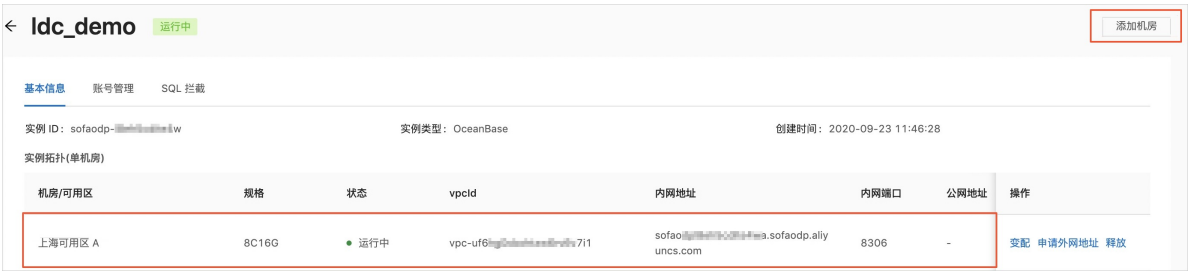
购买 ODP 实例

1. 登录 ODP 控制台页面，左侧导航栏选择 **实例**，单击 **创建实例**。
2. 在实例购买页面，选择地域、可用区、实例规格以及类型等。
   - **地域**：选择华东2（上海）。
   - **可用区**：选择上海非金可用区-A。
   - **实例规格**：选择 8 核 16 GB。
   - **类型**：选择 OceanBase。
   - **专有网络**：选择之前创建好的 VPC。
   - **虚拟交换机**：选择位于上海可用区 A 的虚拟交换机实例 ID。

> **说明**
>
> 请尽量选择 ODP 实例所在可用区的虚拟交换机。如果 ODP与应用不在同一可用区，或者 ODP 没有映射到应用所在可用区的虚拟交换机，应用访问 ODP 时则需要进行跨可用区访问，将会导致较高的延时问题。

3. 单击 **立即购买** 进入购买预览页面。
4. 确认订单信息无误并阅读产品服务条款后，勾选 **数据访问代理服务协议**，单击 **去开通** 完成购买。
5. 在购买成功后的提示页面，单击 **管理控制台** 返回 ODP 实例列表，可以看到刚刚创建的 ODP 实例。

6. 单击该实例 ID 进入其详情页，您可以看到已有的一个机房，单击详情页右上角的 **添加机房**，为该逻辑实例添加第二个机房。



7. 在新页面中，选择当前地域的另外一个可用区，如 **上海非金可用区-C**，且虚拟交换机也需选择位于可用区 C 的虚拟交换机实例 ID。

8. 购买后，返回该实例详情页，即可看到已存在两个机房。此时，该逻辑实例便对应了两个物理实例（即逻辑实例在某一个机房中的真实实例），分别部署在上海非金可用区 A 和上海非金可用区 C，两个物理实例共享相同的元信息配置。

添加物理数据节点

1. 登录 OceanBase 管理控制台，创建一个集群与两个租户，并设置主可用区分别为 **华东 2（上海）可用区 E** 和 **F**。详见 创建集群 与 创建租户。

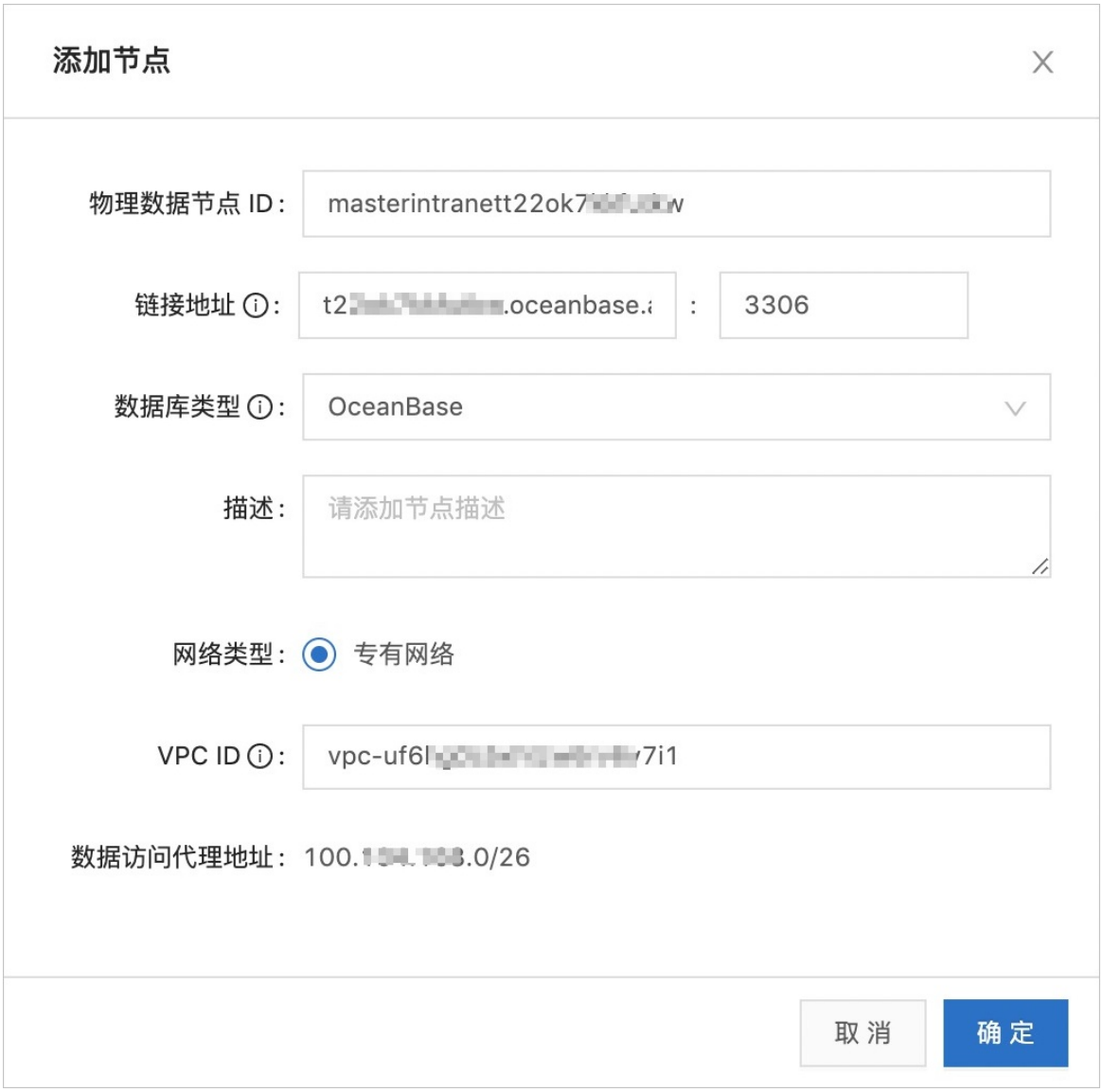
2. 将 ODP 地址加入到该 OceanBase 集群的白名单中：

i. 在 ODP 控制台 > **运维** > **物理数据节点** 页面，点击 **添加节点**，在弹出窗口中，获取 **数据访问代理地址**。

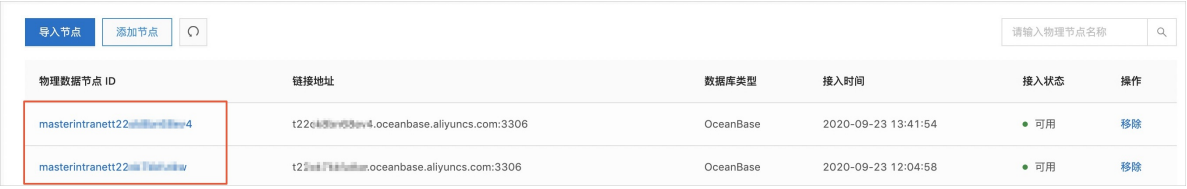
ii. 返回 OB 集群工作台，单击 **集群白名单** 栏旁边的编辑图标，添加该 ODP 地址。

3. 为该 OB 集群下的两个租户分别创建一个超级账号。详见 新建账号。
4. 返回 ODP 控制台 > **运维** > **物理数据节点** 页面，分别添加两个租户的数据节点。
   - **物理数据节点 ID**：格式为 `masterintranet + 租户 ID`，如 `masterintranett22okxxxxxxxx`。
   - **链接地址**：输入该物理数据节点的访问地址。
   - **数据库类型**：选择 OceanBase。
   - **网络类型**：选择 **专有网络**。

- **VPC ID**：输入物理节点所在专有网络 VPC 的 ID。



5. 添加完成后，在物理数据节点列表，即可看到这两个节点。



创建逻辑库与物理库

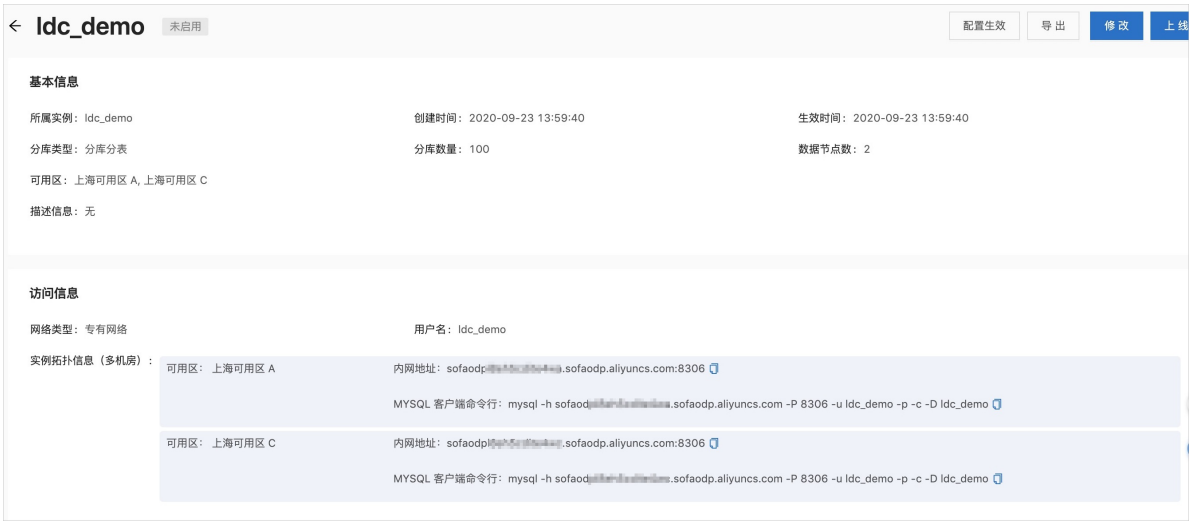
1. 进入 ODP 控制台 > **数据库** 页面，单击 **创建数据库**。
2. 在弹出的 **创建数据库** 窗口中，选择正确的 ODP 实例，单击 **创建**。

3. 在 **选择数据节点** 页面，选择刚刚添加的两个物理数据节点。

4. 单击 **下一步**，根据提示，填写或选择基本信息。根据本文的示例场景，需将 **物理分库数** 设置为 100，如下图所示。

5. 单击 **下一步**，进入建库预览，确认各项信息无误后，选择 **所有数据节点使用不同账号**，并为两个数据节点配置账号信息，如下图所示。

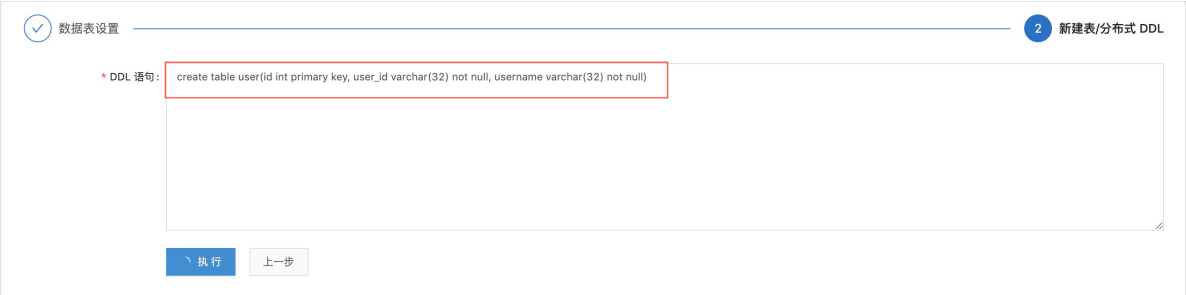
6. 信息确认完毕，单击 **创建** 开始创建数据库。
7. 进入该数据库详情页，您可以查看到对应可用区-A（实际为 E）的租户承载前 50 分片，对应可用区-C（实际为 F）的租户承载后 50 分片。



创建逻辑表与物理表

1. 在刚刚创建的数据库详情页，单击右侧页面下方的 **新增数据表**。
2. 进入创建数据表向导，填写或选择各项数据表信息，勾选 **现在创建物理表**，并点击 **下一步**。**示例**：以 user 表为例，创建拆分表，分表数为 100，分表规则为 MySQL 风格的字符串截断（-2 ~ 2），即取 `user_id` 字符串的最后两位作为 `sharding_key`（分库分表位）。

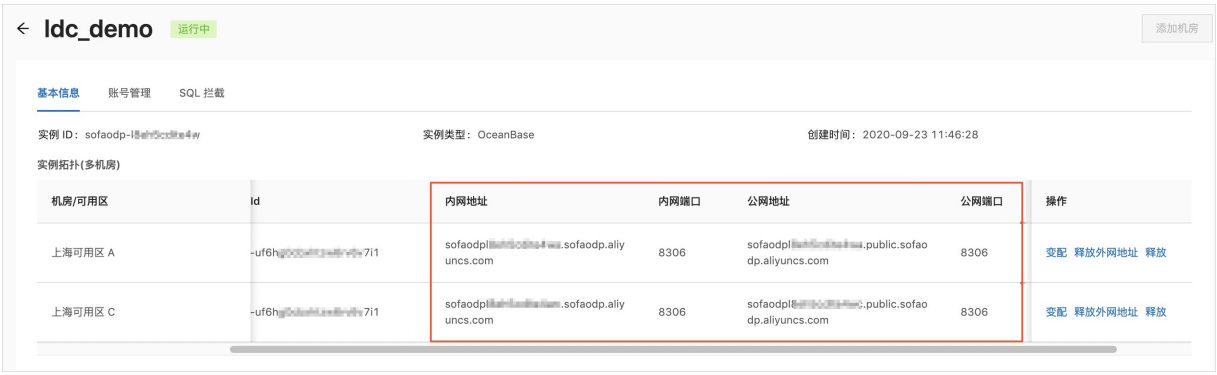
3. 输入 DDL 语句来进行创建数据表，单击 **执行**。



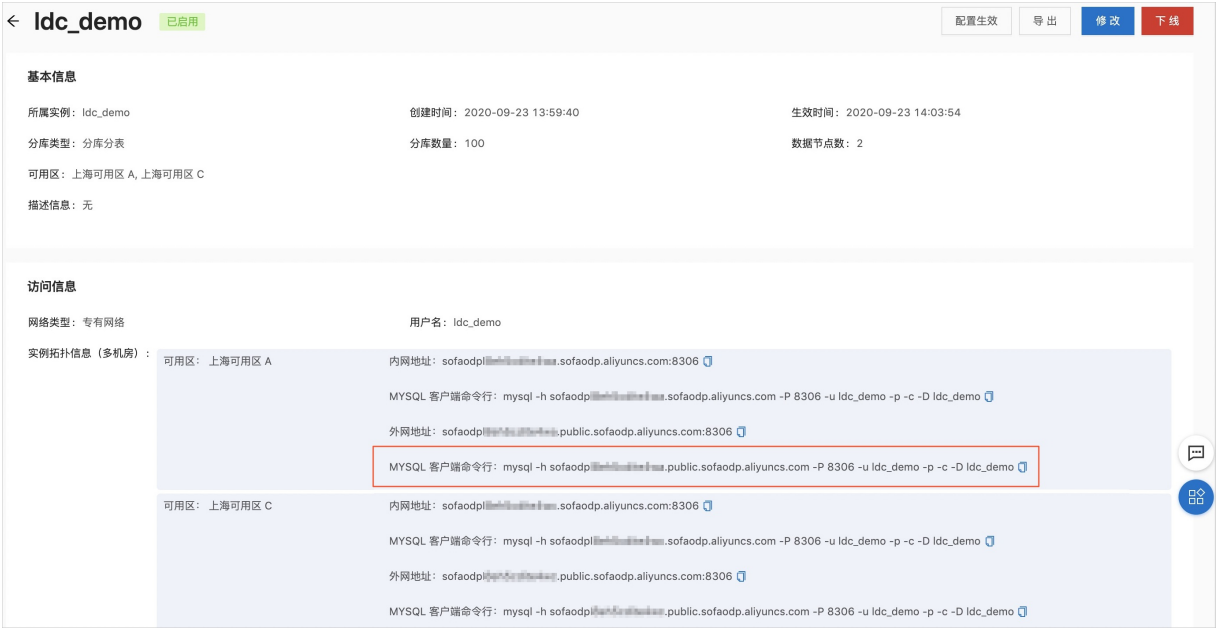
4. 返回数据库详情页，单击右上角的 **配置生效**，并 **确定**。

应用连接逻辑库

在业务 VPC 内，您可以直接使用内网地址访问 ODP。您也可以申请公网地址，然后在本地通过 MySQL CLI 来访问 ODP。



使用逻辑库中的 MYSQL 客户端命令行命令，即可访问某个可用区的 ODP 实例中的逻辑库。



执行 `show topology from $tableName`，即可查看某张逻辑表的拓扑结构，如下图所示。

```
~/Dev/jar » mysql -h sofaodpl[redacted].public.sofaodp.aliyuncs.com -P 8306 -u ldc_demo -p -c -D ldc_demo
Enter password:
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8343
Server version: 5.6.29-dbp-3.0.3 ZdalProxy Server (AntFinancial)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show tables;
+--------------------+
| Tables_in_ldc_demo |
+--------------------+
| user               |
+--------------------+
1 row in set (0.01 sec)

mysql> show topology from user;
+------+------------+------------+-------------+
| id   | group_name | table_name | schema_type |
+------+------------+------------+-------------+
| 0    | group_00   | user_00    | SHARD       |
| 1    | group_01   | user_01    | SHARD       |
| 2    | group_02   | user_02    | SHARD       |
| 3    | group_03   | user_03    | SHARD       |
```

应用可以通过 dbp-connector 较为便捷地访问双机房 ODP。详细的配置方法，可参见 配置同城双活模式。

配置时，需要特别注意以下几点：

- 在引入 dbp-connector 的依赖时，版本号必须是 1.1.1 及以上。
- 在配置连接池时，`url` 配置的格式为 `jdbc:mysql://${阿里云 ODP 实例 ID}@zone@.sofaodp.aliyuncs.com:8306/${数据库名称}` 。

```
<bean id="simpleDataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="i
nit">
     <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
     <property name="url" value="jdbc:mysql://${yourDbpInstanceId}@zone@.sofaodp.aliyuncs
.com:8306/${yourDatabase}"/>
     <property name="username" value="${username}"/>
     <property name="password" value="${password}"/>
</bean>
```

配置与访问示例

1. 修改 Spring 文件。本文示例如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans  xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
default-autowire="byName">

    <bean id="dbpDiscovery"class="com.alipay.sofa.dbp.discovery.DbpDiscovery"/>

    <bean id="simpleDataSourceConnector"class="com.alipay.sofa.dbp.DbpDataSource" init-me
thod="init">
        <property name="delegate" ref="simpleDataSource"/>
        <property name="appName" value="ldc_demo_app"/>
        <property name="database" value="ldc_demo"/>
        <property name="dbpInstanceId" value="sofaodp-l8eh5cdite4w"/>
        <property name="clientTracer" ref="clientTracer"/>
    </bean>

    <!-- dataSource pool -->
    <bean id="simpleDataSource" class="com.alibaba.druid.pool.DruidDataSource" init-metho
d="init">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <!--    这里为了本地测试使用了公网域名，实际应用部署时应该使用内网域名，即不带public      --
>
        <property name="url" value="jdbc:mysql://sofaodp-lxxxxxxxtest@zone@.public.sofaod
p.aliyuncs.com:8306/ldc_demo"/>
        <property name="username" value="${username}"/>
        <property name="password" value="${password}"/>
    </bean>

    <bean id="clientTracer" class="com.alipay.sofa.dbp.DbpClientTracer">
        <!-- 设置客户端日志才样的阈值，默认3ms（3ms内成功的sql按10%采样），设置为0 全部打印 -->
        <property name="sampleThreshold" value="3"/>
    </bean>
</beans>
```

2. 在应用代码（以 Java 为例）中，测试插入一条位于 23 分片的记录，然后查询该条记录。示例如下：

```
public class DbpConnectorAliyunTest extends AbstractTestBase{
    @Autowired
    @Qualifier("simpleDataSourceConnector")
    private DataSource dataSource;

    static{
// 应用部署时，发布部署系统会自动为您注入此系统变量为当前可用区，无需手动设置，此处仅为演示
        System.setProperty("com.alipay.ldc.datacenter","cn-shanghai-a");
    }

    @Test
    public void test() throws SQLException{
       // user_id的最后两位为分片值，即当前这条数据要落到23分片
        String userId ="随机生成的数字串23";
        String username ="user23";
        String insertSql ="insert into user (id, user_id, username) values (?, ?, ?)";
        Connection connection = dataSource.getConnection();
        PreparedStatement insertStmt = connection.prepareStatement(insertSql);
        insertStmt.setInt(1,0);
        insertStmt.setString(2, userId);
        insertStmt.setString(3, username);
        insertStmt.executeUpdate();
        insertStmt.close();

        String querySql ="select username from user where user_id = ?";
        PreparedStatement queryStmt = connection.prepareStatement(querySql);
        queryStmt.setString(1, userId);
        ResultSet resultSet = queryStmt.executeQuery();
        resultSet.next();
        Assert.assertEquals(username, resultSet.getString(1));
    }
}
```

3. 连接至物理库，验证数据路由的正确性。

```
~/Dev/jar » mysql -ht[redacted]o.oceanbase.aliyuncs.com -P3306 -uroot123 -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1049990
Server version: 5.6.25 OceanBase 2.2.30 (r20200701180433-ff6ee627c2a9fe1427d25ae6ea97a162ea70b6ea) (Built Jul  1 2020 19:21:28)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use ldc_demo_23;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+----------------------+
| Tables_in_ldc_demo_23 |
+----------------------+
| user_23              |
+----------------------+
1 row in set (0.01 sec)

mysql> select * from user_23;
+----+--------------------------+----------+
| id | user_id                  | username |
+----+--------------------------+----------+
|  0 | 随机生成的数字串23         | user23   |
+----+--------------------------+----------+
1 row in set (0.01 sec)
```

16.常见问题

16.1. 错误代码

常见错误代码列表如下：

- ERROR 7001: Table rule execute error
- ERROR 7002: Unable to find table rule
- ERROR 7022: Physical database connection pool is full, database name: x
- ERROR 7022: Get connection from physical database timeout, database name: x
- ERROR 7031: Cannot execute update SQLs cross databases in transaction
- ERROR 7100: Getting next sequence error, xx
- ERROR 1003: Transaction error, need to rollback
- ERROR: Server connection execute error: x
- ERROR: Access denied for user 'xx'

ERROR 7001: Table rule execute error

问题描述：出现该错误是表示规则计算出错，请检查数据访问代理管控台上的分库分表规则是否配置正确。

排查方法：

1. 检查分库分表规则，如果使用自定义规则的话，需要返回 int 值。
2. 如果检查分库分表没有问题，请确认是否已生效了数据库配置。

ERROR 7002: Unable to find table rule

问题描述：无法找到表的规则，查看用户 SQL 中的表有没有在数据访问代理控制台上配置分表规则。

排查方法：

1. 检查分库分表规则，是否有配置当前表的分库分表规则。
2. 如果检查分库分表没有问题，请确认一下是否有生效数据库配置。

ERROR 7022: Physical database connection pool is full, database name: x

问题描述：数据访问代理连接后端数据库的连接池已满。

排查方法：

1. 查看是否有耗时 SQL 影响连接池。
2. 如果耗时 SQL 是合理的，则应该是连接池本身设置不合理导致，需在连接参数界面调整连接数。

ERROR 7022: Get connection from physical database timeout, database name: x

排查方法：和物理数据库连接池已满类似，这个错误是由于连接池比较繁忙而导致超时，应该是建立连接耗时较长导致。

ERROR 7031: Cannot execute update SQLs cross databases in transaction

问题描述：事务内，多条更新 SQL 不允许跨物理数据库执行。

排查方法：检查事务内的 SQL 是否跨物理库。

ERROR 7100: Getting next sequence error, xx

问题描述：由于 sequence 的步长用光后，会向数据库取下一个步长的 sequence，如果一瞬间取 sequence 的请求太多的话，会导致数据库的锁冲突严重从而抛出该异常。

排查方法：

建议获取 sequence 失败通过业务层重试解决。

ERROR 1003: Transaction error, need to rollback

问题描述：在事务中执行时如果出现了 SQL 执行异常，数据访问代理为了数据安全，会强制将事务状态设置为中断，需要显式调用 rollback 后才能继续执行其他 SQL。

排查方法：确认是否有数据库异常没有处理，或者捕获了数据库异常但没有调用 rollback 回滚事务。如确认异常后，则客户端需要强制调用 rollback。

ERROR: Server connection execute error: x

问题描述：数据访问代理将 SQL 发往后端物理数据库执行失败，不同的错误码对应不同的错误。

排查方法：此错误一般是由于业务自身的 SQL 导致，请检查 SQL 是否可以优化。

ERROR: Access denied for user 'xx'

问题描述：用户访问被拒绝。

排查方法：

1. 确认是否是连接到的正确的数据访问代理实例，且帐号密码正确。帐号需要和数据库名保持一致，密码是在数据访问代理管控台界面上由用户输入。
2. 确认数据库是否已经上线。

16.2. VPC 问题

常见 VPC 问题列表：

- VPC 网络中如何访问用户数据节点
- 经典网络与 VPC 的区别是什么
- 如何查看当前的 VPC 信息

VPC 网络中如何访问用户数据节点

进入数据访问代理管控台，选择 **运维** > **物理数据节点** > **添加节点**， 输入数据节点的 ID、链接地址、高权限账户的用户名和密码以及节点所在的 VPC ID，即可访问用户数据节点。

经典网络与 VPC 网络的区别什么

- 经典网络：在经典网络中模式下，同一个网路中的机器是能够相互访问，存在一定的安全隐患。

- VPC 网络：在 VPC 网络模式下，整个网络被划分成一个个彼此隔离的专有网络（VPC 网络），各个网络中机器可以相互访问，VPC 之间相互隔离，更加安全，要想打通 VPC 之间的网络，需要使用阿里云提供的反向 VPC 服务。

如何查看当前的 VPC 信息

在控制台，选择 **产品与服务 > 专有网络 VPC > 专有网络** 可以查看和创建专有网络，单击 VPC ID 可以进一步查看 VPC 下的交换机和路由信息。

16.3. 分库分表问题

常见分库分表问题列表：

- 分库分表是否支持多个拆分字段
- 分库分表是否支持跨库 JOIN
- 分库分表对拆分库表的数量是否有限制

分库分表是否支持多个拆分字段

数据访问代理分库分表功能支持多个拆分字段，但是如果配置了拆分字段，后续执行 SQL 均需要加上拆分字段的查询条件。所以建议尽量使用单个拆分字段，减少 SQL 使用场景限制。

例如，如果根据 ID 字段进行拆分，后续执行 SQL 都需要带上 `WHERE ID=?` 过滤条件，否则会出现 `Table rule execute error` 错误。

分库分表是否支持跨库 JOIN

数据访问代理暂不支持分库分表场景下的跨库 JOIN 功能，仅支持同一个分片下的 JOIN SQL。假设有一条 JOIN SQL `SELECT * FROM user as u INNER JOIN join_table as j ON u.id = j.user_id`，那么针对以下两种情况，处理方式有所不同。

- 如果 JOIN 表存在分库分表规则，JOIN 时 JOIN 表的后缀和 FROM 表保持一致：`SELECT * FROM user_00 as u INNER JOIN join_table_00 as j ON u.id = j.user_id`。
- 如果 JOIN 表无分库分表规则，JOIN 时 JOIN 表保持原样，不做表名替换：`SELECT * FROM user_00 as u INNER JOIN join_table as j ON u.id = j.user_id`。

分库分表对拆分库表的数量是否有限制

数据访问代理不限制拆分库表的数量，但分库分表的整体数量受后端数据节点的规格限制，请根据业务场景合理设计分库分表的拆分数量。

16.4. 分布式序列问题

分布式序列是否全局唯一

- 单库单表的分布式序列可以保证生成的序列 ID 全局唯一。
- 分库分表的分布式序列仅保证单个分片的序列 ID 唯一，如果需要确保分库分表的分布式序列全局唯一，可以在序列 ID 中拼接分库分表位，以此保证全局唯一。

分布式序列使用注意事项

- 在运行过程中，不能调整 `dbp_sequence` 表的数据，如果进行了调整，可能会导致序列数据冲突。

- 在运行过程中，不能删除 `dbp_sequence` 表的数据，如果删除，可能会出现不一致风险，导致无法获取错误。如果误删了数据，请点击 **配置生效** 按钮，数据访问代理会重新初始化序列。

16.5. 分布式事务问题

数据访问代理是否支持分布式事务

数据访问代理支持与分布式事务的集成使用。版本要求：

- 数据访问代理 V2.9.5 或更高版本
- 分布式事务 V2.5.2 或更高版本

具体配置信息请参考 依赖与配置项。

在数据访问代理 V2.9.5 之前的版本中，如果事务中存在跨数据库的 SQL，数据访问代理将会返回错误 `ERROR 7031: Cannot execute update SQLs cross databases in transaction` 。

16.6. DDL 问题

DDL 任务创建常见问题

- SQL 格式错误
- SQL 语法错误

DDL 任务执行常见问题

- 无法获取分库分表拓扑
- 找不到物理数据节点
- SQL 执行失败

DDL 任务创建常见问题

SQL 格式错误

错误信息：`sqlContent Format error. must end with ;\n`

解决方法：每条 SQL 语句必须以分号（;）加回车结尾。

SQL 语法错误

- **错误信息**：`Only support DDL statement.`

  解决方法：不要传非 DDL 语句。

- **错误信息**：`Sql parse error.`

  解决方法：SQL 语句有语法错误。修正语法错误或不支持的语法，已经支持的语法参考 数据访问代理中的 DDL 语法。

DDL 任务执行常见问题

无法获取分库分表拓扑

错误信息：根据分库分表拓扑创建 SQL 执行任务失败，原因：xxx。请解决问题后‘重试’，且无法进行‘跳过’操作。

解决方法：根据原因中的信息去修复问题，通常的原因有：无法解析出逻辑表名、没有创建数据表。

找不到物理数据节点

找不到物理数据节点

错误信息： `Skip current sql batch job execution. reason: Cannot find Dbnode by dbnodeId: {dbnodeId}`

解决方法： 数据表的物理数据源配置不正确。请到 **数据库管理 > 选择数据库名 > 物理数据源** 标签页检查配置是否正确。

SQL 执行失败

错误信息： `TableName[{tableName}]\n {SQL 错误信息}`

解决方法： 根据 `{SQL 错误信息}` 来解决问题，可能的情况有，表已经存在、索引名重复、表不存在、列已存在、列不存在、主键重复等，问题解决后点击 **重试** 继续执行。

> **说明**
>
> - 对于“索引名重复”、“表已经存在”的情况，可以连接到 `{tableName}` 对应的数据库核对，如果是正确的状态，可以跳过这条继续往下执行。
> - 如果 `{SQL 错误信息}` 是“超时”（有 `read timed out error` 错误信息），例如：数据量比较大时创建或修改索引、修改表结构，则运行时间较长，可能会超时；这时也可连接到 `{tableName}` 对应的数据库核对，如果是正确的状态，可以跳过这条继续往下执行。

16.7. 数据节点问题

常见数据节点问题列表：

- 在数据访问代理中使用 RDS 或 OceanBase 的节点 ID
- 如何查找 RDS 或 OceanBase 的节点 ID

在数据访问代理中使用 RDS 或 OceanBase 的节点 ID

在添加物理数据节点时，需要通过数据节点 ID 来定位指定的数据节点，如下图所示：

如何查找 RDS 或 OceanBase 的节点 ID

RDS 的节点 ID 即 RDS 的实例 ID，用来唯一标识一个 RDS 实例，可以在 RDS 的实例详情页面找到实例 ID。

OceanBase 的节点 ID 也可以唯一标识一个 OceanBase 实例。目前，OceanBase 的管控台并不展示节点 ID，但您可以从 OceanBase 提供的内网地址中提取出节点 ID。

如内网地址：`obxxxxxxx-741-intranet.cn-hangzhou.oceanbase.aliyuncs.com`，其中第二段的 `741` 就表示节点 ID。

16.8. ODP 容器与实例关联错误修复方案

修复方案

如果您出现 ODP 容器与实例关联错误，您可以通过以下方案进行修复：

1. 登录 DBP 数据库 `dbpconsole_db`，将所有 ODP 实例逻辑库下线，并备份表 `zdalproxy_instance` 和 `zdalproxy_machine`。
2. 分别执行如下 SQL 脚本：

   ```
   select * from zdalproxy_instance where instance_id = '$dbp-实例 ID' and env_tenant = '$租户' and env_mode='$环境'
   ```

   确认实例信息正确，并分别记录实例 ID 列的值，例如实例：dbp-1234，ID：118。
3. 按实例分别查看 ODP 实例 SLB 关联的容器 IP，执行如下脚本：

   ```
   select * from zdalproxy_machine where ip = '$容器 IP'
   ```
4. 确认除 instance_id 外其他信息都正确后，执行如下脚本：

   ```
   update zdalproxy_machine set instance_id = $步骤 2 获得的实例的 ID where env_tenant='$租户' and env_mode = '$环境' and ip = '$容器 IP'
   ```
5. 单实例所有容器均执行完毕后，等待 ODP 自动重新拉取关联信息（约 2 分钟）。
6. 登录所有有修改的容器，执行命令确认 instanceId 是否为相应的 ODP 实例 ID。

   ```
   cat /home/admin/conf/zdalproxy-meta-config.json | python -m json.tool
   ```

   ```
   sh-4.1# cat  zdalproxy-meta-config.json  | python -m json.tool
   {
       "data": {
           "instanceId": "dbp-90Pdrvf1vWGI",
           "instanceName": "dbp-90Pdrvf1vWGI",
           "ipWhiteList": "",
           "schemas": []
       },
       "success": true
   }
   ```
7. 确认无误后上线 ODP 逻辑库。
8. 连接所有操作的 ODP 实例，确认可以正常登录。

回滚方案

还原步骤 1 中备份的表 `zdalproxy_instance` 和 `zdalproxy_machine`。

16.9. 其他问题

Java 应用创建数据访问代理连接失败

检查以下几个要点：

- 白名单：是否在 ODP 上将应用服务器 IP 加入了白名单，以及在物理数据库上将 ODP 实例 IP 加入了白名单。
- MySQL 客户端：如果直接使用命令行方式可以连接 ODP，说明网络和配置都是正确的。
- JDBC Driver for MySQL（mysql-connector-java）版本是否为 5.1.x，目前经过 ODP 兼容性完整测试的版本为 5.1.30 和 5.1.40。

如何修改当前 Session 的 OB 执行超时时间

可通过以下方式进行修改：

- 在执行的 SQL 语句中带上 `select /*+query_timeout(100000000000),trx_timeout(100000000000)*/` 的 Hint。ODP 会将该 Hint 透传到 OB 去。该 SQL 语句所运行的 Session 的超时时间就会改变。
- 在 ODP 控制台上配置如下参数，更改每条新连接的超时时间。这种修改方法在 OB 侧用 `show parameters like "timeout"` 是看不到的。但是 ODP 新建的 session 都会带上这些参数。

SQL 执行超时：SocketTimeoutException 错误

原因：该问题一般是由于慢 SQL 导致。

解决方案：

- 如果 SQL 执行确实需要较长时间，则需要调整应用端和 ODP 的 socketTimeout 参数。
  - 修改应用端 socketTimeout 参数：

```
<!--数据源-->
<bean id="simpleDataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql://                    6/poctest?socketTimeout=5000"/>
    <property name="username" value="          "/>
    <property name="password" value="          "/>
</bean>
```

- 修改数据访问代理 socketTimeout 参数：

- 如果 SQL 执行耗时非预期时间，则需要分析 ODP 的执行耗时日志 `logs/zdalproxy/sql-digest.log`，且需要前往物理库进一步分析慢 SQL 原因。

批量更新报错 bad SQL grammar [], Syntax error

现象：使用 MyBatis 对 ODP 进行批量更新，出现 bad SQL grammar [], Syntax error 错误。当前批量更新写法如下：

```
<mapper namespace="com.forms.lego.il.das.batch.mapper.BatchMapper" >
    <update id="updateRepaymentBill" parameterType="java.util.List">
        <foreach collection="list" item="tl" separator=";">
            update t_loan_repayment_bill
            <set>
                <trim suffixOverrides=",">
                    keprcd_stus_cd = #{tl.keprcdStusCd, jdbcType=VARCHAR},
                </trim>
            </set>
            <where>
                loan_dubil_no  = #{tl.loanDubilNo, jdbcType = VARCHAR}
            </where>
        </foreach>
    </update>
</mapper>
```

原因：根据上述的写法，实际上产生的 SQL 语句是 multisql 类型，也就是用分割符 `;` 分割开的多条 update 语句。ODP 不支持这种multisql类型的语句，因此会出现语法错误。

解决方案：在实际业务中，批量更新使用多条 update 语句一条一条去更新，不仅性能差而且容易造成阻塞。因此建议通过 case when 编写成一条 SQL 语句来执行，语法示例如下：

```
UPDATE mytable
   SET myfield = CASE id
         WHEN 1 THEN 'value'
         WHEN 2 THEN 'value'
         WHEN 3 THEN 'value'
END
WHERE id IN (1,2,3)
```

MyBatis 也是完全支持这种写法的，示例如下：

```
<update id="updateByBatch" parameterType="java.util.List">
     UPDATE t_goods
     SET NODE_ID = CASE
          <foreach collection="list" item="item" index="index">
               WHEN GOODS_ID = #{item.goodsId} THEN #{item.nodeId}
          </foreach>
     END
     WHERE GOODS_ID IN
          <foreach collection="list" index="index" item="item" open="("separator=","close="
)">
                #{item.goodsId}
          </foreach>
</update>
```

ODP 是否支持影子表？例如 zdal 的影子表是在表名后加“_t”

ODP 目前不支持自定义后缀。

16.10. 问题排查案例

16.10.1. RDS 扩容导致 ODP 出现连接不可用的报错

问题现象

RDS 扩容后，访问 ODP 执行 SQL 时出现连接不可用的报错。

问题原因

RDS 扩容操作会造成 ODP 连接池中已有的连接失效，但 ODP 无主动探活连接的能力（ODP 用的是连接池，一旦后端的 RDS 关闭连接，ODP 无法感知到）。

ODP 清除脏连接有两种途径：

- 在连接空闲时间超时后主动清理。

  可配置，当前的 ODP 实例配置为 118 分钟，默认 12 分钟。
- 在有新的连接请求时，ODP 从连接池获取可用连接时发现连接不可用，抛出异常。

  ODP 的逻辑库可通过配置生效功能清空连接池，达到快速清除脏连接的目的。

解决方案

- 建议业务增加重试机制，在获取连接异常时重新连接。
- 若业务无重试机制，需在 RDS 扩容完成后登录 ODP 控制台，然后进入相应逻辑库管理页面单击 **配置生效**。3 分钟后，连接池会重置，并清空连接池中所有连接。

16.10.2. 执行 select 语句出现 Error 2013 (HY000)报错

问题现象

某客户在运行 `select * from xxxx` 的时候出现 `Error 2013 (HY000): Lost connection to MySQL server during query` 报错。

排查步骤

1. 查询 ODP 的 `/home/admin/logs/zdalproxy/sql-digest.log.xxxx` 日志。

   发现如下异常：

```
nnectionFactory.createManagedConnection(LocalManagedConnectionFactory.java:235)
        ... 25 more
2020-04-16 16:52:33,852 ERROR Processor1-E32 session.ErrorHandler - schema: svca
is, msg: Server write back error - Doesn't support zero DATE, DATETIME or TIMEST
AMP, please check mysql official document.
java.sql.SQLException: Doesn't support zero DATE, DATETIME or TIMESTAMP, please
check mysql official document.
        at com.alipay.zdal.proxy.jdbc.response.ComQueryResponse.writeAndSaveCach
e(ComQueryResponse.java:196)
        at com.alipay.zdal.proxy.jdbc.response.ComQueryResponse.write(ComQueryRe
sponse.java:69)
        at com.alipay.zdal.proxy.server.session.SqlTemplate$Default.doExecute(Sq
lTemplate.java:128)
        at com.alipay.zdal.proxy.server.session.ZdalBlockingSession.executeInter
nal(ZdalBlockingSession.java:350)
        at com.alipay.zdal.proxy.server.session.ZdalBlockingSession$2.run(ZdalBl
ockingSession.java:249)
        at com.alipay.zdal.proxy.route.RouteTemplate.execWithoutRouteHint(RouteT
emplate.java:298)
```

2. 根据报错时间查询 `/home/admin/logs/zdalproxy/common-error.log` 日志。

发现如下报错信息：

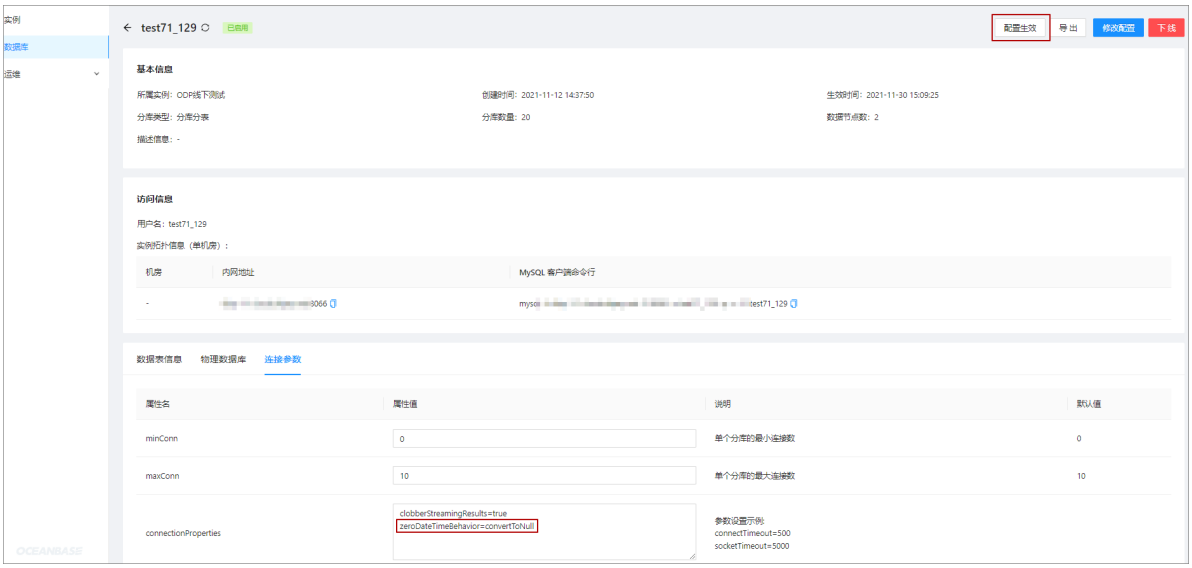
```
        at java.lang.Thread.run(Thread.java:766)
Caused by: java.sql.SQLException: Value '0000-00-00 00:00:00' can not be represe
nted as java.sql.Timestamp
        at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:1084)
        at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:987)
        at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:973)
        at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:918)
        at com.mysql.jdbc.ResultSetRow.getTimestampFast(ResultSetRow.java:1102)
        at com.mysql.jdbc.ByteArrayRow.getTimestampFast(ByteArrayRow.java:127)
        at com.mysql.jdbc.ResultSetImpl.getTimestampInternal(ResultSetImpl.java:
6587)
        at com.mysql.jdbc.ResultSetImpl.getTimestamp(ResultSetImpl.java:6187)
        at com.mysql.jdbc.ResultSetImpl.getObject(ResultSetImpl.java:5053)
        at com.alipay.zdal.atom.jboss.resource.adapter.jdbc.WrappedResultSet.get
Object(WrappedResultSet.java:576)
```

3. 查看目标数据库。

   发现有一个日期字段的值为 `0000-00-00 00:00:00`，当 DB 的记录中某日期字段的值为 `0000-00-00 00:00:00` 时就会出现 Error 2013 (HY000) 报错。

解决方案

1. 登录 ODP 控制台。
2. 在左侧导航栏单击 **数据库**。
3. 在数据库列表单击目标数据库名称。
4. 单击 连接参数 页签，然后在 connectionProperties 参数中添加 `zeroDateTimeBehavior=convertToNull` 参数。
5. 单击右上角的 **配置生效**。



16.10.3. ODP 出现 Communications link failure 报错

问题现象

某用户 ODP 的 `/home/admin/logs/zdalproxy/common-error.log` 中发现以下错误，导致应用的 SQL 执行失败。

```
2020-03-27 00:00:11,526-[ac11000115852384115263001585,0]Connection error occured: com.alipa
y.zdal.atom.jboss.resource.connectionmanager.TxConnectionManager$TxConnectionEventListener@
4fa5cdd[state=NORMAL 。。。。。。 com.mysql.jdbc.exceptions.jdbc4.CommunicationsException: Co
mmunications link failure The last packet successfully received from the server was 7,207,1
73 milliseconds ago. The last packet sent successfully to the server was 0 milliseconds ago
.
```

排查步骤

1. 查询 ODP 报错日志 `/home/admin/logs/zdalproxy/common-error.log` 。

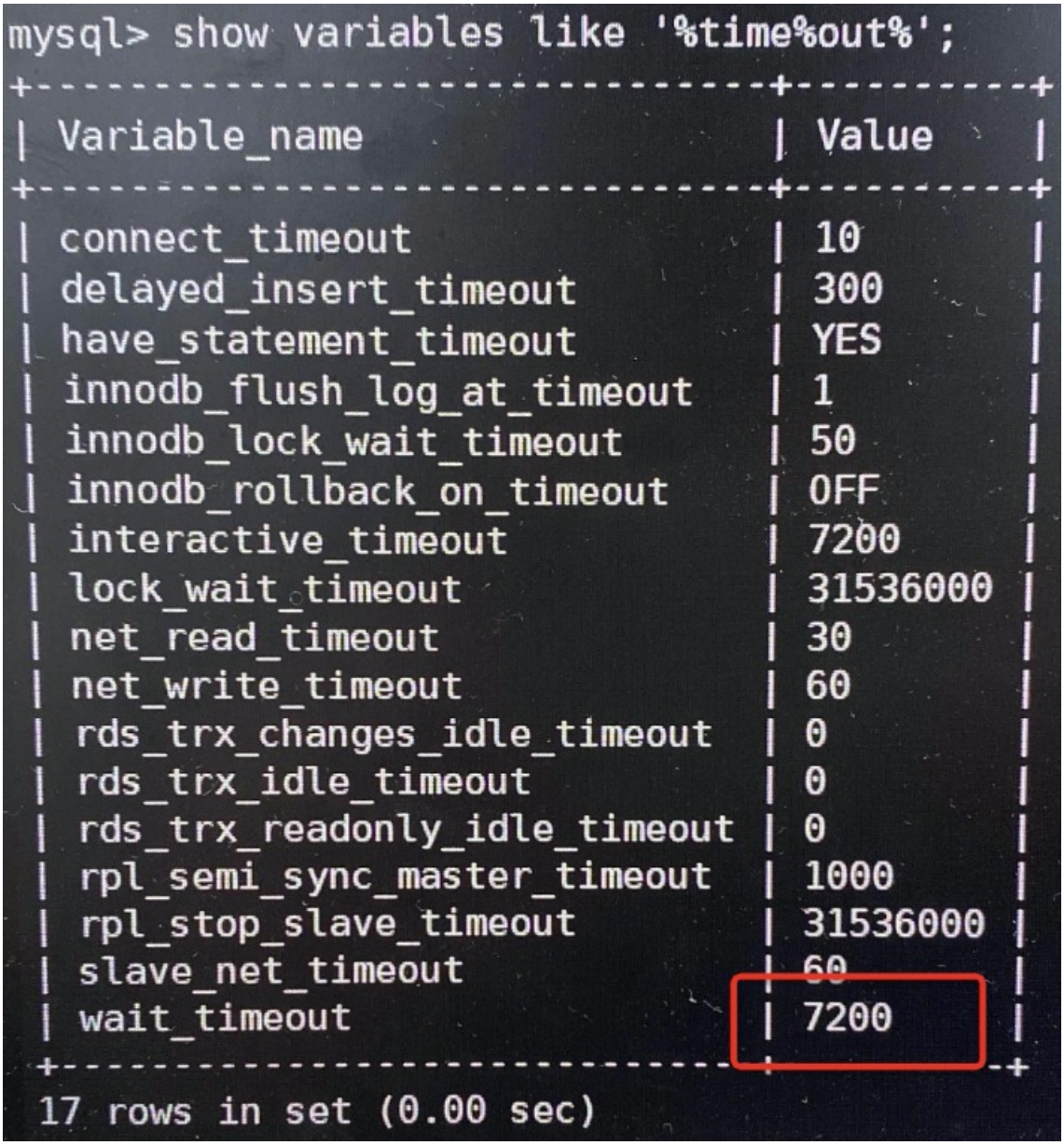
```
2020-03-27 00:00:11,526-[ac11000115852384115263001585,0]Connection error occured: com.alipay.zdal.atom.jboss.resource.connectionmanager.
TxConnectionManager$TxConnectionEventListener@4fa5cdd[state=NORMAL mc=com.alipay.zdal.atom.jboss.resource.adapter.jdbc.local.LocalManagedConnection@52e58ba1
handles=1 lastUse=1585231204353 permit=true trackByTx=false mcp=com.alipay.zdal.atom.jboss.resource.connectionmanager.
DefaultManagedConnectionPool$OnePool@34df302c context=com.alipay.zdal.atom.jboss.resource.connectionmanager.InternalManagedConnectionPool@314d89e2 xaResource=com.
alipay.zdal.atom.jboss.resource.connectionmanager.TxConnectionManager$LocalXAResource@633ec4ba txSync=null]
com.mysql.jdbc.exceptions.jdbc4.CommunicationsException: Communications link failure

The last packet successfully received from the server was 7,207,173 milliseconds ago.  The last packet sent successfully to the server was 0 milliseconds ago.
    at sun.reflect.GeneratedConstructorAccessor31.newInstance(Unknown Source)
    at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
    at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
    at com.mysql.jdbc.Util.handleNewInstance(Util.java:409)
    at com.mysql.jdbc.SQLError.createCommunicationsException(SQLError.java:1127)
    at com.mysql.jdbc.MysqlIO.reuseAndReadPacket(MysqlIO.java:3715)
    at com.mysql.jdbc.MysqlIO.reuseAndReadPacket(MysqlIO.java:3604)
    at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:4155)
    at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:2615)
    at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:2776)
    at com.mysql.jdbc.ConnectionImpl.execSQL(ConnectionImpl.java:2832)
    at com.mysql.jdbc.ConnectionImpl.setAutoCommit(ConnectionImpl.java:5357)
    at com.alipay.zdal.atom.jboss.resource.adapter.jdbc.BaseWrapperManagedConnection.setJdbcAutoCommit(BaseWrapperManagedConnection.java:282)
    at com.alipay.zdal.atom.jboss.resource.adapter.jdbc.local.LocalManagedConnection.setAutoCommit(LocalManagedConnection.java:98)
    at com.alipay.zdal.atom.jboss.pool.PoolingConnection.setAutoCommit(PoolingConnection.java:79)
    at com.alipay.zdal.atom.jboss.resource.adapter.jdbc.WrappedConnection.setAutoCommit(WrappedConnection.java:382)
    at com.alipay.zdal.config.context.GenericExecutionContext.add(GenericExecutionContext.java:183)
    at com.alipay.zdal.config.context.GenericExecutionContext.getConnection(GenericExecutionContext.java:202)
    at com.alipay.zdal.group.jdbc.GroupStatement.createStatement(GroupStatement.java:96)
    at com.alipay.zdal.group.jdbc.GroupStatement.executeQuery(GroupStatement.java:153)
    at com.alipay.zdal.client.jdbc.statement.ZdalStatement.executeQueryWithShardingRule(ZdalStatement.java:430)
    at com.alipay.zdal.client.jdbc.statement.ZdalStatement.executeQuery(ZdalStatement.java:169)
    at com.alipay.zdal.proxy.server.session.SelectScanAll.executeOnce(SelectScanAll.java:148)
    at com.alipay.zdal.proxy.server.session.SelectScanAll.access$000(SelectScanAll.java:40)
    at com.alipay.zdal.proxy.server.session.SelectScanAll$1.doExecute(SelectScanAll.java:75)
    at com.alipay.zdal.proxy.server.session.SelectScanAll$1.doExecute(SelectScanAll.java:72)
    at com.alipay.zdal.proxy.server.session.SqlExecuteCallbackTemplate.execute(SqlExecuteCallbackTemplate.java:22)
    at com.alipay.zdal.proxy.engine.SqlExecuteEngine.lambda$asyncExecute$0(SqlExecuteEngine.java:43)
    at java.util.concurrent.FutureTask.run(FutureTask.java:266)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
    at java.lang.Thread.run(Thread.java:745)
Caused by: java.io.EOFException: Can not read response from server. Expected to read 4 bytes, read 0 bytes before connection was unexpectedly lost.
    at com.mysql.jdbc.MysqlIO.readFully(MysqlIO.java:3161)
    at com.mysql.jdbc.MysqlIO.reuseAndReadPacket(MysqlIO.java:3615)
    ... 26 more
```

   从堆栈信息中可以看出，出现问题时 jdbc 想要重用该连接，而该连接在 7,207,173 毫秒（2 小时零 7 秒）之前还是活跃的。因为 RDS 比 ODP 早一步检测到了 idle 连接，所以 RDS 关闭了此连接。

2. 在 RDS 服务器上执行如下命令查看 RDS 的超时配置。

```
show variables like '%time%out%'
```



```
mysql> show variables like '%time%out%';
+---------------------------------+----------+
| Variable_name                   | Value    |
+---------------------------------+----------+
| connect_timeout                 | 10       |
| delayed_insert_timeout          | 300      |
| have_statement_timeout          | YES      |
| innodb_flush_log_at_timeout     | 1        |
| innodb_lock_wait_timeout        | 50       |
| innodb_rollback_on_timeout      | OFF      |
| interactive_timeout             | 7200     |
| lock_wait_timeout               | 31536000 |
| net_read_timeout                | 30       |
| net_write_timeout               | 60       |
| rds_trx_changes_idle_timeout    | 0        |
| rds_trx_idle_timeout            | 0        |
| rds_trx_readonly_idle_timeout   | 0        |
| rpl_semi_sync_master_timeout    | 1000     |
| rpl_stop_slave_timeout          | 31536000 |
| slave_net_timeout               | 60       |
| wait_timeout                    | 7200     |
+---------------------------------+----------+
17 rows in set (0.00 sec)
```

3. 登录 ODP 控制台，查看 ODP 的超时设置。
   i. 登录 ODP 控制台。
   ii. 在左侧导航栏单击 **数据库**。
   iii. 在数据库列表单击目标数据库名称。

iv. 单击 连接参数 页签，然后查看 idleTimeoutMinutes 配置。

问题原因

ODP 每分钟检测一次 idle 的连接，然后和 `idleTimeoutMinutes` 进行对比。当此连接的 idle 时间大于 `idleTimeoutMinutes` 的时候，就会删除这个连接。

RDS也会检测 idle 的连接，当此连接的 idle 时间大于 `wait_timeout` 的时候，就会删除这个连接。但是当RDS删除这个连接的时候，ODP这边是不知道RDS已经关闭了这个连接的。所以当ODP尝试再次使用此连接执行sql时，就会发生刚才看到的报错。

若 `idleTimeoutMinutes` 大于 `wait_timeout` ，当此连接空闲的时间大于等于 `wait_timeout` ，RDS就会关闭此连接，就会发生出现报错。 若 `idleTimeoutMinutes` 等于 `wait_timeout` ，由于 ODP 是每分钟检查一次空闲连接，还是有一定几率发生报错。本案例中，ODP 的 `idleTimeoutMinutes` 被设置为了 RDS 的 `wait_timeout` 时间，两个值都被设置为了两小时，所以产生了报错。

解决方案

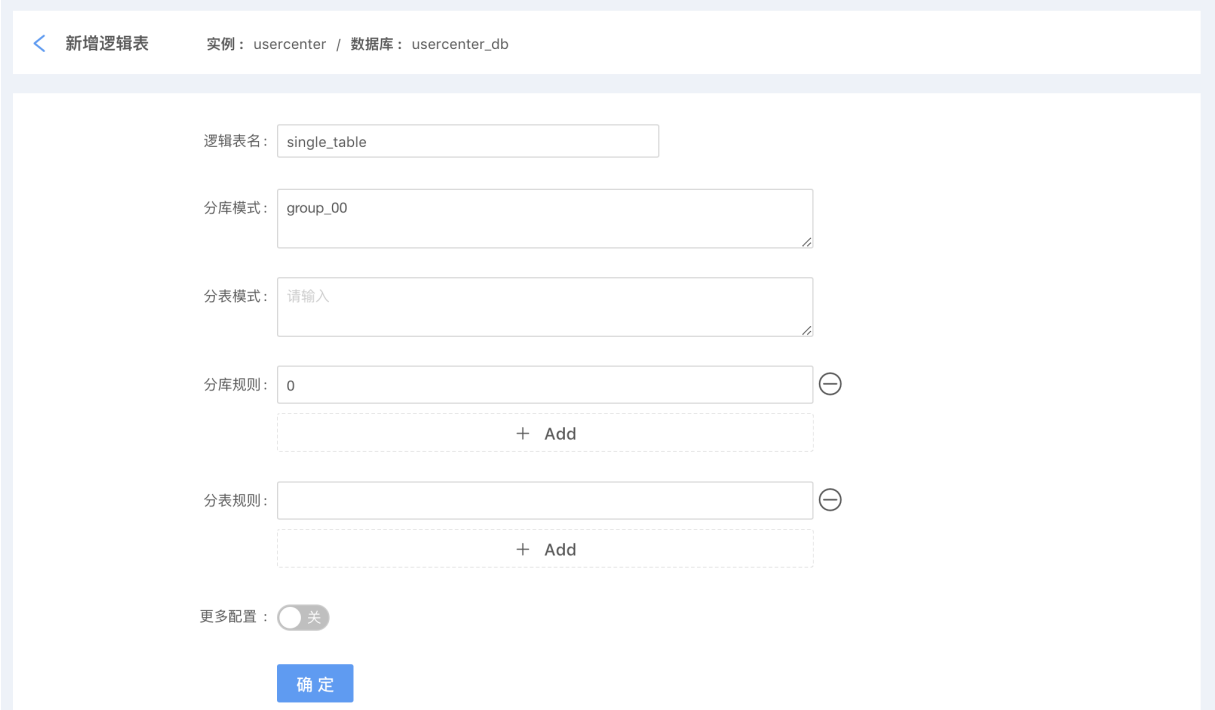
将 ODP 的 `idleTimeoutMinutes` 值修改为小于 RDS 的 `wait_timeout` 的值（idleTimeoutMinutes 默认是 12 分钟，建议使用默认值即可），若您有特别需求，该值也不要和 RDS 的 `wait_timeout` 设置的值相近。

17.注意事项

分库分表规则

数据访问代理可以根据任意字段、多个字段进行分库分表，且同一个分表可以配置多个规则。如果配置了分表规则后，所有 SQL 请求都需要带上分库分表字段。

如果数据库分片集群是分库分表模式，则所有数据库分表都需要配置分库分表规则，如果数据库分片中只有单表，分库分表规则可以直接配置 `0` 即可。在如下配置中，`single_table` 会路由到 `group_00` 分片：



使用限制

- 仅支持 DML 语句（INSERT、SELECT、UPDATE、DELETE）。
- 分库分表场景下不支持 JOIN。
- 分库分表场景下不支持自增 ID。
- 不支持 `SELECT LAST_INSERT_ID()` 。