阿里云
云数据库 PolarDB

PolarDB PostgreSQL数据库

文档版本：20200709

法律声明

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

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

通用约定

格式	说明	样例
	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。	 **禁止：** 重置操作将丢失用户配置数据。
	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。
	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。
	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。	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 概述

PolarDB是阿里云自研的下一代关系型云数据库，兼容MySQL、PostgreSQL、Oracle引擎，存储容量最高可达100TB，单库最多可扩展到16个节点，适用于企业多样化的数据库应用场景。

PolarDB采用存储和计算分离的架构，所有计算节点共享一份数据，提供分钟级的配置升降级、秒级的故障恢复、全局数据一致性和免费的数据备份容灾服务。PolarDB既融合了商业数据库稳定可靠、高性能、可扩展的特征，又具有开源云数据库简单开放、自我迭代的优势。

PolarDB兼容Oracle数据库提供公共云和专有云形态，其中专有云形态支持CentOS、UOS、麒麟等操作系统，支持X86架构CPU以及ARM架构CPU（飞腾等）。

基本概念

- 集群

  一个集群包含一个主节点以及最多15个只读节点（最少一个，用于提供Active-Active高可用）。集群ID以pc开头（代表PolarDB cluster）。
- 节点

  一个独立占用物理内存的数据库服务进程。节点ID以pi开头（代表PolarDB instance）。
- 数据库

  在节点下创建的逻辑单元，一个节点可以创建多个数据库，数据库在节点内的命名唯一。
- 地域和可用区

  地域是指物理的数据中心。可用区是指在同一地域内，拥有独立电力和网络的物理区域。更多信息请参考阿里云全球基础设施。

控制台

阿里云提供了简单易用的Web控制台，方便您操作阿里云的各种产品和服务，包括云数据库PolarDB。在控制台上，您可以创建、连接和配置PolarDB数据库。

关于控制台的界面介绍，请参考阿里云管理控制台。

PolarDB控制台地址：PolarDB控制台。

2 PolarDB-P发布记录

本文介绍PolarDB-P的功能动态。

PolarDB-P对应的原生PostgreSQL版本如下所示。

PolarDB-P版本	原生PostgreSQL版本
V1.1.1	11.2
V1.1.0	11.2

V1.1.1

功能	功能概述
扩展插件	• 新增polar_proxy_utils插件，用于管理与proxy相关的功能，主要支持只读UDF和只读表的配置，允许通过集群地址将只读UDF以及只读表的访问路由到只读节点。 • 新增polar_resource_group插件，支持自定义资源隔离组，基于user、database、session粒度，通过cpu、memory维度进行资源隔离限制。
性能优化	• 数据库计算节点和文件系统解耦，文件系统可独立运维，大幅提高数据库的可靠性和可用性。 • 使用单调递增版本号替代原有的活跃事务列表快照，大幅提升数据库事务处理性能。 • 执行计划优化，避免使用过旧的统计信息。
缺陷修复	• 修复了插件timescaledb在申请内存时出错时进程的异常问题。 • 修复了IO监控功能中进程退出后没有汇总统计信息。 • 修复了lock_debug开启后，可能发生空指针异常问题。 • 修复了特定情况下导致pg_cron插件不可用的问题。 • 修复了社区已知的DSM死锁问题。 • 修复了用户连接数超限的问题。

功能	功能概述
Ganos 2.8	• 栅格数据元数据访问接口增强： - 新增ST_XMin获取栅格数据X方向最小值。 - 新增ST_YMin获取栅格数据Y方向最小值。 - 新增ST_XMax获取栅格数据X方向最大值。 - 新增ST_YMax获取栅格数据Y方向最大值。 - 新增ST_ChunkHeight获取栅格数据分块高度。 - 新增ST_ChunkWidth获取栅格数据栅格数据分块宽度。 - 新增ST_ChunkBands获取栅格数据栅格数据分块波段数量。 • 新增ST_SrFromEsriWkt函数支持Esri格式空间参考字符串转换为OGC 格式空间参考字符串。 • 新增栅格数据类型支持zstd和snappy压缩方式。 • 新增点云数据类型支持二进制拷贝功能。 • 新增支持PROJ_LIB和GDAL_DATA环境变量设置，同时部署相关数据。 • 修复OSS路径非法导致数据库崩溃问题。 • 修复部分栅格数据导入SRID与定义不一致的问题。

V1.1.0

功能	功能描述
内置视图	优化polar_stat_activity视图，新增wait_info列和wait_time列，分别用于监控进程等待对象（pid或fd）的等待时长。
扩展插件	• 新增polar_concurrency_control插件，可以对事务执行、SQL查询、存储过程、DML等操作进行并发限制。您可以自定义大查询，并对大查询进行并发限制，优化高并发下的执行性能。 • 新增oss_fdw插件，用于Aliyun OSS外部表支持。您可以通过OSS外部表将数据库数据写入OSS，也可以通过OSS外部表将OSS数据加载到数据库中，OSS外部表支持并行和压缩，极大提高了导入和导出数据的性能，同时也可以使用这个功能来实现多类型存储介质的冷热数据存储。
性能优化	提供插入索引时索引页的预扩展功能，用于提升将数据插入带有索引的表的执行性能。

功能	功能描述
Ganos 2.7	• 新增空间栅格对象的MD5操作函数，可以用于数据的一致性检查和去重等操作： - 新增ST_MD5Sum函数，用于获取栅格对象的MD5码值。 - 新增ST_SetMD5Sum函数，用于设置栅格对象的MD5码值。 • 新增空间栅格对象OSS认证方式操作函数： - 新增ST_AKId函数，用于获取以OSS方式存储的栅格对象的AcessKey ID。 - 新增ST_SetAccessKey函数，用于设置以OSS方式存储的栅格对象的AcessKey ID和AccessKey Secret。 - 新增ST_SetAKId函数，用于设置以OSS方式存储的栅格对象的AcessKey ID。 - 新增ST_SetAKSecret函数，用于设置以OSS方式存储的栅格对象的AcessKey Secret。 • 新增空间栅格元数据操作函数： - 新增ST_ScaleX函数，用于获取栅格对象在空间参考系下X方向像素宽度。 - 新增ST_ScaleY函数，用于获取栅格对象在空间参考系下Y方向像素宽度。 - 新增ST_SetScale函数，用于设置栅格对象在空间参考系下像素宽度。 - 新增ST_SkewX函数，用于获取栅格对象在空间参考系下X方向旋转。 - 新增ST_SkewY函数，用于获取栅格对象在空间参考系下Y方向旋转。 - 新增ST_SetSkew函数，用于设置栅格对象在空间参考系下旋转。 - 新增ST_UpperLeftX函数，用于获取栅格对象在空间参考系下左上角点的X坐标。 - 新增ST_UpperLeftY函数，用于获取栅格对象在空间参考系下左上角点的Y坐标。 - 新增ST_SetUpperLeft函数，用于获取栅格对象在空间参考系下左上角点坐标。 - 新增ST_PixelWidth函数，用于获取栅格对象在空间参考系下像素宽度。 - 新增ST_PixelHeight函数，用于获取栅格对象在空间参考系下像素高度。 • 修复由于聚集函数导致扩展升级失败的问题。

3 PolarDB PostgreSQL快速入门

快速入门旨在介绍如何创建PolarDB PostgreSQL集群、进行基本设置以及连接数据库集群，使您能够了解从购买PolarDB到开始使用的流程。

使用流程

通常，从购买PolarDB（创建新集群）到可以开始使用，您需要完成如下操作。

1. 创建PolarDB PostgreSQL数据库集群
2. 设置集群白名单
3. 创建数据库账号
4. 查看连接地址
5. 连接数据库集群

4 数据迁移方案概览

云数据库POLARDB提供了多种数据迁移方案，可满足不同上云、迁云的业务需求，使您可以在不影响业务的情况下平滑将数据库迁移至阿里云云数据库POLARDB上面。

通过使用阿里云数据传输服务（DTS），您可以实现POLARDB的结构迁移、全量迁移等。

数据迁移

使用场景	文档链接
从RDS迁移至POLARDB	从RDS for PostgreSQL迁移至POLARDB for PostgreSQL
从自建数据库迁移至POLARDB	从自建PostgreSQL迁移至POLARDB for PostgreSQL

5 数据迁移

5.1 从自建PostgreSQL迁移至POLARDB for PostgreSQL

本文介绍通过pg_dumpall、pg_dump和pg_restore命令将自建PostgreSQL数据库迁移至POLARDB for PostgreSQL中。

迁移的源库为RDS for PostgreSQL实例时，请参考从RDS for PostgreSQL迁移至POLARDB for PostgreSQL。

前提条件

POLARDB for PostgreSQL实例的存储空间应大于自建PostgreSQL数据库的存储空间。

注意事项

该操作为全量数据迁移。为避免迁移前后数据不一致，迁移操作开始前请停止自建数据库的相关业务，并停止数据写入。

准备工作

1. 创建一个Linux操作系统的ECS实例，本案例使用的ECS为Ubuntu 16.04 64位操作系统。详情请参考创建ECS实例。

   说明:

   - 要求ECS实例和迁移的目标POLARDB for PostgreSQL实例处于同一个专有网络。
   - 可创建一个按量付费的ECS实例，迁移完成后释放实例。

2. 在ECS实例中安装PostgreSQL，以便执行数据恢复的命令。详情请参考PostgreSQL官方文档。

   说明:

   请确保安装的PostgreSQL数据库版本与自建PostgreSQL数据库版本一致。

操作步骤一 备份自建数据库

该操作为全量数据迁移。为避免迁移前后数据不一致，迁移操作开始前请停止自建数据库的相关业务，并停止数据写入。

1. 在自建PostgreSQL数据库服务器上执行以下命令，备份数据库中的所有角色信息。

   ```
   pg_dumpall -U <username> -h <hostname> -p <port> -r -f <filename>
   ```

   参数说明：

   - <username>：登录自建PostgreSQL数据库的账号。
   - <hostname>：自建PostgreSQL数据库的连接地址，本机可使用localhost。
   - <port>：数据库服务的端口号。
   - <filename>：生成的备份文件名称。

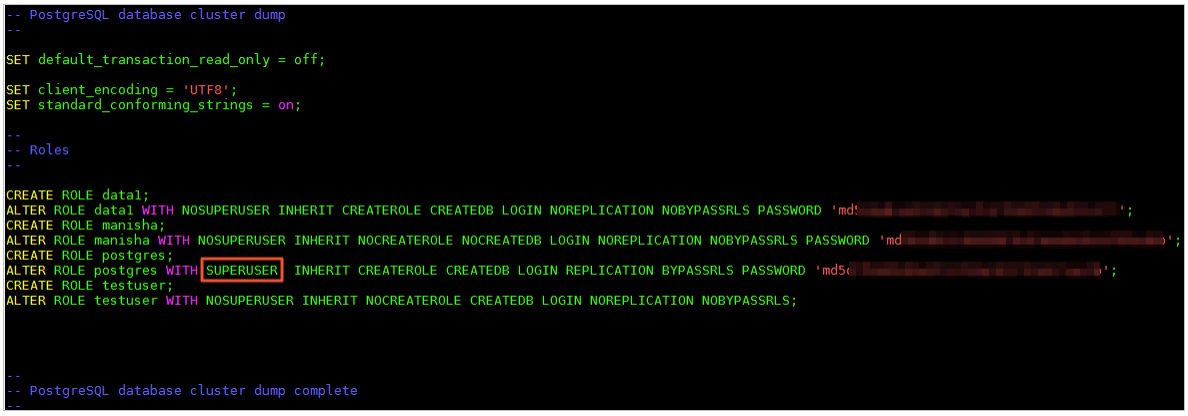
   示例：

   ```
   pg_dumpall -U postgres -h localhost -p 5432 -r -f roleinfo.sql
   ```

2. 命令行提示Password:时，输入数据库账号对应的密码，开始备份数据库中的所有角色信息。
3. 使用vim命令将角色信息备份文件中的SUPERUSER替换为polar_superuser。

   说明：

   如果角色信息备份文件中没有SUPERUSER信息，可跳过本步骤。



4. 在自建PostgreSQL数据库服务器上执行以下命令，备份数据库中的数据。

```
pg_dump -U <username> -h <hostname> -p <port> <dbname> -Fd -j <njobs> -f <
dumpdir>
```

参数说明：

- <username>：登录自建PostgreSQL数据库的账号。
- <hostname>：自建PostgreSQL数据库的连接地址，本机可使用localhost。
- <port>：数据库服务的端口号。
- <dbname>：要备份的数据库名。
- <njobs>：同时执行备份作业的并发数。

  说明：

  - 参数<njobs>可减少转储的时间，但也会增加数据库服务器的负载。
  - 如果您的自建PostgreSQL数据库是9.2以前的版本，您还需要指定--no-synchronized-snapshots参数。
- <dumpdir>：生成的备份文件所属目录。

示例：

```
pg_dump -U postgres -h localhost -p 5432 mytestdata -Fd -j 5 -f postgresdump
```

5. 命令行提示Password:时，输入数据库账号对应的密码，数据库开始备份。
6. 等待备份完成，PostgreSQL数据库数据将备份至指定的目录中，本案例为postgresdump。

操作步骤二 数据迁移至POLARDB for PostgreSQL

1. 将备份文件所属的目录上传至ECS实例中。

   说明：
   包含角色信息备份文件和数据库备份文件。

2. 在ECS上执行以下命令，将角色信息备份文件中的角色信息迁移至POLARDB for PostgreSQL实例中。

```
psql -U <username> -h <hostname>  -p <port> -d <dbname>  -f <filename>
```

参数说明：

- <username>：登录POLARDB for PostgreSQL数据库的账号。
- <hostname>：POLARDB for PostgreSQL实例的主地址（私网）。
- <port>：数据库服务的端口号，默认为**1921**。
- <dbname>：连接的数据库的名称，默认为postgres。
- <filename>：角色信息备份文件名。

```
psql -U gctest -h pc-xxxxxxxx.pg.polardb.cn-qd-pldb1.rds.aliyuncs.com -d postgres -p 1921 -f roleinfo.sql
```

3. 命令行提示Password:时，输入数据库账号对应的密码，角色信息开始导入。
4. 在ECS上执行以下命令，将数据库数据恢复至POLARDB for PostgreSQL实例中。

```
pg_restore -U <username> -h <hostname> -p <port> -d <dbname> -j <njobs> <dumpdir>
```

参数说明：

- <username>：登录POLARDB for PostgreSQL数据库的账号。
- <hostname>：POLARDB for PostgreSQL实例的主地址（私网），详情请参考#unique_15。
- <port>：数据库服务的端口号，默认为**1921**。
- <dbname>：连接并直接恢复到的目标数据库名。

  说明：
  目标数据库需已存在，如不存在请在目标实例中创建该数据库。

- <njobs>：同时执行数据恢复作业的并发数。

  说明：

此选项可减少数据恢复的时间，但也会增加数据库服务器的负载。

- <dumpdir>：备份文件所在目录。

示例：

```
pg_restore -U gctest -h pc-mxxxxxxxx.pg.polardb.cn-qd-pldb1.rds.aliyuncs.com -p 1921 -d mytestdata -j 6 postgresdump
```

5. 命令行提示 Password:时，输入数据库账号对应的密码，数据开始迁移。

说明：

如果忘记密码，请参考#unique_16/unique_16_Connect_42_section_ckb_hpq_tdb。

等待数据迁移完成即可。

5.2 从RDS for PostgreSQL迁移至POLARDB for PostgreSQL

本文介绍通过pg_dump和pg_restore命令将自建PostgreSQL数据库迁移至POLARDB for PostgreSQL中。

迁移的源库为自建PostgreSQL数据库时，请参考从自建PostgreSQL迁移至POLARDB for PostgreSQL。

前提条件

POLARDB for PostgreSQL实例的存储空间应大于RDS for PostgreSQL实例的存储空间。

注意事项

该操作为全量数据迁移。为避免迁移前后数据不一致，迁移操作开始前请停止自建数据库的相关业务，并停止数据写入。

准备工作

1. 创建一个Linux操作系统的ECS实例，本案例使用的ECS为Ubuntu 16.04 64位操作系统。详情请参考创建ECS实例。

说明：

- 要求ECS实例和迁移的目标POLARDB for PostgreSQL实例处于同一个专有网络。
- 可创建一个按量付费的ECS实例，迁移完成后释放实例。

2. 在ECS实例中安装PostgreSQL，以便执行数据恢复的命令。详情请参考PostgreSQL官方文档。

说明：

请确保安装的PostgreSQL数据库版本与自建PostgreSQL数据库版本一致。

操作步骤一 备份RDS for PostgreSQL数据库

该操作为全量数据迁移。为避免迁移前后数据不一致，迁移操作开始前请停止自建数据库的相关业务，并停止数据写入。

1. 在ECS上执行以下命令，备份数据库中的数据。

```
pg_dump -U <username> -h <hostname> -p <port> <dbname> -Fd -j <njobs> -f <dumpdir>
```

参数说明：

- <username>：登录自建PostgreSQL数据库的账号。
- <hostname>：自建PostgreSQL数据库的连接地址，本机可使用localhost。
- <port>：数据库服务的端口号。
- <dbname>：指定要连接的数据库的名称，默认为postgres。
- <njobs>：同时执行备份作业的并发数。

  说明：

  - 参数<njobs>可减少转储的时间，但也会增加数据库服务器的负载。
  - 如果您的自建PostgreSQL数据库是9.2以前的版本，您还需要指定--no-synchronized-snapshots参数。
- <dumpdir>：生成的备份文件所属目录。

示例：

```
pg_dump -U postgres -h localhost -p 5432 postgres -Fd -j 5 -f postgresdump
```

2. 命令行提示Password:时，输入数据库账号对应的密码，数据库开始备份。
3. 等待备份完成，PostgreSQL数据库数据将备份至指定的目录中，本案例为postgresdump。

操作步骤二 数据迁移至POLARDB for PostgreSQL

1. 在ECS上连接POLARDB for PostgreSQL数据库。

   ```
   psql -U <username> -h <hostname> -p <port> -d <dbname>
   ```

   参数说明：

   - <username>：登录POLARDB for PostgreSQL数据库的账号。
   - <hostname>：POLARDB for PostgreSQL实例的主地址（私网），详情请参考#unique_15。
   - <port>：数据库服务的端口号，默认为**1921**。
   - <dbname>：要连接的数据库名称。

   示例：

   ```
   psql -h pc-mxxxxxxxx.pg.polardb.cn-qd-pldb1.rds.aliyuncs.com -p 3433 -d postgres -U gctest
   ```

2. 根据源RDS for PostgreSQL实例数据库中的角色信息，在目标POLARDB for PostgreSQL实例中创建角色信息，并对数据恢复的目标数据库进行授权，详情请参考官方文档CREATE ROLE和GRANT。
3. 在ECS上执行以下命令，将数据库数据迁移至POLARDB for PostgreSQL实例中。

   ```
   pg_restore -U <username> -h <hostname> -p <port> -d <dbname> -j <njobs> <dumpdir>
   ```

   参数说明：

   - <username>：登录POLARDB for PostgreSQL数据库的账号。
   - <hostname>：POLARDB for PostgreSQL实例的主地址（私网）。
   - <port>：数据库服务的端口号，默认为**1921**。
   - <dbname>：连接并直接恢复到的目标数据库名。

     说明：
     目标数据库需已存在，如不存在请在目标实例中创建该数据库。

   - <njobs>：同时执行数据恢复作业的并发数。

     说明：

此选项可减少数据恢复的时间，但也会增加数据库服务器的负载。

- <dumpdir>：备份文件所在目录。

示例：

```
pg_restore -U gctest -h pc-mxxxxxxxx.pg.polardb.cn-qd-pldb1.rds.aliyuncs.com -p
1921 -d postgres -j 6 postgresdump
```

4. 命令行提示Password:时，输入数据库账号对应的密码，数据开始迁移。

说明：

如果忘记密码，请参考#unique_17/unique_17_Connect_42_section_ckb_hpq_tdb。

等待数据迁移完成即可。

6 读写分离

PolarDB PostgreSQL集群自带读写分离功能，通过一个集群地址（读写分离地址）实现读写请求的自动转发。

背景信息

在对数据库有少量写请求，但有大量读请求的应用场景下，单个节点可能无法承受读取压力，甚至对业务产生影响。使用集群地址（读写分离地址），就可以使写请求自动转发到主节点，读请求自动转发到各个只读节点。从而实现读取能力的弹性扩展，满足大量的数据库读取需求。

功能优势

- 统一读写分离地址，方便维护。

  您未使用集群地址（读写分离地址）时，需要在应用程序中分别配置主节点和每个只读节点的连接地址，才能实现将写请求发往主节点而将读请求发往只读节点。PolarDB提供一个集群地址（读写分离地址），您连接该地址后即可对主节点和只读节点进行读写操作，读写请求被自动转发到对应节点，可降低维护成本。同时，您只需添加只读节点的个数，即可不断扩展系统的处理能力，应用程序无需做任何修改。
- Session级别的读一致性。

  当客户端通过读写分离建立与后端的连接后，读写分离中间件会自动与主节点和各个只读节点建立连接。在同一个连接内（同一个session内），读写分离中间件会根据各个数据库节点的数据同步程度，来选择合适的节点，在保证数据正确的基础上（写操作之后的读有正确的结果），实现读写请求的负载均衡。
- 扩展查询（prepare语句或命令）的负载均衡。

  读写分离中间件会自动根据excute语句或命令中的信息来找到之前执行prepare语句或命令的数据库节点，从而做到扩展查询的负载均衡。
- 支持原生高安全链路，提升性能。

  如果您在云上自行搭建代理层实现读写分离，数据在到达数据库之前需要经历多个组件的语句解析和转发，对响应延迟有较大的影响。而PolarDB读写分离中间件隶属于集群组件，相比外部组件而言，能够有效降低延迟，提升处理速度。
- 节点健康检查，提升数据库系统的可用性。

  读写分离模块将自动对主节点和只读节点进行健康检查，当发现某个节点出现宕机或者延迟超过阈值时，将不再分配读请求给该节点，读写请求在剩余的健康节点间进行分配。以此确保单个只

读节点发生故障时，不会影响应用的正常访问。当节点被修复后，该节点会自动被加入请求分配体系内。

功能限制

- 暂不支持如下命令或功能：
  - 不支持Replication-mode方式进行建连，即不支持通过读写分离地址自行搭建主备复制集群。如需自行搭建主备复制集群，请使用主节点的连接地址。
  - 不支持临时表的ROWTYPE。

    ```
    create temp table fullname (first text, last text);
    select '(Joe,von Blow)'::fullname, '(Joe,d''Blow)'::fullname;
    ```

  - 不支持函数中创建临时资源。
    - 函数中创建临时表，然后再执行对临时表的查询SQL可能会收到表不存在的错误信息。
    - 函数中带有prepare语句，后续execute时可能会收到statement name不存在的错误信息。
- 路由相关限制：
  - 事务中的请求都路由到主节点，事务退出后，恢复负载均衡。
  - 所有使用函数（除聚合函数，例如，count、sum）的语句，会路由到主节点。

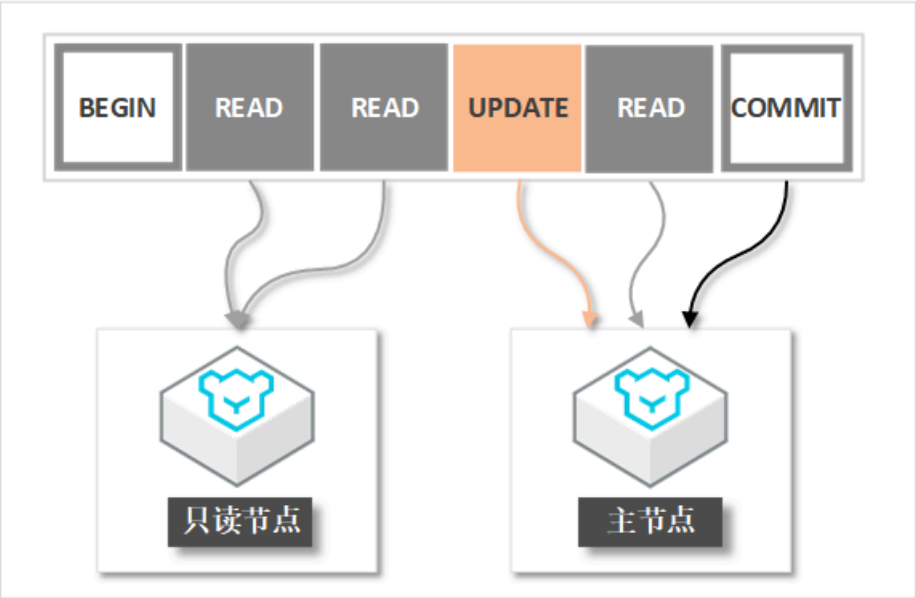
创建或修改集群地址

- 创建自定义集群地址操作方式请参见创建自定义集群地址。
- 修改集群地址操作方式请参见修改集群地址。

高级选项-事务拆分

在默认的Read Commited隔离级别下，当PolarDB接收到开启事务的语句（例如begin或set autocommit=0）时，不会立即开启事务，而是在发生写操作时才正式开启事务。

默认情况下，PolarDB会将事务内的所有请求都发送到主节点以保障事务的正确性，但是某些框架会将所有请求封装到事务中，导致主节点负载过大。此时您可以开启事务拆分功能，开启后PolarDB会识别当前事务的状态，将正式开启事务前的读请求通过负载均衡模块分流至只读节点。

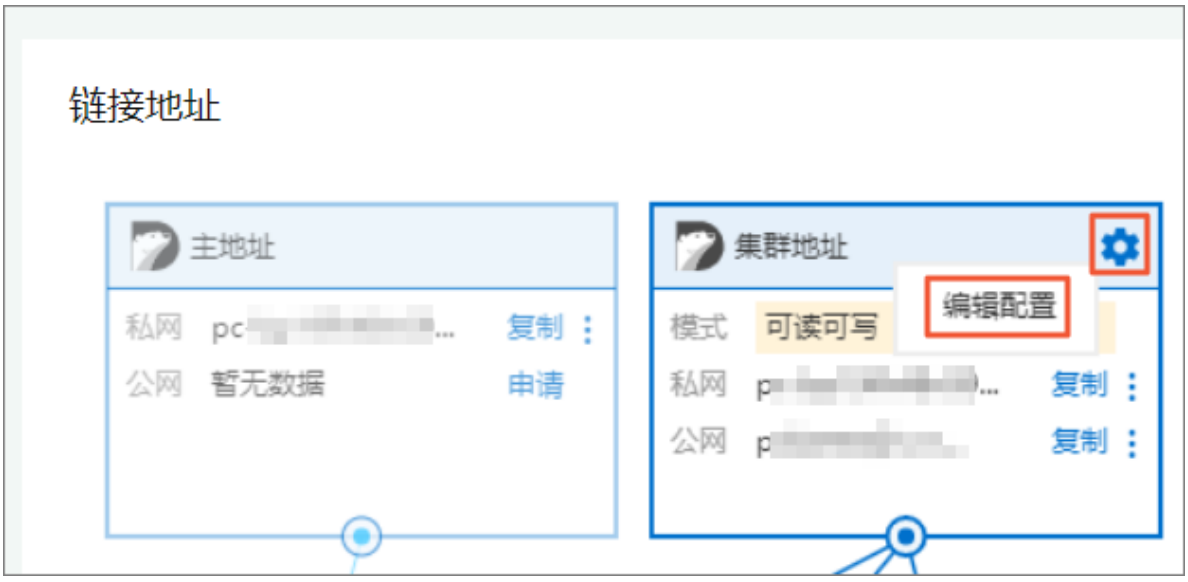


说明：

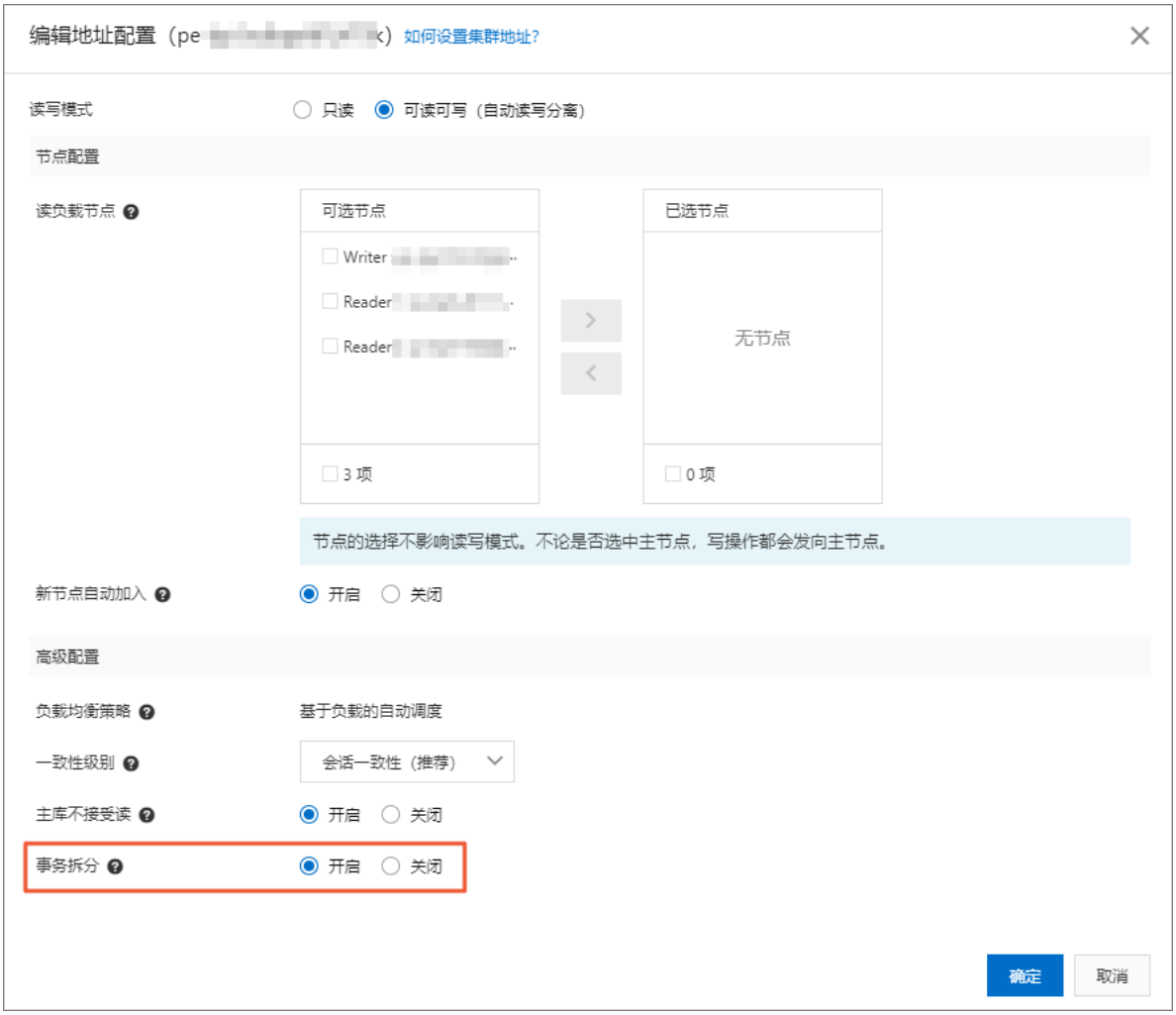
某些业务对全局一致有要求，开启事务拆分后将不满足全局一致，因此在事务拆分前请充分评估事务拆分功能是否适用于您的业务。

开启事务拆分操作如下：

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 找到目标集群，单击集群ID。
4. 在区域，找到目标集群地址，单击目标集群名称右侧的 。



5. 在**编辑地址配置**对话框内，开启**事务拆分**。



说明:

开启事务拆分后将对新的连接生效，现有的连接需重新连接才生效。

6. 单击**确定**。

高级选项-一致性级别

详情请参见PolarDB PostgreSQL一致性级别。

自定义路由-Hint

在SQL语句前加上/*FORCE_MASTER*/或/*FORCE_SLAVE*/可以强制指定这条SQL的路由方向。例如select * from test默认会路由到只读节点，改为/*FORCE_MASTER*/ select * from test就会路由到主节点。

下一步

连接数据库集群

说明：

申请集群地址后，您只需在应用中配置该集群地址，即可实现自动读写分离。

相关API

API	描述
#unique_22	创建PolarDB集群的公网地址。
#unique_23	查询PolarDB集群的地址信息。
#unique_24	修改PolarDB集群地址属性。
#unique_25	修改PolarDB集群默认访问地址前缀。
#unique_26	释放PolarDB集群地址（除了自定义集群地址的私网地址）。

7 使用存储包

为了更好地帮助您降低存储成本，PolarDB推出了预付费形式的存储包。

背景信息

PolarDB的存储空间无需手动配置，根据数据量自动伸缩，您只需为实际使用的数据量按量付费。当您的数据量较大时，推荐您使用PolarDB存储包，相比按量付费，预付费购买存储包更加优惠，购买的容量越大，优惠力度就越大。

存储空间费用

PolarDB存储空间费用请参见#unique_28/unique_28_Connect_42_section_u9d_n9d_3jt。

存储包和按量付费的价格对比

下表为按月计费存储包和按量付费的价格对比。 使用容量越大的存储包，价格越优惠。

容量（GB）	中国内地		中国香港及海外	
	按量付费（元/月）	存储包（元/月）	按量付费（元/月）	存储包（元/月）
50	175	175	195	195
100	350	350	390	390
200	700	700	780	780
300	1,050	1,050	1,170	1,170
500	1,750	1,750	1,950	1,950
1,000	3,500	3,150	3,900	3,510
2,000	7,000	6,300	7,800	7,020
3,000	10,500	7,800	11,700	8,600
5,000	17,500	13,000	19,500	14,400
10,000	35,000	21,000	39,000	23,400
20,000	70,000	42,000	78,000	46,800
30,000	105,000	63,000	117,000	70,200
50,000	175,000	96,000	195,000	106,900
100,000	350,000	192,000	390,000	213,900

注意事项

- 每种类型资源包只允许购买一个，存储包容量不够时，可以升级存储包。
- 存储包暂不支持降级。
- 存储包由**资源包类型**规定的地域内的所有集群共享使用。
- 超出存储包容量的部分以按量付费的方式收取费用。例如您有三个存储容量均为400GB的PolarDB集群，这三个集群可以共享1个1000GB的存储包，超出的200GB则按量付费，您可以在**费用中心**查看存储包抵扣量。

> **说明:**
> 更多关于存储包的说明，请参见常见问题。

购买存储包

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 单击页面左上角。
4. 单击**存储包**页签，设置如下参数。

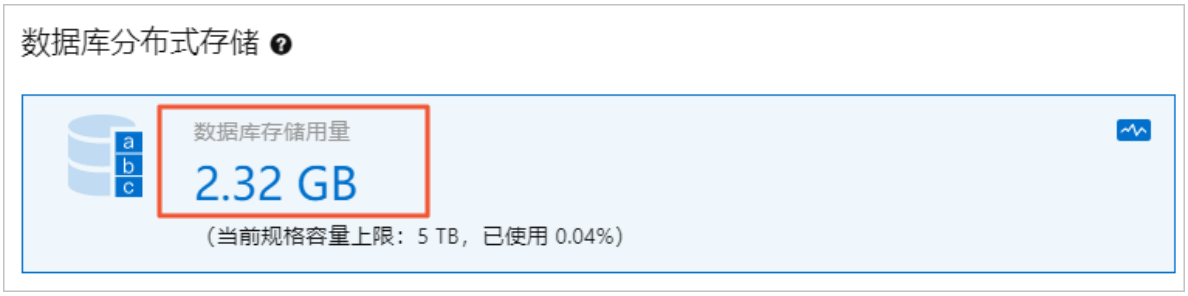
参数	说明
资源包类型	• **中国内地通用**：购买后可用于中国内地地域的PolarDB集群。 • **中国香港及海外通用**：购买后可用于中国香港及海外地域的PolarDB集群。
存储包规格	存储包的容量大小。
购买时长	购买存储包的时长。

5. 单击**立即购买**。

6. 勾选服务协议，单击**去支付**完成支付。

查看数据库存储用量

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 找到目标集群，单击集群ID。
4. 在页面的**数据库分布式存储**区域，查看**数据库存储用量**。

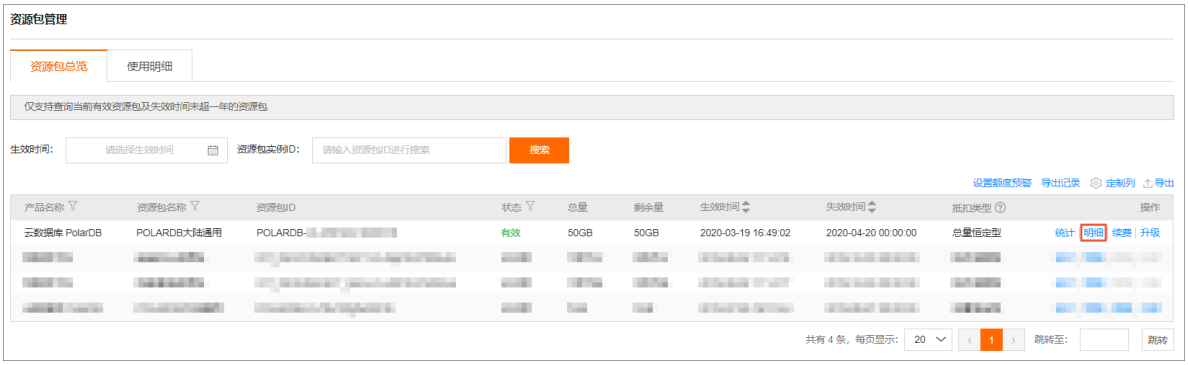


查看存储包抵扣量

说明：

仅支持查看当前有效资源包及失效时间未超一年的资源包。

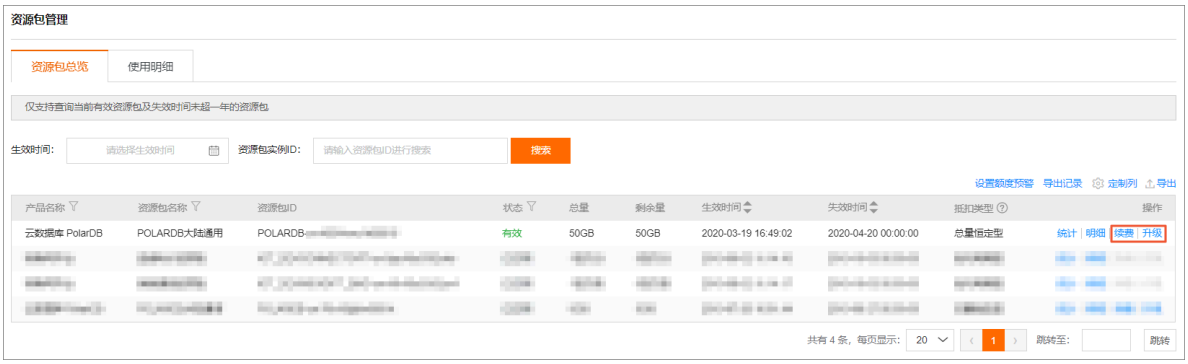
1. 登录费用中心。
2. 在左侧导航栏中，单击**资源管理** > **资源包**。
3. 在**资源包管理**页面的**资源包总览**页签，找到目标资源包，单击**操作**栏中的**明细**。



4. 在**使用明细**页签，查看资源包使用情况。

续费或升级存储包

1. 登录费用中心。
2. 在左侧导航栏中，单击**资源管理 > 资源包**。
3. 在**资源包管理**页面的**资源包总览**页签，找到目标资源包，单击**操作**栏中的**续费**或**升级**。



4. 您可以按以下方法进行续费或升级操作：

   - **续费**

     a. 单击**操作**列的**续费**。

     b. 选择**续费时长**，勾选服务协议，单击**去支付**完成支付。

   - **升级**

     a. 单击**操作**列的**升级**。

     b. 选择**存储包规格**，勾选服务协议，单击**去支付**完成支付。

常见问题

- 存储包是否跟集群绑定售卖？

  答：不绑定。您需要单独购买存储包，购买后会自动抵扣相应地域内的集群存储空间。

- 存储包是否可以被多个集群共享？

  答：可以。存储包由**资源包类型**（中国内地或中国香港及海外）规定的地域内的所有集群共享使用。

- 存储包是否可以被不同引擎的集群共享？

  答：可以，存储包可以同时用于PolarDB MySQL/PolarDB-P/PolarDB-O集群。

- 存储包容量不够用或用完了怎么办？可以再买一个吗？

  答：每种类型的存储包只能购买一个，您可以通过升级存储包的方式解决，具体请参见续费或升级存储包。

- 当前数据量超出存储包容量的部分如何计费？

  答：超出存储包容量的部分以按量付费的方式收取费用，具体请参见#unique_28/unique_28_Connect_42_section_u9d_n9d_3jt。

- 存储包支持3TB和5TB，业务只需要4TB，如何选购？

  答：您可以先购买3TB，存储量接近5TB时再升级为5TB。

- 存储包要到期了怎么续费？

  答：您可以在用户中心对即将到期的存储包进行续费，具体请参见续费或升级存储包。

8 待处理事件

当PolarDB出现待处理事件时，会在控制台提醒您及时处理。

前提条件

您有未处理的运维事件。

说明：

当您有未处理的运维事件时，可以在控制台左侧导航栏的**待处理事件**看到提醒。

背景信息

PolarDB运维事件（数据库软件升级、硬件维护与升级）除了在短信、语音、邮件或站内信通知之外，还会在控制台进行通知。您可以查看具体的事件类型、任务ID、集群名称、切换时间等，也可以手动修改切换时间。

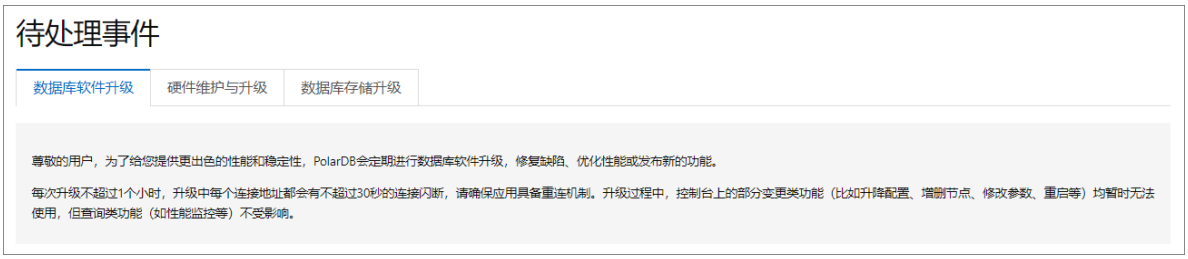
修改切换时间

1. 登录PolarDB控制台。
2. 在左侧导航栏中，单击**待处理事件**。

说明：

强制要求预约时间的运维事件会弹窗提醒，请尽快完成预约。

3. 在**待处理事件**页面，选择相应的事件类型。



说明：

不同的事件类型页面会显示不同的通知信息。

4. 在下方事件列表查看事件的详细信息，如需修改**切换时间**，请在左侧勾选对应的实例，然后单击**修改切换时间**，在弹出的对话框中设置时间并单击**确认**。

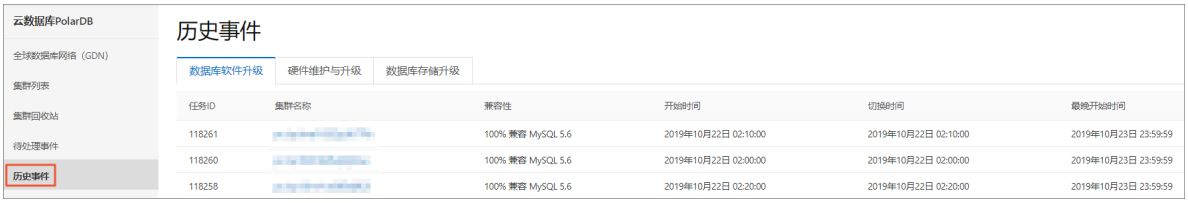
说明：

切换时间不能晚于**最晚操作时间**。

历史事件

您可以在PolarDB控制台里查看已完成的事件。

1. 登录PolarDB控制台。
2. 在左侧导航栏中，单击**历史事件**。



3. 在**历史事件**页面，选择相应的事件类型。不同的事件类型页面会显示不同的信息。

9 设置集群白名单

创建PolarDB PostgreSQL数据库集群后，您需要设置PolarDB集群的IP白名单，并创建集群的初始账号，才能连接和使用该集群。

注意事项

- 默认情况下，IP白名单只包含IP地址127.0.0.1，表示任何IP地址均无法访问该数据库集群。
- 若将IP白名单设置为%或者0.0.0.0/0，表示允许任何IP地址访问数据库集群。该设置将极大降低数据库的安全性，如非必要请勿使用。
- PolarDB暂不支持自动获取VPC中的ECS内网IP以供您选择，请手动填写需要访问PolarDB的ECS内网IP。

设置白名单

1. 登录PolarDB控制台。
2. 在页面左上角，选择实例所在地域。
3. 单击目标集群ID，进入页面。
4. 单击。
5. 在**集群白名单**页面，可以**配置**已有白名单或**新增IP白名单分组**。

   - 单击栏中的，配置该IP白名单。
   - 单击，可以新增IP白名单。

6. 填写白名单，单击。
   - 如果您的ECS服务器需要访问PolarDB，可在ECS**实例详情**页面**配置信息**区域，查看ECS服务器的IP地址，然后填写到白名单中。

> **说明：**

如果ECS与PolarDB位于同一地域（例如，华东1），填写ECS的私网IP地址；如果ECS与PolarDB位于不同的地域，填写ECS的公网IP地址，或者将ECS迁移到PolarDB所在地域后填写ECS私网IP地址。

- 如果您本地的服务器、电脑或其它云服务器需要访问PolarDB，请将其IP地址添加到白名单中。

下一步

设置集群白名单以及创建数据库账号后，您就可以连接数据库集群，对数据库进行操作。

- 创建数据库账号
- 连接数据库集群

常见问题

1. 已添加ECS的IP地址到IP白名单中，但是还是无法访问。

   答：

   a. 确认IP白名单是否正确。如果是通过内网地址访问，需添加ECS的私网IP地址。如果是通过公网地址进行访问，需添加ECS的公网IP地址。

   b. 确认网络类型是否一致。如果ECS实例的网络类型是经典网络，可参考经典网络迁移到专有网络方案将ECS实例迁移至PolarDB所在的专有网络。

   说明：
   如果该ECS 还要访问其他经典网络内网资源，请勿操作，因为迁移后会无法访问经典网络。

   或者通过Classlink打通经典网络到专有网络的网络。

   c. 确认是否位于同一个VPC。如果不是，需要重新购买一个PolarDB，或者通过云企业网来打通两个VPC网络实现访问。

2. 什么原因导致公网连接失败？

   答：

   a. 如果是ECS通过公网地址进行访问，请确认添加的是ECS的公网IP地址，而不是私网IP地址。

   b. IP白名单设置为0.0.0.0/0，然后尝试访问，如果能成功访问，表示IP白名单中之前填写的公网地址错误。请参考查看连接地址确定真正的公网地址。

3. 如何实现内网连接?

   答：ECS和PolarDB内网访问需要满足以下条件：

   - 相同地域。
   - 相同网络类型，如果是VPC，需要在相同VPC下。
   - ECS内网IP在PolarDB集群IP白名单中。

4. 如何限制某个用户只能从特定的IP地址访问PolarDB?

   答：可以创建高权限账号，然后使用高权限账号对普通账号限定访问IP。

```
1
2
3 CREATE USER 'alitest'@'192.168.1.101' ;
4
5
6 select * from mysql.user where user='alitest';
```

相关API

API	描述
#unique_31	查看允许访问数据库集群的IP名单。
#unique_32	修改允许访问数据库集群的IP名单。

10 计费

10.1 按小时付费转包年包月

您可以根据需求将后付费（按小时付费）的集群转变为预付费（包年包月）的计费方式。本操作对集群的运行不会有任何影响。

> **说明：**
> 历史规格不支持直接转包年包月，请先#unique_35，然后再转包年包月。

注意事项

包年包月的集群无法转成按小时付费的集群。在您将集群转为包年包月前请务必考虑清楚，以免造成资源浪费。

前提条件

- 集群状态为**运行中**。
- 集群没有未完成的按小时付费转包年包月的订单。如果有，需要先在订单管理页面支付或作废该订单。

操作步骤

1. 登录POLARDB控制台。
2. 选择集群所在的地域。
3. 找到目标集群，在**操作**列中选择**...** > **转包年包月**。

4. 选择**购买时长**，勾选**云数据库POLARDB服务协议**，单击**去开通**，根据提示完成支付。

> **说明：**
> - 本操作会产生一个新购订单，只有完成了订单的支付，计费方式的变更才能生效。
> - 若未支付或未成功支付，您的订单管理页面将会出现未完成订单，导致您无法新购集群或再次执行转包年包月。此时需支付或作废该订单。

10.2 手动续费集群

本文介绍如何通过PolarDB控制台或续费管理控制台进行手动续费。

背景信息

您可以通过PolarDB控制台或续费管理控制台进行手动续费。通过续费管理控制台还可以为多个实例进行批量续费。

> **说明：**
> 按小时付费集群没有到期时间，不涉及续费操作。

方法一：PolarDB控制台续费

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 找到目标集群，在右侧选择**...** > **续费**。

4. 选择续费时长并勾选服务协议，单击**去支付**完成支付即可。

方法二：续费管理控制台续费

1. 登录PolarDB控制台。
2. 在控制台右上方，选择**费用** > **续费管理**。

3. 在控制台左上角单击**体验新版**，切换到新版控制台。

4. 通过搜索过滤功能在**手动续费**页签中找到目标集群，您可以单个续费或批量续费：

   - 单个续费

     a. 在目标集群右侧，单击**续费**。

> **说明：**
>
> - 示例为新版续费管理控制台操作步骤，如果您使用旧版控制台，需要在左侧导航栏中找到**云数据库PolarDB**，然后进行续费操作。

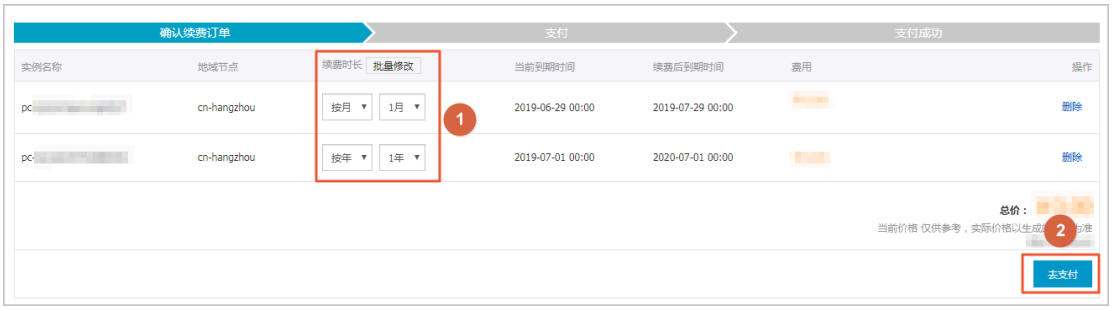
- 如果目标集群在**自动续费**或**到期不续费**页签中，您可以单击**恢复手动续费**，在弹出的对话框中单击**确定**即可恢复为手动续费。

b. 选择续费时长并勾选服务协议，单击**去支付**完成支付即可。

- 批量续费

a. 勾选目标集群，单击下方**批量续费**。

b. 选择每个集群的**续费时长**，单击**去支付**完成支付即可。



自动续费

开通自动续费可以免去您定期手动续费的烦恼，且不会因为忘记续费而导致业务中断。详情请参见#unique_37。

10.3 自动续费集群

本文为您介绍如何为集群开通自动续费，免去您手动续费的烦恼。

背景信息

包年包月集群有到期时间，如果到期未续费，会导致业务中断甚至数据丢失。开通自动续费可以免去您定期手动续费的烦恼，且不会因为忘记续费而导致业务中断。

说明：

按小时付费集群没有到期时间，不涉及续费操作。

注意事项

- 自动续费将于集群到期前9天开始扣款，支持现金及代金券扣款，请保持账户余额充足。
- 若您在自动扣款日期前进行了手动续费，则系统将在下一次到期前进行自动续费。
- 自动续费功能于次日生效。若您的集群将于次日到期，为避免业务中断，请手动进行续费，详细步骤请参见手动续费集群。

购买集群时开通续费

说明：

开通自动续费后，系统将根据您的**购买时长**为周期进行自动续费。例如，如果您购买了3个月的集群并勾选了自动续费，则每次自动续费时会缴纳3个月的费用。

在创建新集群时，可以勾选**自动续费**。

购买集群后开通自动续费

说明：

开通自动续费后，系统将根据您选择的续费周期进行自动续费。例如，如果您选择了3个月的续费周期，则每次自动续费时会缴纳3个月的费用。

1. 登录POLARDB管理控制台。
2. 在控制台右上方，选择**费用 > 续费管理**。

3. 在控制台左上角单击**体验新版**，切换到新版控制台。

4. 通过搜索过滤功能在**手动续费**或**到期不续费**页签中找到目标集群，您可以单个开通或批量开通：

   - 单个开通

     a. 单击右侧**开通自动续费**。

说明：

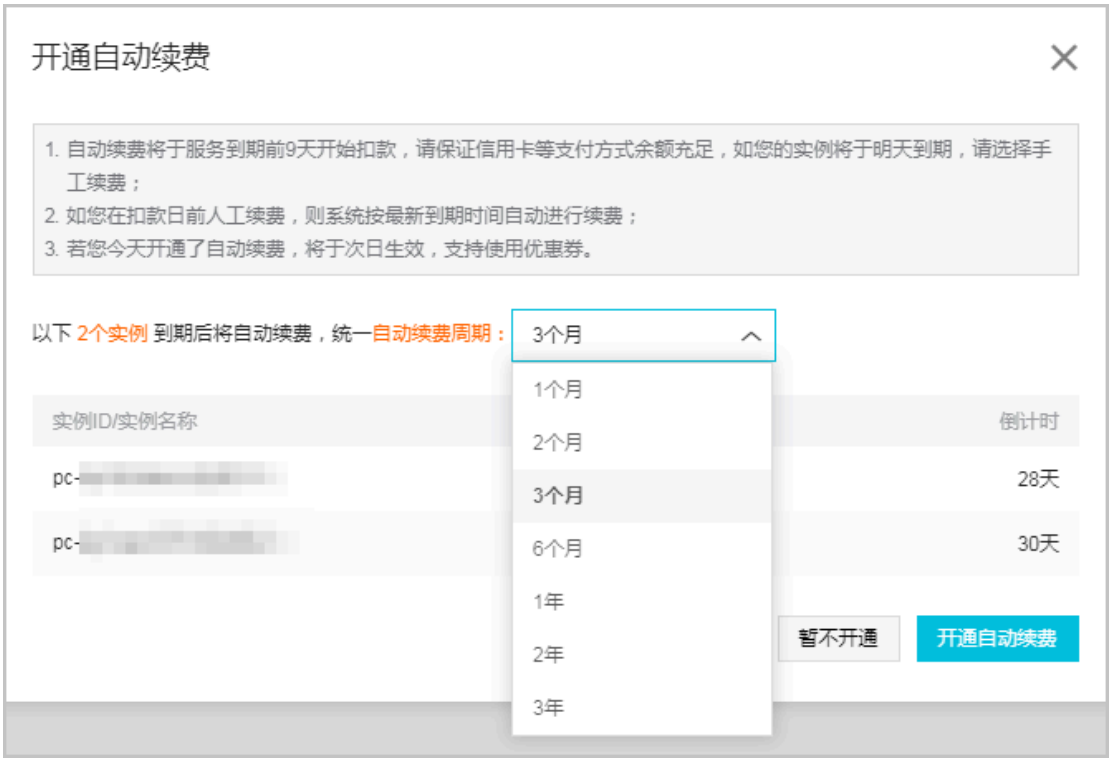
示例为新版续费管理控制台操作步骤，如果您使用旧版控制台，需要在左侧导航栏中找到**云数据库POLARDB**，然后开通自动续费。

b. 在弹出的对话框中，选择**自动续费周期**，单击**开通自动续费**。

- 批量开通

a. 勾选目标集群，单击下方**开通自动续费**。

b. 在弹出的对话框中，选择**自动续费周期**，单击**开通自动续费**。



修改自动续费周期

1. 登录POLARDB管理控制台。
2. 在控制台右上方，选择**费用 > 续费管理**。

3. 在控制台左上角单击**体验新版**，切换到新版控制台。

4. 通过搜索过滤功能在**自动续费**页签中找到目标集群，单击右侧**修改自动续费**。

> **说明：**
>
> 示例为新版续费管理控制台操作步骤，如果您使用旧版控制台，需要在左侧导航栏中找到**云数据库POLARDB**，然后修改自动续费。

5. 在弹出的对话框中，修改自动续费周期后，单击**确定**。

关闭自动续费

1. 登录POLARDB管理控制台。
2. 在控制台右上方，选择**费用 > 续费管理**。

3. 在控制台左上角单击**体验新版**，切换到新版控制台。

4. 通过搜索过滤功能在**自动续费**页签中找到目标集群，单击右侧**恢复手动续费**。

> **说明：**
> 示例为新版续费管理控制台操作步骤，如果您使用旧版控制台，需要在左侧导航栏中找到**云数据库POLARDB**，然后关闭自动续费。

5. 在弹出的对话框中，单击**确定**。

相关API

API	描述
#unique_39	创建数据库集群 。 **说明：** 创建集群时开通自动续费。
#unique_40	设置包年包月集群自动续费状态 。 **说明：** 创建集群后开通自动续费。
#unique_41	查询包年包月集群自动续费状态。

11 连接数据库集群

11.1 查看连接地址

在连接PolarDB-P集群时，您需要填写集群的连接地址。PolarDB-P提供了集群地址和主地址，本文将介绍如何在控制台查看这些连接地址。

查看方法

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 找到目标集群，单击集群ID。
4. 在**基本信息**页面的**访问信息**区域，查看连接地址。

连接地址

地址类型	地址说明	支持的网络类型
集群地址	PolarDB包含一个默认的集群地址。应用程序只需连接一个集群地址，即可连接到多个节点。带有读写分离功能，写请求会自动发往主节点，读请求会自动根据各节点的负载发往主节点或只读节点。	公网和私网
主地址	主地址总是连接到主节点，支持读和写操作。当主节点发生故障时，只读节点会接替它成为主节点，主地址也会自动切换到新的主节点。	公网和私网

私网地址和公网地址

地址类型	说明	使用场景
私网地址	• 通过私网地址访问可以发挥PolarDB的最佳性能。 • 私网地址无法被释放。	• ECS与数据库集群位于同一VPC，那么ECS可以通过私网地址访问数据库集群。 • 使用DMS通过VPC访问数据库集群。
公网地址	• 需手动申请公网的连接地址，也可以释放公网的连接地址。 • 公网即因特网。通过公网访问将无法实现PolarDB最佳性能。	通过公网访问数据库集群进行维护操作。

下一步

连接数据库集群

相关API

API	描述
#unique_23	查询集群的地址信息。
#unique_22	创建集群的公网地址。
#unique_25	修改集群默认访问地址。
#unique_26	释放集群地址。

11.2 创建自定义集群地址

您可以在PolarDB-P集群上新增自定义集群地址，通过设置集群地址的读写模式、一致性级别及关联的只读节点等，来满足不同的业务场景，增强业务的灵活性。 本文将介绍如何为PolarDB-P新增自定义集群地址。

操作步骤

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 找到目标集群，单击集群ID。
4. 在**链接地址**区域，单击**创建自定义地址**。

5. 在**创建自定义地址**对话框内，设置如下参数。

参数	说明
读写模式	新建自定义集群地址的读写模式，可选模式为**只读**和**可读可写（自动读写分离）**。  **说明:** 创建自定义地址后还可以修改读写模式。修改读写模式后，已有的连接会断开，请确保应用程序有自动重连机制。
读负载节点	在左侧选择想要加入本地址用于处理读请求的节点，可选节点包括主节点和所有只读节点。  **说明:** • 读写模式为**可读可写（自动读写分离）**时，至少要选择2个节点且必须包含主节点。 • 读写模式为**只读**时，支持创建单节点地址，详情请参见常见问题。
新节点自动加入	新增的节点是否要自动添加到该地址中。
负载均衡策略	读写分离时，在多个节点间用于处理读请求的调度策略，默认为**基于负载的自动调度**，且不可更改。
一致性级别	• 读写模式为**可读可写（自动读写分离）**时，可选一致性级别有**最终一致性**和**会话一致性（推荐）**，详情请参见PolarDB PostgreSQL一致性级别。 • 读写模式为**只读**时，默认一致性级别为**最终一致性**且不可更改。
主库不接受读	开启之后，查询SQL将仅发送到只读节点，来降低主节点的负载，确保主节点稳定。  **说明:** 仅读写模式为**可读可写（自动读写分离）**时支持该配置。
事务拆分	开启或关闭事务拆分，详情请参见高级选项-事务拆分。  **说明:** 仅读写模式为**可读可写（自动读写分离）**时支持该配置。

6. 单击**确定**。

常见问题

- Q：如何创建单个节点的独立地址？

  A：仅当集群地址读写模式为**只读**且集群内拥有三个及以上节点时，才支持创建单节点地址。单节点地址的读负载节点只能是只读节点，详细操作步骤请参见操作步骤。

  警告：

  创建单节点地址后，当此节点故障时，该地址可能会有最多1小时的不可用，请勿用于生产环境。

- Q：一个集群最多允许创建多少个单节点地址？

  A：如果您的集群内有3个节点，则只允许为其中1个只读节点创建单节点地址；若集群内有4个节点，则允许为其中2个只读节点创建各自的单节点地址，以此类推。

- Q：当出现系统故障需要进行主备切换时，单节点地址下对应的只读节点能否切换为新主节点？

  A：主备切换时，单节点地址下对应的只读节点不会被自动切换为新主节点。但您仍可以通过手动切换将其设置为新主节点，详细操作请参见主备切换。

- Q：一个集群最多可拥有多少个集群地址？

  A：一个集群最多可拥有4个集群地址，其中1个为默认地址，另外3个为自定义地址。

- Q：可以修改集群地址吗？

  A：默认集群地址和自定义地址都支持修改配置，详情请参见修改集群地址。

- Q：可以释放集群地址吗？

  A：仅自定义集群地址可以被释放，默认集群地址无法被释放，详情请参见释放自定义集群地址。

相关API

API	描述
#unique_45	创建自定义集群地址。
#unique_23	查询集群地址。

11.3 连接数据库集群

本文介绍如何通过DMS和客户端连接到POLARDB for PostgreSQL集群。

前提条件

已创建数据库集群的高权限账号或普通账号。具体操作请参见创建数据库账号。

使用DMS登录POLARDB

数据管理（Data Management Service，简称DMS）是一种集数据管理、结构管理、访问安全、BI图表、数据趋势、数据轨迹、性能与优化和服务器管理于一体的数据管理服务。支持对关系型数据库（MySQL、SQL Server、PostgreSQL等）和NoSQL数据库（MongoDB、Redis等）的管理，同时还支持Linux服务器管理。

1. 登录POLARDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 单击目标集群ID，进入基本信息页面。
4. 单击页面右上角的**登录数据库**。

5. 在数据库登录页面，输入主地址和端口号（以英文冒号隔开），以及高权限账号或普通账号的用户名和密码，然后单击**登录**。

说明：

DMS登录仅支持主地址，不支持集群地址。关于如何查看连接地址，请参见#unique_15。

通过客户端连接

由于POLARDB for PostgreSQL暂不支持设置集群白名单，只有相同VPC内的实例才可以访问集群，所以客户端所在主机和POLARDB for PostgreSQL集群需要在同一VPC内。

1. 启动pgAdmin 4客户端。
2. 右击**Servers**，选择**创建 > 服务器** ，如下图所示。

3. 在**创建-服务器**页面的**通常**标签页面中，输入服务器名称，如下图所示。

参数说明如下：

- **主机名称/地址**：输入POLARDB集群的内网地址。查看POLARDB集群的地址及端口信息的步骤如下：

  a. 进入POLARDB控制台。

  b. 找到目标集群，单击集群的ID。

  c. 在**访问信息**区域查看地址及端口信息。

- **端口**：需输入POLARDB集群的内网端口。
- **用户名**：POLARDB集群的高权限账号名称。
- **密码**：POLARDB集群的高权限账号所对应的密码。

4. 选择**Connection**标签页，输入要连接的实例信息，如下图所示。

5. 单击**保存**。

6. 若连接信息无误，选择**Servers > 服务器名称 > 数据库 > postgres**，会出现如下界面，则表示连接成功。

说明：

postgres是POLARDB for PostgreSQL集群默认的系统数据库，请勿在这个数据库中进行任何操作。

11.4 释放自定义集群地址

本文介绍如何释放PolarDB-P集群的自定义集群地址。

注意事项

- 仅自定义集群地址支持释放，默认集群地址无法释放。
- 自定义集群地址释放后无法恢复，请及时修改客户端的连接地址。

操作步骤

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 找到目标集群，单击集群ID。

4. 在**链接地址**区域，找到目标集群，单击目标集群名称右侧的图标后，再单击**释放**。

5. 在弹出的对话框中，单击**确定**。

相关API

API	描述
#unique_23	查询集群地址。
#unique_46	释放自定义集群地址。

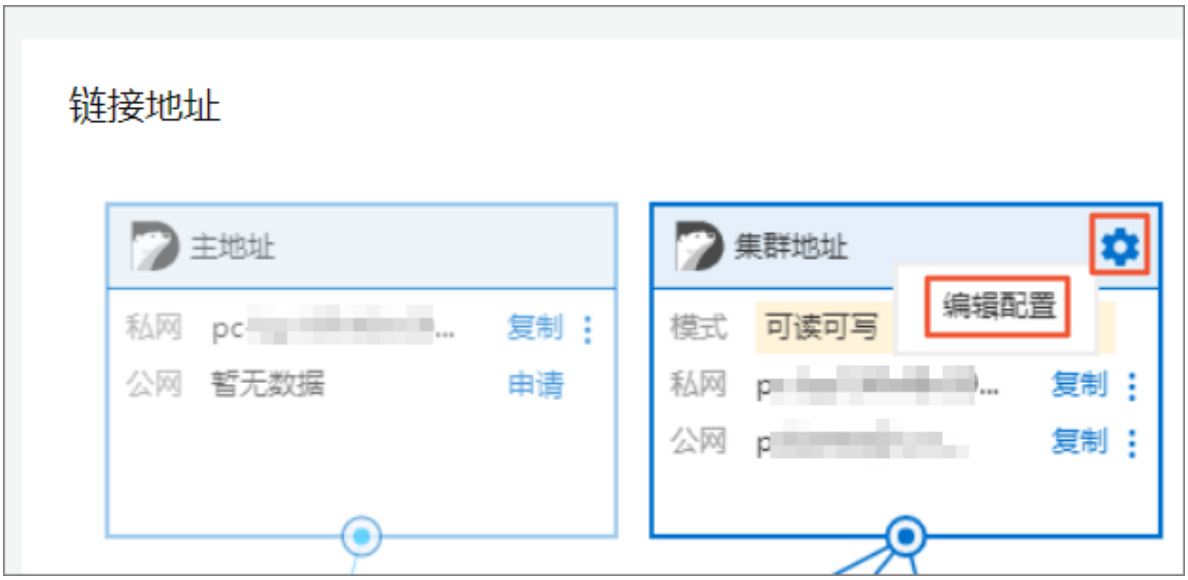
11.5 修改集群地址

您可以在PolarDB-P集群上修改默认集群地址或修改自定义集群地址，通过设置集群地址的读写模式、一致性级别及关联的只读节点等，来满足不同的业务场景，增强业务的灵活性。

操作步骤

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 找到目标集群，单击集群ID。

4. 在**链接地址**区域，找到目标集群地址，单击目标集群名称右侧的 > **编辑配置**。



5. 在**编辑地址配置**对话框内，设置如下参数。

参数	说明
读写模式	新建自定义集群地址的读写模式，可选模式为**只读**和**可读可写（自动读写分离）**。 **说明:** 修改读写模式后，已有的连接会断开，请确保应用程序有自动重连机制。
读负载节点	在左侧选择想要加入本地址用于处理读请求的节点，可选节点包括主节点和所有只读节点。 **说明:** • 读写模式为**可读可写（自动读写分离）**时，至少要选择2个节点且必须包含主节点。 • 读写模式为**只读**时，支持创建单节点地址。若创建单节点地址，当此节点故障时，该地址可能会有最多1小时的不可用，请勿用于生产环境。因此推荐至少选择2个节点，提升可用性。
新节点自动加入	新增的节点是否要自动添加到该地址中。
负载均衡策略	读写分离时，在多个节点间用于处理读请求的调度策略，默认为**基于负载的自动调度**，且不可更改。

参数	说明
一致性级别	• 读写模式为**可读可写（自动读写分离）**时，可选一致性级别有**最终一致性**和**会话一致性（推荐）**，详情请参见PolarDB PostgreSQL一致性级别。 • 读写模式为**只读**时，默认一致性级别为**最终一致性**且不可更改。
主库不接受读	开启之后，查询SQL将仅发送到只读节点，来降低主节点的负载，确保主节点稳定。 **说明:** 仅读写模式为**可读可写（自动读写分离）**时支持该配置。
事务拆分	开启或关闭事务拆分，详情请参见高级选项-事务拆分。 **说明:** 仅读写模式为**可读可写（自动读写分离）**时支持该配置。

6. 单击**确定**。

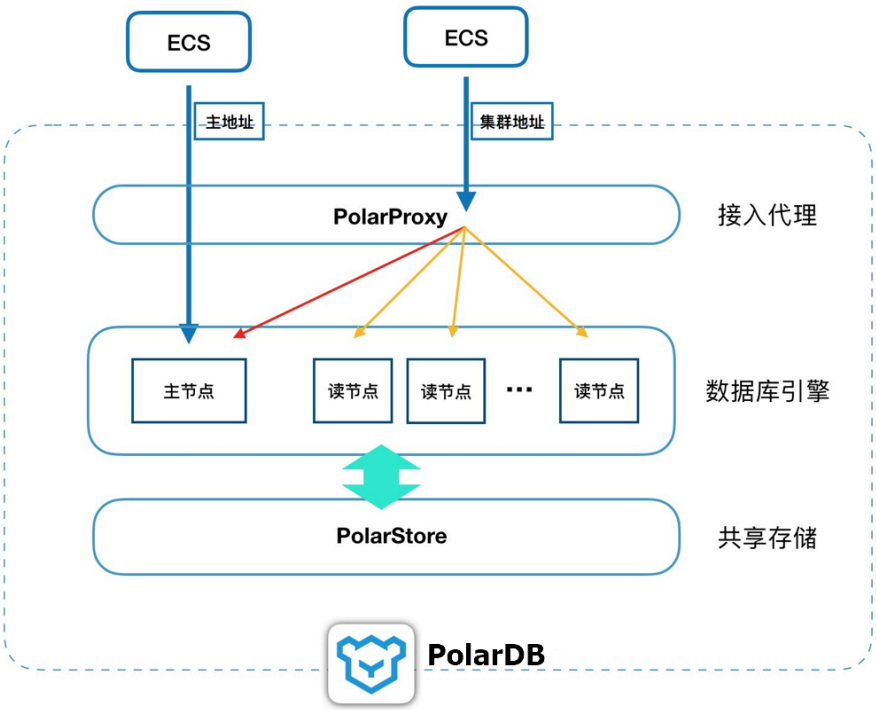
相关API

API	描述
#unique_23	查询集群地址。
#unique_24	修改集群地址。

11.6 PolarDB PostgreSQL一致性级别

PolarDB架构

PolarDB是一个由多个节点构成的数据库集群，一个主节点，多个读节点。对外默认提供两个地址，一个是集群地址，一个是主地址，推荐使用集群地址，因为它具备读写分离功能可以把所有节点的资源整合到一起对外提供服务。



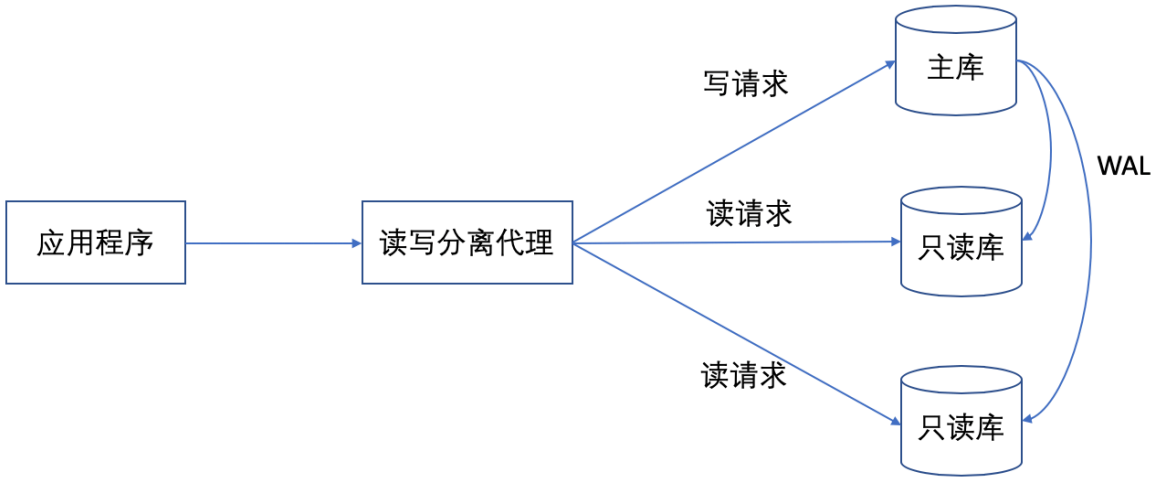
PostgreSQL读写分离解决和引入

PostgreSQL具有主从复制简单易用的特点，通过把主库的WAL异步地传输到备库并实时应用，一方面可以实现高可用，另一方面备库也可以提供查询，来减轻对主库的压力。

虽然备库可以提供查询，但存在两个问题：

- 问题1：主库和备库一般提供两个不同的访问地址，应用程序端需要选择使用哪一个，对应用有侵入。
- 问题2：PostgreSQL的复制是异步的。客户端提交commit并且成功之后，数据可能还没有同步到只读节点。因此备库的数据并不是最新的，无法保证查询的一致性。

为了解决问题1，PolarDB引入了读写分离代理。代理会伪造成PostgreSQL与应用程序建立连接，解析发送进来的每一条SQL，如果是UPDATE、DELETE、INSERT、CREATE等写操作则直接发往主节点，如果是SELECT则发送到只读节点。



但是问题2，延迟导致的查询不一致还是没有解决，使用时就不可避免遇到备库SELECT查询数据不一致的现象（因为主备有延迟）。PostgreSQL负载低的时候延迟可以控制在5秒内，但当负载很高时，尤其是对大表做DDL（比如加字段）或者大批量插入的时候，延迟会非常严重。

PolarDB最终一致性和会话一致性

- **最终一致性**：PolarDB采用异步物理复制方式在主库和只读库间做数据同步， 在主库更新后，相关的更新会apply到只读库，具体的延迟时间与写入压力有关， 一般在ms级别， 通过异步复制的方式实现主库和只读库之间的最终数据一致。
- **会话一致性**：为了解决最终一致性会出现的查询不一致，PolarDB利用自身物理复制速度快的优点，将查询发给已经更新了数据的只读节点，详细原理请参见实现原理。

PolarDB读写分离的会话一致性

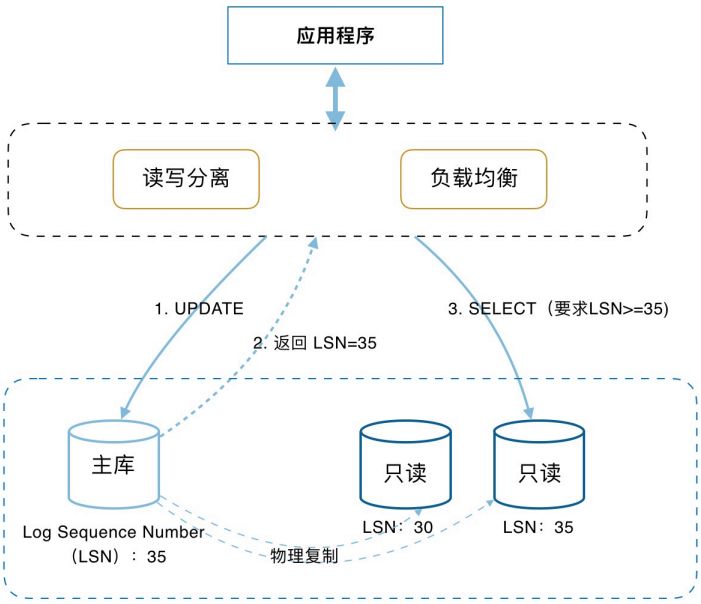
PolarDB是读写分离的架构，传统的读写分离都只提供最终一致性的保证，主从复制延迟会导致从不同节点查询到的结果不同，比如一个会话内连续执行以下QUERY：

```
INSERT INTO t1(id, price) VALUES(111, 96);
UPDATE t1 SET price = 100 WHERE id=111;
SELECT price FROM t1;
```

在读写分离的下，最后一个查询的结果是不确定的，因为读会发到只读库，在执行SELECT时之前的更新是否同步到了只读库时不确定的，因此结果也是不确定的。因为存在查询结果不确定的问题，所以就要求应用程序去适应最终一致性，而一般的解决方法是： 将业务做拆分，有高一致性要求的请求直连到主库，可以接受最终一致性的部分走读写分离。这样会增加应用开发的负担，还会增大主库的压力，影响读写分离的效果。

为了解决这个问题，PolarDB提供了会话一致性或者说因果一致性的保证，会话一致性即保证同一个会话内，后面的请求一定能够看到此前更新所产生版本的数据或者比这个版本更新的数据，保证单调性，就很好的解决了上面这个例子里的问题。

实现原理



PolarDB链路的中间层做读写分离的同时，中间层会跟踪各个节点已经apply了的redolog位点并产生LSN，同时每次更新时会记录此次更新的位点为Session LSN，当有新请求到来时PolarDB会比较Session LSN 和当前各个节点的LSN，仅将请求发往LSN >= Session LSN的节点，从而保证了会话一致性；表面上看该方案可能导致主库压力大，但是因为PolarDB是物理复制，速度极快，在上述场景中，当更新完成后，返回客户端结果时复制就同步在进行，而当下一个读请求到来时主从极有可能已经完成，然后大多数应用场景都是读多写少，所以经验证在该机制下即保证了会话一致性，也保证了读写分离负载均衡的效果。

一致性级别选择最佳实践

PolarDB会话一致性，该级别对性能影响很小而且能满足绝大多数应用场景的需求。

如果您有有不同会话间一致性需求的可以选择以下方案：

使用Hint将特定查询强制发到主库。

```
eg: /*FORCE_MASTER*/ select * from user;
```

12 集群

12.1 创建PolarDB PostgreSQL数据库集群

本文介绍如何通过PolarDB管理控制台创建PolarDB PostgreSQL数据库集群。

前提条件

已注册并登录阿里云账号，详细操作步骤请参见注册和登录阿里云账号。

背景信息

一个集群包含一个主节点以及最多15个只读节点，最少一个只读节点，用于提供双主（Active-Active）高可用。节点是虚拟化的数据库服务器，节点中可以创建和管理多个数据库。

说明：

- PolarDB支持专有网络VPC（Virtual Private Cloud）和经典网络。VPC是阿里云上一种隔离的网络环境，安全性比传统的经典网络更高。
- PolarDB与其他阿里云产品通过内网互通时才能发挥PolarDB的最佳性能，因此，建议将PolarDB与云服务器ECS配合使用，且与ECS创建于同一个VPC，否则PolarDB无法发挥最佳性能。如果您ECS的网络类型为经典网络，需将ECS从经典网络迁移到VPC，具体请参见ECS实例迁移。

优惠活动

首购折扣价：首次购买PolarDB享受折扣价。详情请参见优惠活动。

操作步骤

1. 登录PolarDB控制台。
2. 单击页面左上角。
3. 选择**包年包月**或**按量付费**。

说明：

- **包年包月**：在创建集群时支付计算节点（一个主节点和一个只读节点）的费用，而存储空间会根据实际数据量按小时计费，并从账户中按小时扣除。如果您要长期使用该集群，**包年包月**方式更为经济，而且购买时长越长，折扣越多。

- **按量付费（按小时付费）**：无需预先支付费用，计算节点和存储空间（根据实际数据量）均按小时计费，并从账户中按小时扣除。如果您只需短期使用该集群，可以选择**按量付费**，用完即可释放，节省费用。

4. 设置如下参数。

控制台区域	参数	说明
基本配置	地域	集群所在的地理位置。购买后无法更换地域。  **说明:** 请确保PolarDB与需要连接的ECS创建于同一个地域，否则它们无法通过内网互通，只能通过外网互通，无法发挥最佳性能。

控制台区域	参数	说明
	创建方式	创建PolarDB集群的方式。 • **创建主集群**：创建一个全新的PolarDB。 • **创建只读集群**：在一个全球数据库网络（GDN）中创建只读集群。 • **从RDS迁移**：先从RDS克隆数据，然后保持增量同步，主要用于迁移。在正式迁移切换前PolarDB为只读，且默认开启Binlog。详情请参见#unique_50。 - **源RDS引擎**：源RDS实例的引擎类型，不可变更。 - **源RDS版本**：源RDS实例的版本，不可变更。 - **源RDS实例**：可选的源RDS实例，不包括只读实例。 **说明:** 可选择的源实例列表，不包含只读实例。暂不支持已开通 TDE/SSL 功能以及包含非 InnoDB 引擎的 RDS 实例。从备份文件恢复到 PolarDB，不影响源实例的正常运行。 • **从RDS克隆**：基于所选的RDS实例，快速克隆一个数据完全一样的PolarDB集群。详情请参见#unique_51。 - **源RDS引擎**：源RDS实例的引擎类型，不可变更。 - **源RDS版本**：源RDS实例的版本，不可变更。 - **源RDS实例**：可选的源RDS实例，不包括只读实例。 **说明:** 可选择的源实例列表，不包含只读实例。暂不支持已开通 TDE/SSL 功能以及包含非 InnoDB 引擎的 RDS 实例。从备份文件恢复到 PolarDB，不影响源实例的正常运行。 • **从回收站恢复**：您可以通过从回收站中恢复已删除集群的备份来创建新集群。 - **原版本**：已删除集群的版本。 - **已删除集群**：已删除的集群名称。 - **历史备份**：选择想要恢复的备份。
	主可用区	集群的主可用区。 • 可用区是地域中的一个独立物理区域，不同可用区之间没有实质性区别。 • 您可以选择将PolarDB与ECS创建在同一可用区或不同的可用区。 • 您只需要选择主可用区，系统会自动选择备可用区。

控制台区域	参数	说明
	网络类型	固定为VPC专有网络，无需选择。如需使用经典网络请先任选一个VPC专有网络，待创建完成后再进行配置，详情请参见#unique_52/unique_52_Connect_42_section_xxt_kpv_tdb。
	VPC网络 VPC交换机	请确保PolarDB与需要连接的ECS创建于同一个VPC，否则它们无法通过内网互通，无法发挥最佳性能。 • 如果您已创建符合您网络规划的VPC，直接选择该VPC。例如，如果您已创建ECS，且该ECS所在的VPC符合您的规划，那么选择该VPC。 • 如果您未创建符合您网络规划的VPC，您可以使用默认VPC和交换机： - 默认VPC： ■ 在您选择的地域中是唯一的。 ■ 网段掩码是16位，如172.31.0.0/16，最多可提供65536个私网IP地址。 ■ 不占用阿里云为您分配的VPC配额。 - 默认交换机： ■ 在您选择的可用区中是唯一的。 ■ 网段掩码是20位，如172.16.0.0/20，最多可提供4096个私网IP地址。 ■ 不占用VPC中可创建交换机的配额。 • 如果以上默认VPC和交换机无法满足您的要求，您可以自行创建VPC和交换机。
资源组	资源组	您可以选择**全部**或**自营**里的资源组。
实例配置	兼容性	• 兼容MySQL 8.0（完全兼容）。原生支持并行查询，特定场景下（TPC-H测试）性能提升十倍，详情请参见#unique_53。 • 兼容MySQL 5.6（完全兼容）。 • 兼容PostgreSQL 11（完全兼容）。 • 兼容Oracle（高度兼容）。
	系列	默认为**普通版**。
	节点规格	按需选择。所有PolarDB节点均为独享型，性能稳定可靠。关于各规格的具体信息，请参见#unique_28。
	节点个数	• 无需选择。系统将自动创建一个与主节点规格相同的只读节点。 • 如果主节点故障，系统会自动将只读节点切换为新的主节点，并重新生成一个只读节点。 • 关于只读节点的更多信息，请参见#unique_55。

控制台区域	参数	说明
	存储费用	无需选择。系统会根据实际数据使用量按小时计费，详情请参见产品价格。 **说明：** 创建集群时无需选择存储容量，存储容量随数据量的增减而自动弹性伸缩。
	开启TDE	选择是否开启TDE加密。启用TDE加密后，PolarDB将对集群数据文件进行加密，对于业务访问透明，会有5%~10%的性能损失。 **说明：** • 仅PolarDB PostgreSQL和PolarDB兼容Oracle数据库支持开启TDE加密。 • TDE功能开启后不可关闭。

5. 设置**购买时长**（仅针对预付费集群）和**集群数量**后，单击右侧的**立即购买**。

   说明：
   最多可以一次性创建50个集群，适用于游戏批量开服等业务场景。

6. 在**确认订单**页面，确认订单信息，阅读和勾选服务协议，单击**去支付**。

7. 支付成功后，需要10~15分钟创建集群，之后您就可以在集群列表中看到新创建的集群。

   说明：
   - 当集群中的节点状态为时，整个集群可能仍未创建完成，此时集群不可用。只有当集群状态为时，集群才可以正常使用。
   - 请确认已选中正确的地域，否则无法看到您创建的集群。
   - 当您的数据量较大时，推荐您购买PolarDB存储包，相比按小时付费，预付费购买存储包有折扣，购买的容量越大，折扣力度就越大。

下一步

#unique_56

相关API

API	描述
#unique_57	创建数据库集群。
#unique_58	查看集群列表。

API	描述
#unique_59	查看指定PolarDB集群的详细属性。
#unique_60	查询PolarDB包年包月集群自动续费状态。
#unique_61	设置PolarDB包年包月集群自动续费状态。

12.2 临时升配

PolarDB的包年包月集群支持临时升配，可以帮助您轻松应对短时间的业务高峰期。

前提条件

- 集群为包年包月集群。
- 集群没有尚未生效的续费变配订单。
- 集群没有尚未生效的临时升配订单。

背景介绍

临时升配是指临时升级规格，提升整体性能。到达指定的还原时间后，集群的规格会自动还原到临时升配前的状态。

说明：
不支持临时降级，如需降级请参见#unique_63。

注意事项

- 还原过程可能会出现闪断，请确保应用程序具备重连机制。
- 还原时间不能晚于集群到期时间的前1天。例如集群1月10日到期，则临时升配的还原时间最多为1月9日。
- 临时升配的最短时间为1小时，由于设置还原时间后无法修改，建议升配时间最长不超过14天。
- 临时升配期间不支持普通的#unique_63。
- 临时升配后如果性能不够，在还原时间到达之前最多可以再进行1次升配，此次设置的**还原时间**不能早于第1次。

计费

临时升配的价格是新老配置差价的1.5倍。计算公式如下：

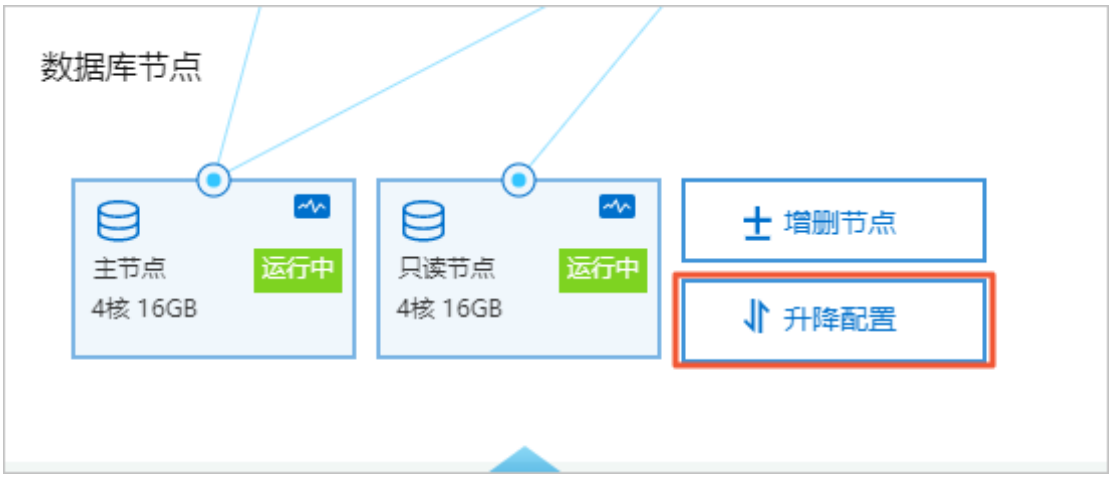
临时升配N天，费用 =（新规格包月价格 - 老规格包月价格）/30 x 1.5 x N。

操作步骤

1. 登录PolarDB控制台。

2. 在控制台左上角，选择集群所在地域。
3. 在**集群列表**页，找到目标集群。
4. 您可以选择如下两种方式中的任意一种进入**升降配向导（包年包月）**页面：
   - 单击目标集群**操作**栏中的**升降配**。

   - **a.** 单击目标集群ID，进入目标集群**基本信息**页。
     b. 在**数据库节点**区域，单击**升降配置**。



5. 在**升降配向导（包年包月）**页面，勾选**临时升级配置**，单击**确定**。

> **说明：**
> 仅包年包月集群支持**临时升级配置**。

6. 在弹出的对话框中，设置如下参数。

参数	说明
节点	为当前节点选择升级后的目标节点规格。
还原时间	选择短时升配的到期还原时间。 **说明：** • 临时升配后如果性能不够，在还原时间到达之前最多可以再进行1次升配，此次设置的**还原时间**不能早于第1次。 • 临时升配的最短时间为1小时，由于设置还原时间后无法修改，建议升配时间为14天以内。 • 还原时间不能晚于集群到期时间的前一天。

7. 勾选服务协议，单击**去支付**完成支付。
8. 在**支付**页面，确认待支付订单，单击**订购**。

12.3 使用存储包

为了更好地帮助您降低存储成本，PolarDB推出了预付费形式的存储包。

背景信息

PolarDB的存储空间无需手动配置，根据数据量自动伸缩，您只需为实际使用的数据量按量付费。当您的数据量较大时，推荐您使用PolarDB存储包，相比按量付费，预付费购买存储包更加优惠，购买的容量越大，优惠力度就越大。

存储空间费用

PolarDB存储空间费用请参见#unique_28/unique_28_Connect_42_section_u9d_n9d_3jt。

存储包和按量付费的价格对比

下表为按月计费存储包和按量付费的价格对比。 使用容量越大的存储包，价格越优惠。

容量（GB）	中国内地		中国香港及海外	
	按量付费（元/月）	存储包（元/月）	按量付费（元/月）	存储包（元/月）
50	175	175	195	195
100	350	350	390	390
200	700	700	780	780
300	1,050	1,050	1,170	1,170
500	1,750	1,750	1,950	1,950
1,000	3,500	3,150	3,900	3,510
2,000	7,000	6,300	7,800	7,020
3,000	10,500	7,800	11,700	8,600
5,000	17,500	13,000	19,500	14,400
10,000	35,000	21,000	39,000	23,400
20,000	70,000	42,000	78,000	46,800
30,000	105,000	63,000	117,000	70,200
50,000	175,000	96,000	195,000	106,900
100,000	350,000	192,000	390,000	213,900

注意事项

- 每种类型资源包只允许购买一个，存储包容量不够时，可以升级存储包。
- 存储包暂不支持降级。
- 存储包由**资源包类型**规定的地域内的所有集群共享使用。
- 超出存储包容量的部分以按量付费的方式收取费用。例如您有三个存储容量均为400GB的PolarDB集群，这三个集群可以共享1个1000GB的存储包，超出的200GB则按量付费，您可以在**费用中心**查看存储包抵扣量。

说明：

更多关于存储包的说明，请参见常见问题。

购买存储包

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 单击页面左上角。
4. 单击**存储包**页签，设置如下参数。

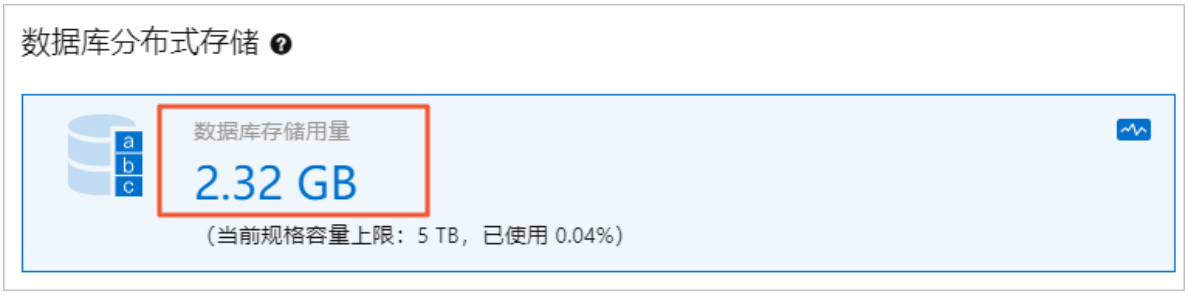
参数	说明
资源包类型	• **中国内地通用**：购买后可用于中国内地地域的PolarDB集群。 • **中国香港及海外通用**：购买后可用于中国香港及海外地域的PolarDB集群。
存储包规格	存储包的容量大小。
购买时长	购买存储包的时长。

5. 单击**立即购买**。
6. 勾选服务协议，单击**去支付**完成支付。

查看数据库存储用量

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 找到目标集群，单击集群ID。

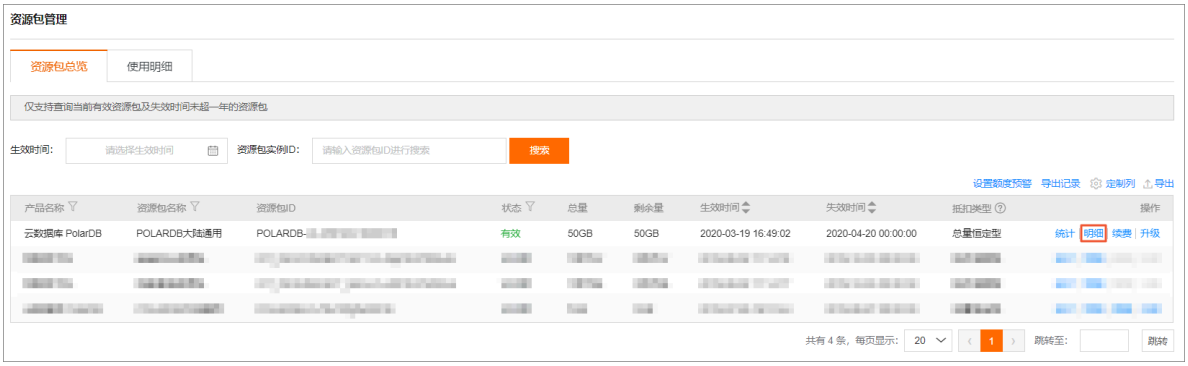
4. 在页面的**数据库分布式存储**区域，查看**数据库存储用量**。



查看存储包抵扣量

> **说明：**
> 仅支持查看当前有效资源包及失效时间未超一年的资源包。

1. 登录费用中心。
2. 在左侧导航栏中，单击**资源管理 > 资源包**。
3. 在**资源包管理**页面的**资源包总览**页签，找到目标资源包，单击**操作**栏中的**明细**。

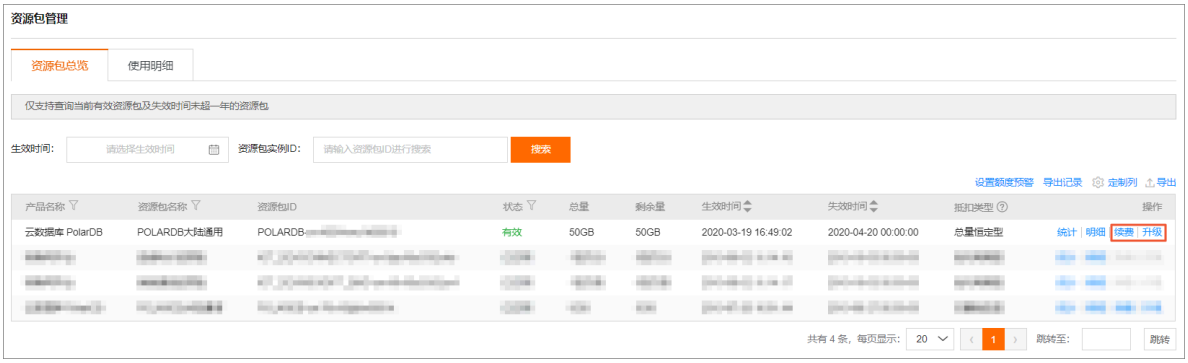


4. 在**使用明细**页签，查看资源包使用情况。

续费或升级存储包

1. 登录费用中心。
2. 在左侧导航栏中，单击**资源管理 > 资源包**。

3. 在**资源包管理**页面的**资源包总览**页签，找到目标资源包，单击**操作**栏中的**续费**或**升级**。



4. 您可以按以下方法进行续费或升级操作：

   - **续费**

     a. 单击**操作**列的**续费**。

     b. 选择**续费时长**，勾选服务协议，单击**去支付**完成支付。

   - **升级**

     a. 单击**操作**列的**升级**。

     b. 选择**存储包规格**，勾选服务协议，单击**去支付**完成支付。

常见问题

- 存储包是否跟集群绑定售卖？

  答：不绑定。您需要单独购买存储包，购买后会自动抵扣相应地域内的集群存储空间。

- 存储包是否可以被多个集群共享？

  答：可以。存储包由**资源包类型**（中国内地或中国香港及海外）规定的地域内的所有集群共享使用。

- 存储包是否可以被不同引擎的集群共享？

  答：可以，存储包可以同时用于PolarDB MySQL/PolarDB-P/PolarDB-O集群。

- 存储包容量不够用或用完了怎么办？可以再买一个吗？

  答：每种类型的存储包只能购买一个，您可以通过升级存储包的方式解决，具体请参见续费或升级存储包。

- 当前数据量超出存储包容量的部分如何计费？

  答：超出存储包容量的部分以按量付费的方式收取费用，具体请参见#unique_28/unique_28_Connect_42_section_u9d_n9d_3jt。

- 存储包支持3TB和5TB，业务只需要4TB，如何选购？

  答：您可以先购买3TB，存储量接近5TB时再升级为5TB。

- 存储包要到期了怎么续费？

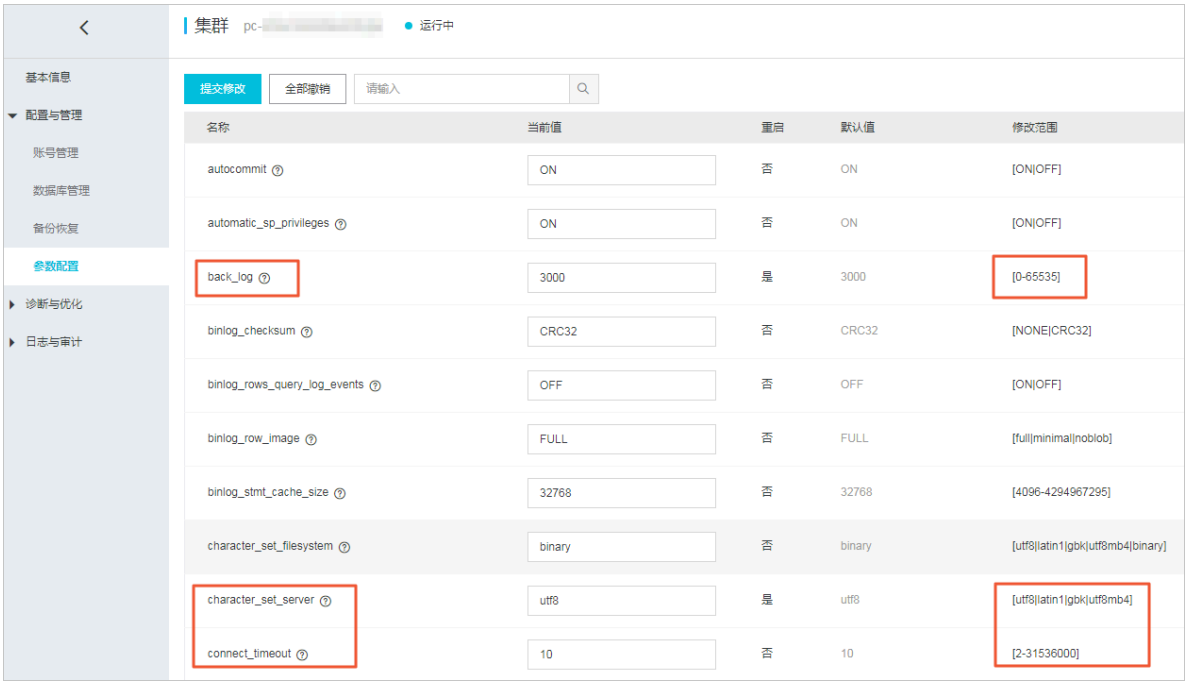
  答：您可以在用户中心对即将到期的存储包进行续费，具体请参见续费或升级存储包。

12.4 设置集群参数

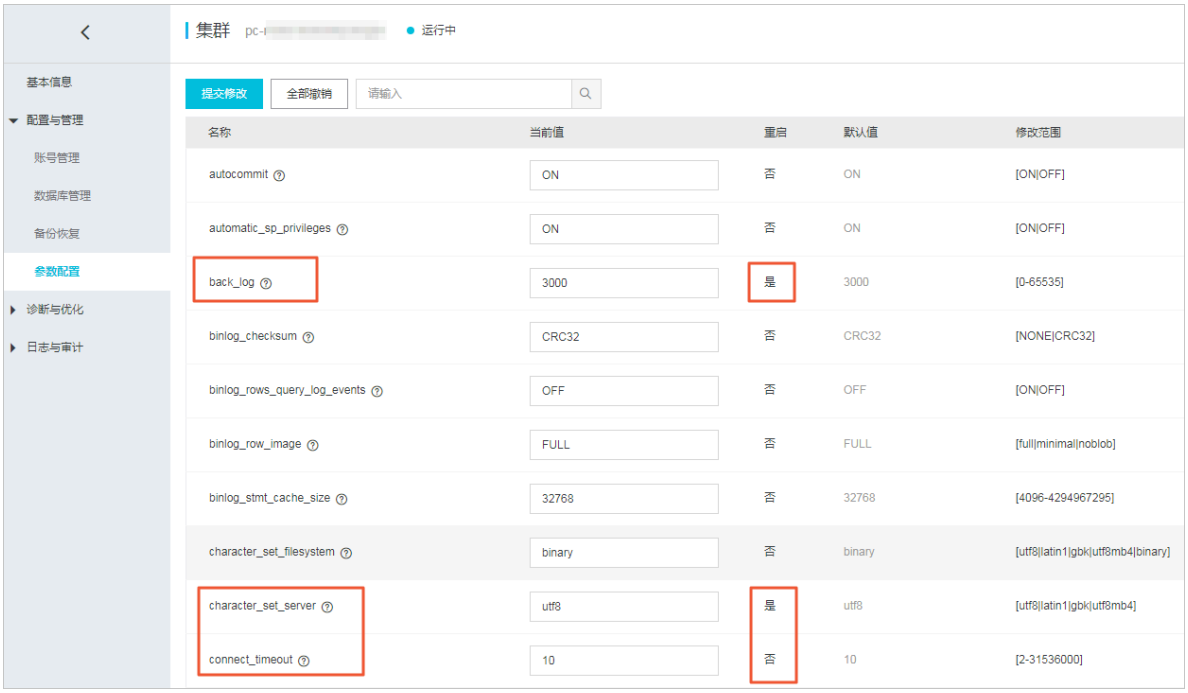
本文将介绍如何通过POLARDB控制台修改集群参数。

注意事项

- 请按照控制台上规定的**修改范围**修改参数值。

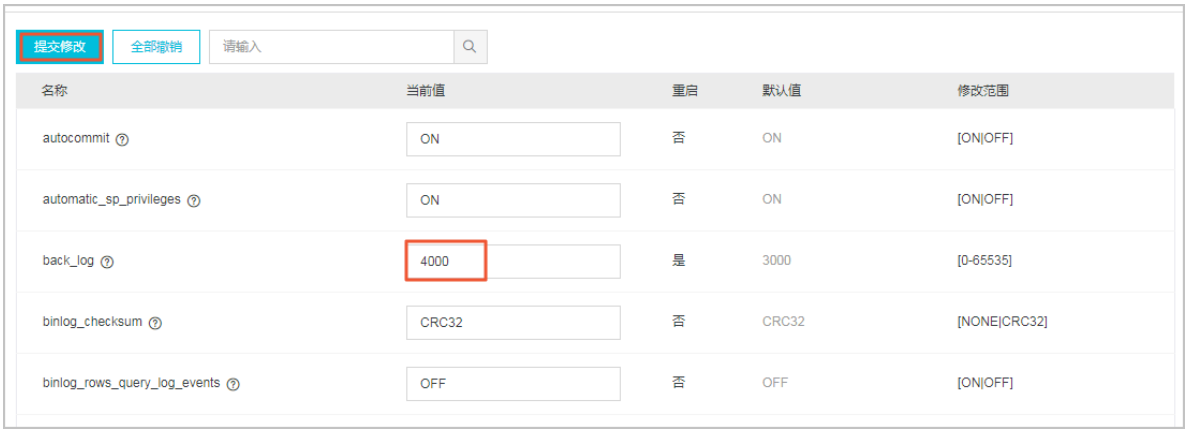


- 部分参数修改后需要重启全部节点，重启前请做好业务安排，谨慎操作。详情请参见**参数设置**页面中的**重启**列。



操作步骤

1. 登录POLARDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，选择**配置与管理 > 参数配置**。
5. 修改一个或多个参数的**当前值**，单击**提交修改**。



6. 在弹出的**保存改动**对话框中，单击**确定**。

相关API

12.5 变更配置

您可以根据业务需求变更集群的配置。

前提条件

集群没有正在进行的配置变更时，才可以变更集群规格。

背景信息

POLARDB支持三维扩展能力：

- 计算能力纵向扩展：集群规格升降配。本文介绍详细信息。
- 计算能力横向扩展：增加或减少只读节点。具体操作说明，请参见增加或删除节点。
- 存储空间横向扩展：POLARDB采用Serverless架构，无需手动设置容量或扩缩容，容量随用户数据量的变化而自动在线调整。

本文介绍如何升级或降级集群的规格，新规格会立即开始生效（每个节点需要5到10分钟）。

变更配置的费用说明

详情请参见#unique_67。

注意事项

- 您只能对整个集群进行规格升降级，无法对集群中的单个节点进行规格升降级。
- 集群规格的升降级不会对集群中已有数据造成任何影响。
- 在集群规格变更期间， POLARDB服务会出现几秒钟的闪断且部分操作不能执行的状况，建议您在业务低谷期执行变更。闪断后需在应用端重新连接。

操作步骤

1. 登录POLARDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 进入**升降配向导**页面。您可以按照如下两种方式操作：
   - 找到目标集群，在**操作**列单击**升降配**。

   - 找到目标集群，单击集群ID，在基本信息页面下方单击**升降配**。

4. 勾选**升级配置**或**降级配置**，单击**确定**。

> **说明：**
> 仅包年包月集群支持**临时升级配置**，详情请参见临时升配。

5. 选择所需的规格。

> **说明：**
> 同一集群中，所有节点的规格总是保持一致。

6. 勾选服务协议，单击**去支付**并完成支付。

> **说明：**
> 规格变更预计需要10分钟生效。

12.6 增加或删除节点

创建POLARDB集群后，您可以手动增加或删除只读节点。集群最多包含15个只读节点，最少一个只读节点（用于保障集群的高可用）。同一集群中，所有节点的规格总是保持一致。

节点费用

增加节点时的计费方式如下：

- 如果集群为包年包月（预付费），则增加的节点也是包年包月。
- 如果集群为按小时付费（后付费），则增加的节点也是按小时付费。

说明：

- 包年包月和按小时付费的只读节点都可以随时释放，释放后会退款或停止计费。
- 增加节点仅收取节点规格的费用（详情请参见#unique_28），存储费用仍然按实际使用量收费，与节点数量无关。

注意事项

- 仅当集群没有正在进行的配置变更时，才可以增加或删除只读节点。
- 为避免操作失误，每次操作只能增加或删除一个只读节点，增加或删除多个只读节点请多次操作。
- 增加或删除节点需要5分钟左右生效。

增加只读节点

1. 进入POLARDB控制台。
2. 选择地域。

3. 进入**增删节点向导**页面。您可以按照如下两种方式操作：

   - 找到目标集群，在**操作**列单击**增删节点**。

   - 找到目标集群，单击集群ID，在基本信息页面下方单击**增删节点**。

4. 勾选**增加节点**并单击**确定**。

5. 单击 + 增加一个只读节点，勾选服务协议，单击**去支付**并完成支付。

删除只读节点

1. 进入POLARDB控制台。
2. 选择地域。
3. 进入**增删节点向导**页面。您可以按照如下两种方式操作：

   - 找到目标集群，在**操作**列单击**增删节点**。

   - 找到目标集群，单击集群ID，在基本信息页面下方单击**增删节点**。

4. 勾选**删除节点**并单击**确定**。

5. 单击想要删除的节点后面的 — ，并在弹出对话框中单击**确定**。

> **说明:**
> 集群中必须保留至少一个只读节点，以保障集群的高可用。

6. 勾选服务协议，单击**确认**。

相关API

API	描述
#unique_68	增加POLARDB集群节点
#unique_69	变更POLARDB集群节点规格
#unique_70	重启POLARDB集群节点
#unique_71	删除POLARDB集群节点

12.7 设置可维护窗口

本文为您介绍如何设置可维护窗口，以便您在维护过程中不会影响业务。

背景信息

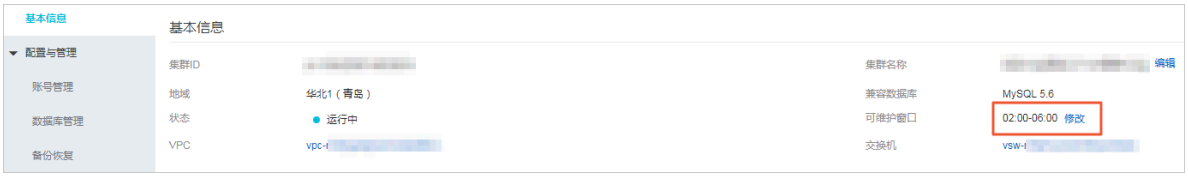
在阿里云平台上，为保障云数据库POLARDB的稳定性，后端系统会不定期对集群进行维护操作，确保集群平稳运行。您可以根据业务规律，将可维护窗口设置在业务低峰期，以免维护过程中对业务造成影响。

注意事项

- 在进行正式维护前，POLARDB 会给阿里云账号中设置的联系人发送短信和邮件，请注意查收。
- 集群维护当天，为保障整个维护过程的稳定性，集群会在所设置的可维护窗口之前的一段时间，进入集群维护中的状态，当集群处于该状态时，数据库本身正常的数据访问不会受到任何影响，但该集群的控制台上，除了账号管理、数据库管理和添加 IP 白名单外，其他涉及变更类的功能均无法使用（如常用的升降级、重启等操作均无法重启），查询类如性能监控等可以正常查阅。
- 在进入集群所设置的可维护窗口后，集群会在该段时间内发生1到2次的连接闪断，请确保您的应用程序具有重连机制。闪断后，集群即可恢复到正常状态。

操作步骤

1. 登录POLARDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 单击目标集群ID。
4. 在**基本信息**中的**可维护窗口**后，单击**修改**。



5. 在编辑**可维护窗口**中，选择集群的可维护窗口，单击**提交**。

相关API

API	描述
CreateDBCluster	创建数据库集群。
ModifyDBClusterMaintainTime	修改集群可运维时间。

12.8 重启节点

本文介绍当数据库节点出现连接数满或性能问题时，如何手动重启节点。

背景信息

当数据库节点出现连接数满或性能问题时，您可以手动重启节点。重启节点会造成连接中断，重启前请做好业务安排，谨慎操作。

操作步骤

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 单击目标集群ID。
4. 在**基本信息**页面下方**节点信息**里找到需要重启的节点。

5. 单击**操作**栏中的**重启**。
6. 在弹出的对话框中，单击**确认**。

相关API

API	描述
#unique_70	重启数据库节点。

12.9 释放集群

根据业务需求，您可以手动释放后付费（按小时付费）的集群。

注意事项

- 预付费（包年包月）集群不支持手动释放，集群到期后会自动被释放。
- 只有在运行状态下的集群才能被手动释放。
- 集群被释放后，数据将无法找回，请谨慎操作。
- 本功能用于释放整个集群，包括集群中的所有节点。如要释放单个只读节点，请参考增加或删除节点。
- 按小时付费的集群可以直接转为包年包月，具体请参见按小时付费转包年包月。

操作步骤

1. 登录POLARDB控制台。
2. 选择地域。

3. 找到目标集群，单击**操作**列的**...** > **释放**。

4. 在弹出的提示框中，单击**确认**。

相关API

API	描述
#unique_76	查看集群列表
#unique_77	删除数据库集群

12.10 克隆集群

您可以根据已有的PolarDB集群的数据（包括账号信息，但不包括集群参数配置信息），克隆出相同的PolarDB集群。

注意事项

被克隆的是执行克隆动作时的数据。克隆开始后，新写入的数据不会被克隆。

操作步骤

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 找到要克隆的集群，单击**操作**列的**...** > **恢复到新集群**。
4. 在配置页面设置以下参数：

参数	说明
克隆源类型	这里选择**本集群**。
地域	指集群所在的地理位置。克隆集群的地域和原集群相同，不支持修改。
主可用区	• 可用区是地域中的一个独立物理区域，不同可用区之间没有实质性区别。 • 您可以选择将PolarDB与ECS创建在同一可用区或不同的可用区。
网络类型	• 无需选择。 • 仅支持专有网络VPC（Virtual Private Cloud）。VPC是一种隔离的网络环境，安全性和性能均高于传统的经典网络。

参数	说明
VPC网络 VPC交换机	从下拉菜单中选择VPC和交换机，或者创建新的VPC和交换机。 **说明:** 请确保PolarDB与需要连接的ECS创建于同一个VPC，否则它们无法通过内网互通，无法发挥最佳性能。
数据库引擎	无需选择。
节点规格	按需选择。不同规格有不同的最大存储容量和性能，具体请参见#unique_28。
节点个数	无需选择。系统将自动创建一个与主节点规格相同的只读节点。
集群名称	• 可选。 • 如果留空，系统将为您自动生成一个集群名称。创建集群后还可以修改集群名称。
购买时长	预付费集群需要填写此参数。
购买数量	默认为1，无法修改。

5. 阅读并勾选**《云数据库 PolarDB服务协议》**，单击**去开通**。

12.11 小版本升级

PolarDB PostgreSQL支持手动升级内核小版本，内核小版本的升级涉及性能提升、新功能或问题修复等。

注意事项

- 升级内核小版本会重启集群，请您尽量在业务低峰期执行升级操作，或确保您的应用有自动重连机制。
- 升级内核小版本后无法降级。

操作步骤

1. 登录PolarDB管理控制台。

2. 在页面左上角，选择集群所在地域。

3. 找到目标集群，单击集群ID。

4. 在**基本信息**页面中单击**升级到最新版本**。

> **说明：**
> 如果您的集群内核小版本已经是最新版本，将不会显示**升级到最新版本**按键。

5. 在**升级到最新版本**对话框中单击**确定**。

> **说明：**
> 升级过程有60s左右的连接闪断，请确保业务有重连机制。

12.12 主备切换

PolarDB是一个多节点的集群，其中一个节点是主节点（Master），其他节点为只读节点。除了因系统故障自动进行主备切换外，您也可以手动进行主备切换，指定一个只读节点为新的主节点，适用于高可用演练，或者需要指定某个节点为主节点的场景。

影响

PolarDB集群进行主备切换时，可能会出现30秒左右的闪断，请您尽量在业务低峰期执行升级操作，并且确保您的应用有自动重连机制。

手动切换

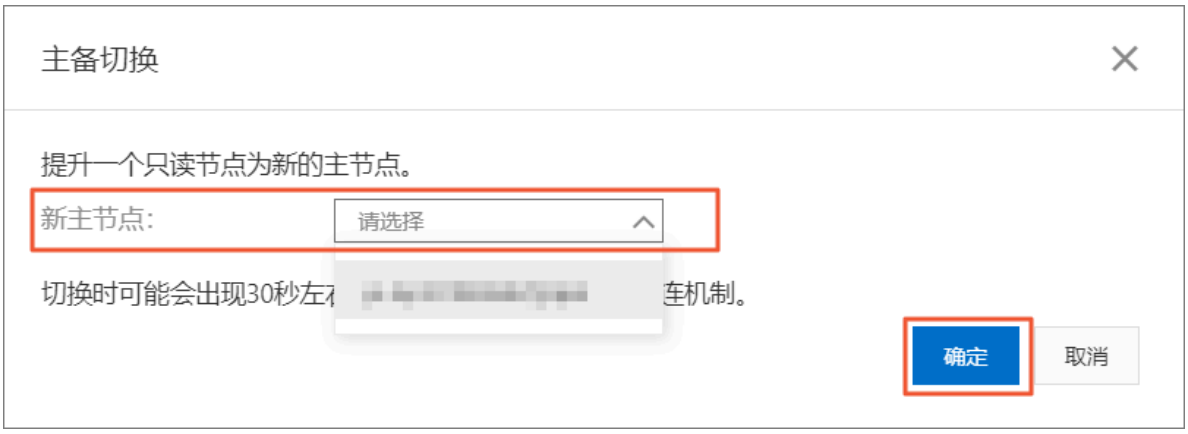
1. 登录PolarDB管理控制台。
2. 在页面左上角，选择集群所在地域。

3. 找到目标集群，单击集群ID。
4. 在页面，将浏览器拖动到最下方。
5. 在**节点信息**栏右上角，单击**主备切换**。

6. 在**主备切换**对话框中选择**新主节点**，单击**确定**。

说明：

如果您没有选择**新主节点**，系统将自动选取优先级最高的只读节点作为新主节点进行切换。切换时可能会出现30秒左右的闪断，请确保应用具备重连机制。



自动切换

PolarDB采用双活（Active-Active）的高可用集群架构，可读写的主节点和只读节点之间自动进行故障切换（Failover），系统自动选取新的主节点。

PolarDB每个节点都有一个故障切换（Failover）优先级，决定了故障切换时被选取为主节点的概率高低。当多个节点的优先级相同时，则有相同的概率被选取为主节点。

自动选取主节点按以下步骤进行：

1. 系统找出当前可以被选取的所有只读节点。
2. 系统选择优先级最高的一个或多个只读节点。
3. 如果切换第一个节点失败（例如，网络原因、复制状态异常等），系统会尝试切换下一个，直至成功。

相关API

API	描述
#unique_80	手动对PolarDB集群进行主备切换，可以指定一个只读节点为新的主节点。

12.13 多可用区部署和更换主可用区

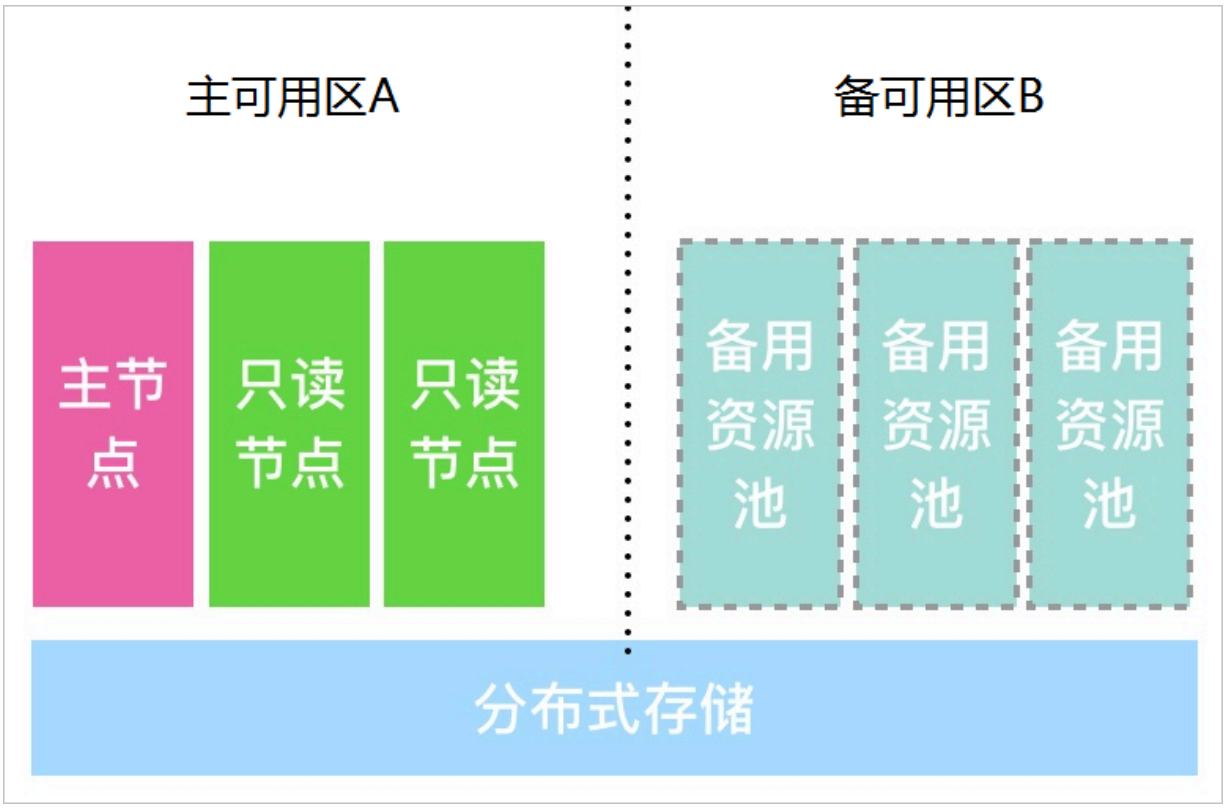
PolarDB-P支持创建多可用区的集群。相比单可用区集群，多可用区集群具备更高的容灾能力，可以抵御机房级别的故障。本文将为您介绍如何实施多可用区部署以及如何更换主可用区。

前提条件

- 可用区数量为两个及以上的地域。
- 目标可用区拥有足够计算资源。

多可用区架构

使用多可用区集群时，数据分布在多个可用区内。计算节点暂时要求位于主可用区，PolarDB会在备可用区预留足够的资源用于主可用区故障时进行故障切换。多可用区架构如下。



费用

多可用区功能不需要支付额外费用。

说明:
单可用区集群也会免费升级至多可用区集群。

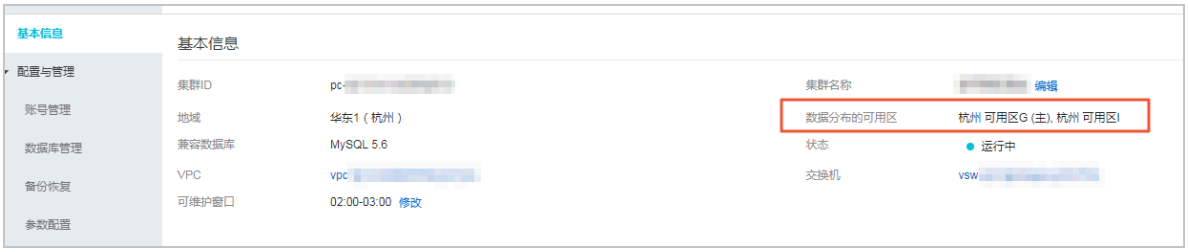
如何实现多可用区架构

当满足前提条件时，新建集群会默认为多可用区集群。

存量的单可用区集群也会升级至多可用区集群，该升级通过在线迁移数据的方式自动完成，对您的业务无任何影响。

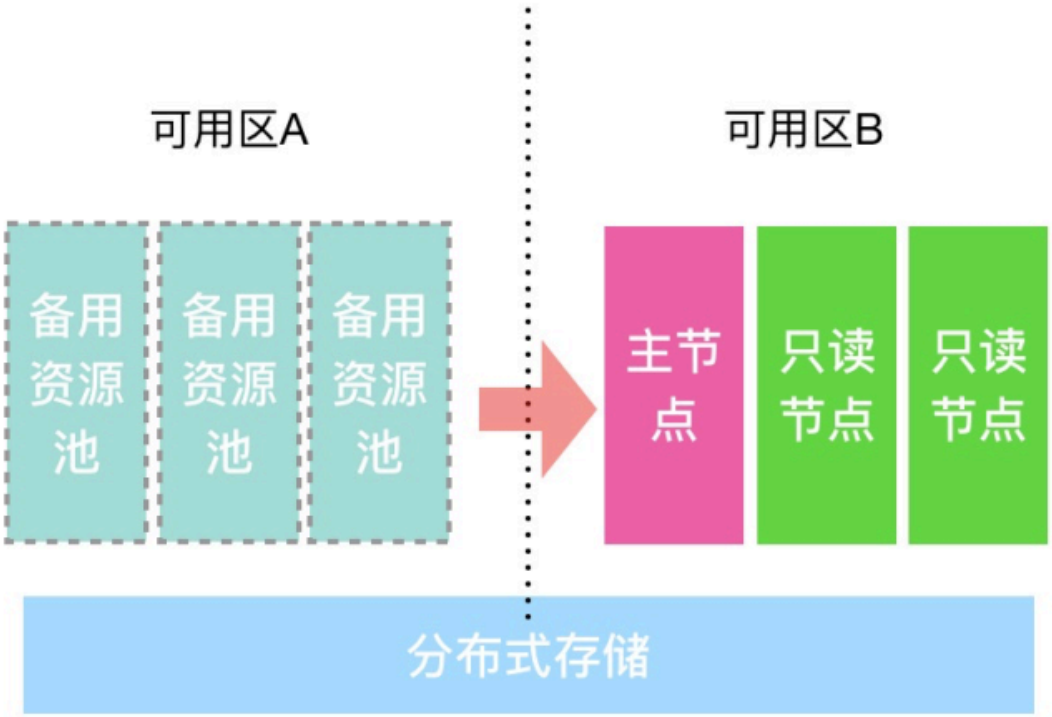
查看集群所属可用区

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 单击目标集群ID。
4. 在**基本信息**页面，查看**数据分布的可用区**。



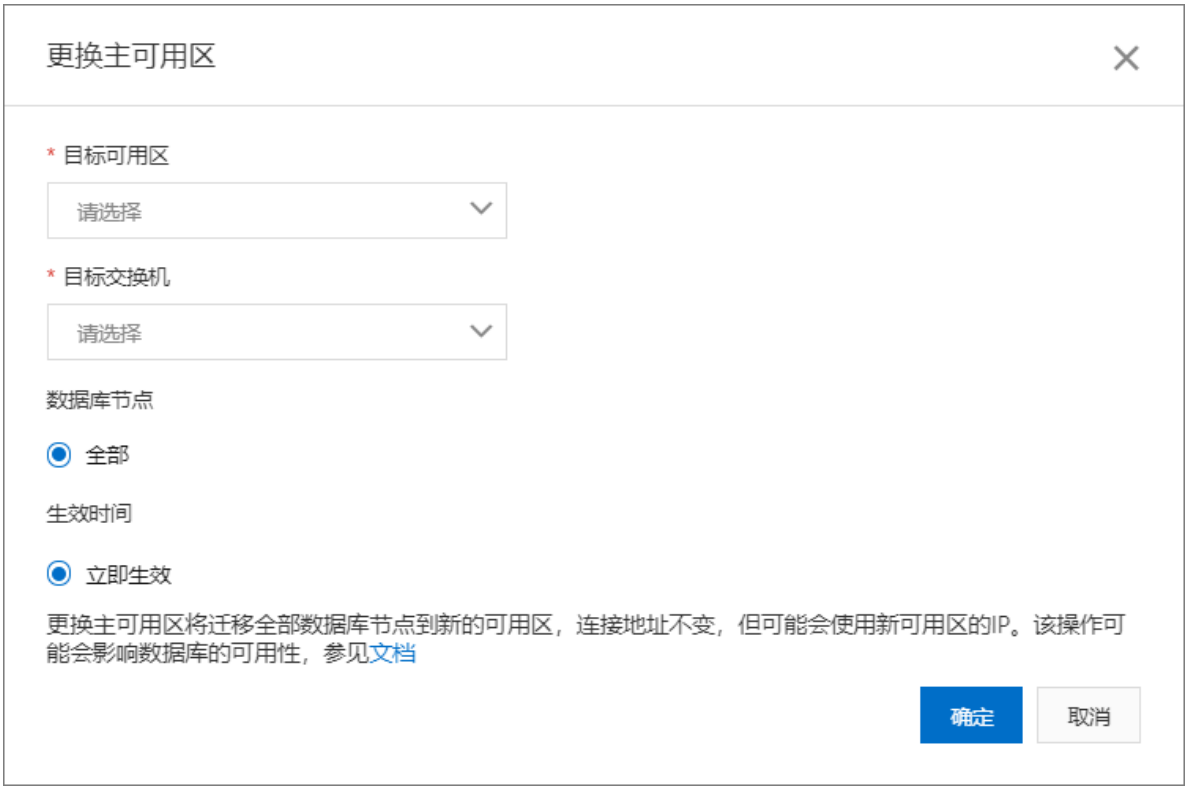
更换主可用区

PolarDB支持更换主可用区，您可以通过该功能将数据库集群计算节点迁移到其他可用区，适用于灾难恢复或者让ECS就近访问的场景。



1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 单击目标集群ID。
4. 在**基本信息**页面的**节点信息**区域，单击**迁移可用区**。

5. 在**更换主可用区**对话框中，选择**目标可用区**和**目标交换机**。



说明：

- 如果目标可用区是备可用区，则不需要迁移数据。系统只需要切换数据库计算节点，因此可以达到比较快的跨机房切换效果（平均耗时5分钟/节点），该操作常用于容灾演练。
- 如果目标可用区不在备可用区，则需要迁移数据。系统执行迁移时间长短跟数据容量有关，可能需要几个小时，请谨慎操作。该操作一般用于调整应用和数据库的可用区分布，达到就近访问数据库的目的。

6. 单击**确认**。

注意：

更换主可用区后，数据库连接地址（集群访问地址和主访问地址）不变，但使用的虚拟交换机vSwitch和IP地址可能会发生变化。该操作可能会对数据库服务可用性造成1分钟以内的影响，请谨慎操作。

13 账号

13.1 账号概述

控制台账号

您可以使用以下账号登录控制台：

- **阿里云账号**：该账号是阿里云资源的归属和计费主体。购买阿里云产品之前，您需要先注册阿里云账号。
- **子账号**（可选）：如果其他人需要使用您账号下的资源，您可以使用RAM控制台创建和管理子账号。子账号本身不拥有资源，且以主账号作为计费主体。

数据库集群账号

您可以使用以下账号登录数据库集群。更多信息请参见#unique_84。

账号类型	说明
高权限账号	• 只能通过控制台创建和管理。 • 一个集群只能有一个高权限账号，可以管理所有普通账号和数据库。 • 开放了更多权限，可满足个性化和精细化的权限管理需求，比如可按用户分配不同表的查询权限。 • 拥有集群中所有数据库的所有权限。 • 可以断开任意账号的连接。

相关API

API	描述
#unique_85	创建账号
#unique_86	查看账号列表
#unique_87	修改账号备注
#unique_88	修改账号密码
#unique_89	账号授权
#unique_90	撤销账号权限
#unique_91	重置账号权限
#unique_92	删除账号

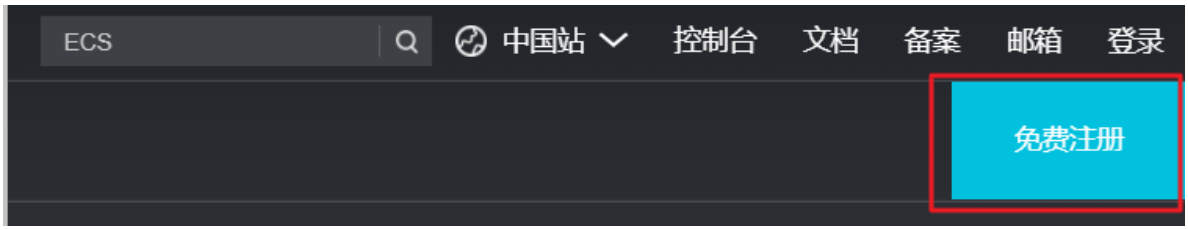
13.2 注册和登录阿里云账号

本文介绍如何注册和登录阿里云账号。

注册阿里云账号

您可以通过两种方式注册阿里云账号：

- 进入阿里云官网，单击右上角的**免费注册**。



- 直接访问注册页面。

登录阿里云账号

阿里云账号的登录入口与子账号不同。

- 阿里云账号的登录入口。


密码登录
扫码登录
淘宝及1688会员可直接使用会员名登录
邮箱/会员名/8位ID
登录密码
登录
忘记密码
忘记会员名
免费注册
其他方式登录：

- 子账号的登录入口。

13.3 创建和使用子账号

如果只有您本人使用PolarDB，那么使用阿里云账号即可。如果需要让其他人使用您账号下的资源，请创建子账号。

创建RAM子账号

1. 您可以使用阿里云账号或有RAM权限的子账号来创建子账号，首先需要登录RAM控制台。具体操作如下：
   - 如果使用阿里云账号，请使用阿里云账号登录。
   - 如果使用子账号，请使用RAM用户登录。

   > **说明：**
   > 子账号登录的格式为子账号名@公司别名。

2. 在左侧导航栏的**人员管理**菜单下，单击**用户**。

3. 单击**新建用户**。

> **说明:**
> 单击**添加用户**，可一次性创建多个RAM用户。

4. 输入**登录名称**和**显示名称**。
5. 在**访问方式**区域下，选择**控制台密码登录**。
6. **控制台密码**选择**自动生成默认密码**或**自定义登录密码**。
7. **要求重置密码**选择**用户在下次登录时必须重置密码**或**无需重置**。
8. **多因素认证**选择**不要求**。
9. 单击**确认**。

在授权页面下为RAM用户授权

1. 在左侧导航栏的**权限管理**菜单下，单击**授权**。
2. 单击**新增授权**。
3. 在**被授权主体**区域下，输入RAM用户名称后，单击需要授权的RAM用户。
4. 在左侧**权限策略名称**列表下，单击需要授予RAM用户的权限策略。

   可选策略如下所示：

权限策略名称	说明
AliyunPolarDBReadOnlyAccess	只读访问云数据库PolarDB的权限。
AliyunPolarDBFullAccess	管理云数据库PolarDB的权限。

> **说明:**
> 在右侧区域框，选择某条策略并单击×，可撤销该策略。

5. 单击**确定**。
6. 单击**完成**。

在用户页面下为RAM用户授权

1. 在左侧导航栏的**人员管理**菜单下，单击**用户**。
2. 在**用户登录名称/显示名称**列表下，找到目标RAM用户。
3. 单击**添加权限**，被授权主体会自动填入。

4. 在左侧**权限策略名称**列表下，单击需要授予RAM用户的权限策略。

   可选策略如下所示：

权限策略名称	说明
AliyunPolardbReadOnlyAccess	只读访问云数据库PolarDB的权限。
AliyunPolardbFullAccess	管理云数据库PolarDB的权限。

   说明：
   在右侧区域框，选择某条策略并单击×，可撤销该策略。

5. 单击**确定**。
6. 单击**完成**。

登录子账号

前提条件：您已完成上述账号授权步骤。

您可以通过两种地址登录子账号：

- 通用登录地址：RAM用户登录

  如果通过此地址登录，您需手动输入子账号名以及公司别名。格式为`子账号名@公司别名`。

- 专用登录地址：如果您可以登录RAM控制台，可以在RAM控制台查看到您公司的子账号登录地址。

如果通过此地址登录，系统将自动为您填写公司别名，您只需输入子账号名。

更多操作

您还可以对子账号进行更多的操作，如把子账号添加到用户组、为子账号分配角色、为用户组或角色授权等。详情请参见RAM用户指南。

13.4 创建数据库账号

本文为您介绍如何创建数据库账号，以及高权限账号与普通账号的区别。

背景信息

PolarDB支持两种数据库账号：高权限账号和普通账号。您可以在控制台管理所有账号。

账号类型

账号类型	说明
高权限账号	• 只能通过控制台或API创建和管理。 • 一个集群可以创建多个高权限账号，高权限账号可以管理所有普通账号和数据库。 • 开放了更多权限，可满足个性化和精细化的权限管理需求，例如可按用户分配不同表的查询权限。 • 拥有集群中所有数据库的所有权限。 • 可以断开任意账号的连接。
普通账号	• 可以通过控制台或者SQL语句创建和管理。 • 一个集群可以创建多个普通账号，具体的数量与数据库内核有关 。 • 需要手动给普通账号授予特定数据库的权限。 • 普通账号不能创建和管理其他账号，也不能断开其他账号的连接。

创建账号

1. 登录PolarDB控制台。
2. 在页面左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，单击**配置与管理** > **账号管理**。
5. 单击**创建账号**。
6. 设置以下参数：

参数	说明
账号名	填写账号名称。要求如下： • 以小写字母开头，以字母或数字结尾。 • 由小写字母、数字或下划线组成。 • 长度为2~16个字符。 • 不能使用某些预留的用户名，如root、admin。
账号类型	• 选择**高权限账号**，创建高权限账号。 • 选择**普通账号**，创建普通账号。

参数	说明
密码	设置账号的密码。要求如下： • 由大写字母、小写字母、数字或特殊字符组成，至少包含其中三类。 • 长度为8~32个字符。 • 特殊字符为： - ! - @ - # - $ - % - ^ - & - * - (-) - _ - +
确认密码	再次输入密码。
备注	备注该账号的相关信息，便于后续账号管理。要求如下： • 不能以http://或https://开头。 • 必须以大小写字母或中文开头。 • 可以包含大小写字母、中文、数字、下划线（_）或连字符（-）。 • 长度为2~256个字符。

7. 单击**确定**。

下一步

查看连接地址

相关API

API	描述
#unique_94	创建账号。
#unique_95	查看账号列表。
#unique_96	修改账号备注。
#unique_97	修改账号密码。
#unique_98	删除账号。

13.5 管理数据库账号

本文为您介绍如何管理数据库账号，包括修改密码、锁定账号、解锁账号和删除账号等。

背景信息

POLARDB支持两种数据库账号：高权限账号和普通账号。您可以在控制台管理所有账号。

创建数据库账号

具体操作请参见创建数据库账号。

修改密码

1. 登录POLARDB控制台。
2. 在页面左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，单击**配置与管理** > **账号管理**。
5. 在目标账号操作栏中，单击**修改密码**。

6. 在弹出的对话框中，输入**新密码**和**确认新密码**，单击**确定**。

锁定账号

您可以通过锁定账号功能，锁定目标账号，禁止使用该账号登录数据库。

1. 登录POLARDB控制台。
2. 在页面左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，单击**配置与管理** > **账号管理**。
5. 在目标账号操作栏中，单击**锁定**。
6. 在弹出的对话框中，单击**确定**。

解锁账号

1. 登录POLARDB控制台。
2. 在页面左上角，选择地域。

3. 单击目标集群ID。
4. 在左侧导航栏中，单击**配置与管理** > **账号管理**。
5. 在目标账号操作栏中，单击**解锁**。
6. 在弹出的对话框中，单击**确定**。

删除账号

1. 登录POLARDB控制台。
2. 在页面左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，单击**配置与管理** > **账号管理**。
5. 在目标账号操作栏中，单击**删除账号**。
6. 在弹出的对话框中，单击**确定**。

相关API

API	描述
#unique_85	创建账号
#unique_86	查看账号列表
#unique_87	修改账号备注
#unique_88	修改账号密码
#unique_89	账号授权
#unique_90	撤销账号权限
#unique_91	重置账号权限
#unique_92	删除账号

14 数据库

本文为您介绍如何创建和删除PolarDB PostgreSQL数据库。

创建数据库

1. 登录PolarDB控制台。
2. 在控制台左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，选择**配置与管理 > 数据库管理**。
5. 单击**创建数据库**。

6. 设置以下参数。

参数	说明
数据库（DB）名称	• 以字母开头，以字母或数字结尾； • 由小写字母、数字、下划线或中划线组成； • 长度为2~64个字符。 • 数据库名称在实例内必须是唯一的。
数据库Owner	数据库的所有者，对数据库拥有ALL权限。
支持字符集	数据库支持的字符集，默认为UTF8。 如果需要其他字符集，请在下拉列表中选择需要的字符集。
Collate	字符串排序规则。
Ctype	字符分类。
备注说明	用于备注该数据库的相关信息，便于后续数据库管理。要求如下： • 不能以http://或https://开头。 • 必须以大小写字母或中文开头。 • 可以包含大小写字母、中文、数字、下划线"_"或连字符"-"。 • 长度为2~256个字符。

7. 单击**确定**。

删除数据库

1. 登录PolarDB控制台。
2. 在控制台左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，选择**配置与管理** > **数据库管理**。
5. 单击目标数据库操作栏中的**删除**。
6. 在弹出的对话框中，单击**确认**。

相关API

API	描述
#unique_101	创建数据库
#unique_102	查看数据库列表信息
#unique_103	修改数据库描述
#unique_104	删除数据库

15 备份与恢复

15.1 备份数据

可靠的备份功能可以有效防止数据丢失，PolarDB PostgreSQL支持周期性的自动备份以及即时生效的手动备份。在删除PolarDB PostgreSQL集群时，您还可以选择保留备份数据。

特色功能

PolarDB支持长期保留备份文件，并且可以设置删除实例后继续保留备份文件，避免了误操作导致的数据丢失。

一级备份采用了ROW（Redirect-on-Write）快照的方式。 每次保存快照并没有真正复制数据，当数据块有修改时系统会将其中一个历史版本的数据块保留给快照，同时生成新的数据块被原数据引用（Redirect）。因此无论数据库容量多少，都可以做到秒级备份。

日志备份通过实时并行上传Redo日志到OSS来达到备份的目的。通过一个完整的数据全量备份（快照）以及后续一段时间的Redo日志，就可以将PolarDB集群恢复到任意时间点（Point-In-Time Recovery，简称：PITR）。

PolarDB集群备份和恢复功能均采用多线程并行处理来提高效率。目前，一级备份（快照）进行恢复或克隆的速度为40分钟/TB；如果您需要**按时间点恢复**，则需要包含应用Redo日志的时间，应用Redo日志速度为20秒/GB至70秒/GB ，整个恢复时间是快照恢复时间以及应用Redo日志时间之和。

警告：

如果您需要关闭备份功能，可以提交工单联系售后客服进行关闭，但您须自行承担关闭备份所带来的风险（如误操作导致的数据丢失等）。

产品定价

PolarDB备份和恢复功能均免费使用，但备份文件需要占用一定的存储空间，PolarDB会根据备份文件（数据+日志）的存储容量和保存时长收取一定费用。

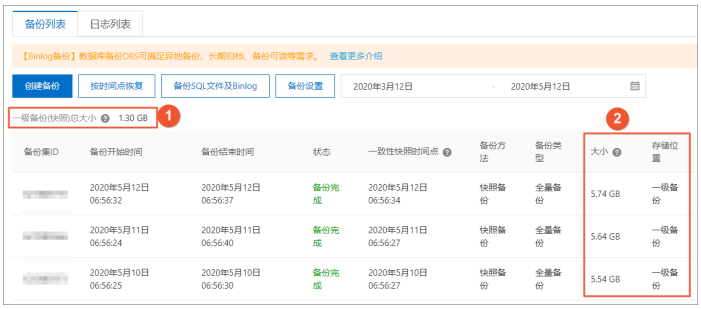
说明：

PolarDB备份功能于2020年5月11日商业化，为了您可以更方便地了解备份功能以及备份文件的存储空间情况，在接下来的一周时间，PolarDB备份不会产生任何费用。详细信息请参见#unique_107。

表 15-1: 价格表

地域	一级备份	二级备份	日志备份
中国内地	0.003元/GB/小时	0.00021元/GB/小时	0.00021元/GB/小时
中国香港及海外	0.0042元/GB/小时	0.000294元/GB/小时	0.000294元/GB/小时

计费方式

备份类型	免费额度	计费方式
一级备份	数据库存储用量 x 50% 您可以在控制台**基本信息**页面查看数据库存储用量。	每小时费用 =（ 一级备份总大小 - 免费额度）x 每小时价格 • 一级备份小于免费额度时，一级备份不收取任何费用。 • 每小时单价请参见表 15-1: 价格表。 • **一级备份（快照）总大小**如下图中①所示，而非②。  示例：一级备份（快照）总大小为700GB，数据库存储用量为1000GB，那么每小时费用为0.6元。 计算公式：[700GB -（1000GB x 50%）] x 0.003元 = 0.6元
二级备份	无	每小时费用 = 二级备份总大小 x 每小时价格 示例：二级备份总大小为1000GB，那么每小时费用为0.21元。 计算公式：1000GB x 0.00021元 = 0.21元
日志备份	100GB	每小时费用 =（日志备份总大小 - 100GB） x 每小时价格 示例：日志备份总大小为1000GB，那么每小时费用为0.189元。 计算公式：（1000GB - 100GB） x 0.00021元 = 0.189元

备份类型

备份类型	说明
一级备份（数据备份）	直接存储在分布式存储集群上，备份和恢复速度快，但保存成本高。 最短保留时间为7天，最长保留时间为14天。 一级备份（快照）总大小如下图所示。 
二级备份（数据备份）	指一级备份压缩后保存在其它离线存储介质上的备份数据，使用二级备份恢复数据的速度较慢，但其保存成本较低。 最短保留时间为30天，最长保留时间为7300天，也可以开启功能。 **说明：** • 二级备份功能默认关闭。 • 开启二级备份后，一级备份超出您设置的保存时间将后自动转存为二级备份，转存速度在150MB/s左右。 • 如果转存二级备份较久，到下次转存二级备份时仍没有完成，将跳过下次转存。例如每天凌晨1点自动备份1次，一级备份A在凌晨1点过期开始转存为二级备份，由于备份文件较大转存时间较久，第二天凌晨1点该转存任务仍未完成，一级备份B在第二天凌晨1点到期后将直接删除而不会转存为二级备份。
日志备份	日志备份是把数据库的Redo日志保存下来，用于按时间点恢复数据，保证最近一段时间的数据安全性，避免误操作导致的数据丢失。 日志备份最短保留时间为7天，最长保留时间为7300天，您也可以通过开启功能永久保存。

备份方式

备份方式	说明
系统备份（自动）	• 自动备份默认为每天1次，您可以设置自动执行备份的时间段和周期。具体请参见设置自动备份。 • 备份文件不可删除。 **说明：** 出于安全考虑，自动备份的频率为每周至少2次。
主动备份	• 您可以可随时发起主动备份。每个集群最多可以有3个主动创建的备份。具体请参见手动创建备份。 • 备份文件可删除。

设置自动备份

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 找到目标集群，单击集群ID。
4. 在左侧导航栏中，选择**配置与管理** > **备份恢复**。
5. 单击**备份设置**。

6. 在弹出的窗口中，设置如下参数。

参数	说明
数据备份方式	默认为**快照备份**。

参数	说明
数据备份周期	设置数据自动备份的周期。 **说明：** 出于安全考虑，自动备份的频率为每周至少2次。
备份开始时间	设置自动备份开始的时间。
一级备份保留	设置一级备份保留时间。 **说明：** 一级备份最短保留时间为7天，最长保留时间为14天。
二级备份开关	开启或关闭二级备份。 **说明：** 二级备份默认为关闭状态。
二级备份保留	设置二级备份的保留时间。 **说明：** • 二级备份最短保留时间为30天，最长保留天数为7300天。 • 如果您需要永久保存二级备份，可以选中**删除集群前永久保留**，选中后将无法设置保留天数。
日志备份保留	设置日志备份的保留时间。 **说明：** • 日志备份最短保留时间为7天，最长保留天数为7300天。 • 如果您需要永久保存日志备份，可以选中**删除集群前永久保留**，选中后将无法设置保留天数。

参数	说明
删除集群时	设置删除集群时的备份保留策略。 • **永久保留全部备份**：删除集群时保留所有备份。 • **永久保留最后一个备份（删除前自动备份）**：删除集群时保留最后一个备份。 • **立即删除该集群的所有备份**：删除集群时不保留任何备份。 **说明：** • 如果您选择了**永久保留全部备份**或**永久保留最后一个备份（删除前自动备份）**策略，删除PolarDB集群时，系统会主动发起1次备份，为您保存删除前的所有数据。 • 删除集群后，一级备份将自动转为二级备份，您可以在**集群回收站**中查看所有保存的备份，更多内容请参见#unique_108。

7. 完成备份设置后，单击**确定**。

手动创建备份

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 找到目标集群，单击集群ID。
4. 在左侧导航栏中，选择**配置与管理** > **备份恢复**。
5. 在**备份列表**页签，单击**创建备份**。

6. 在弹出的**创建备份**对话框中，单击**确定**。

说明：
每个集群最多可以有3个手动创建的备份。

恢复数据

请参见#unique_109。

常见问题

- Q：一级备份大小是否为所有快照备份之和？

  A：一级备份大小不是所有快照备份之和，即并非下图中的②，而是①所示的大小。

- Q：为什么一级备份的总大小比单个备份还要小？

  A：PolarDB的一级备份有两个容量数据，一个是每个备份的逻辑大小，一个是全部备份的物理大小。PolarDB的一级备份采用快照链的机制，相同的数据块只会记录一份，因此总物理大小要小于逻辑大小，有时候甚至会小于单个备份逻辑大小。

- Q：PolarDB备份有哪些费用？

  A：一级备份、二级备份以及日志备份的存储空间费用。其中一级备份和日志备份默认开启，并赠送一定的免费空间。二级备份默认关闭。

- Q：一级备份的费用怎么算？

  A：每小时费用 = [一级备份总大小 - （数据库存储用量 x 50%）] x 每小时价格。例如，PolarDB数据库的一级备份总大小为700G，数据库存储用量为1000GB，那么每小时费用为（700G-500G）* 0.003元/GB = 0.6元。

- Q：存储包能否抵扣备份空间的费用？

  A：目前存储包只能抵扣数据存储占用的空间，不能抵扣备份占用的空间。

相关API

API	描述
#unique_110	创建PolarDB集群全量快照备份。
#unique_111	查询PolarDB集群备份信息。
#unique_112	删除PolarDB集群备份。
#unique_113	查询PolarDB集群自动备份策略。

API	描述
#unique_114	修改PolarDB集群自动备份策略。

15.2 恢复数据

POLARDB for PostgreSQL支持按备份集（快照）恢复，将历史数据恢复到新集群中。

说明:

恢复后的集群包含原集群的数据和账号信息，不包含原集群的参数设置。

按备份集（快照）恢复

1. 进入POLARDB控制台。
2. 选择集群所在的地域。
3. 找到目标集群，单击集群ID。
4. 在左侧导航栏中，选择**配置与管理** > **备份恢复**。
5. 找到目标备份集（快照），单击**恢复备份**，在弹出的对话框中单击**确认**。
6. 在弹出的页面中，选择新集群的计费方式：
   - **预付费**：在创建集群时需要支付计算实例（一个主实例和一个只读实例）的费用，而存储空间会根据实际数据量按小时计费，并从账户中按小时扣除。如果您要长期使用该集群，**预付费**方式更加划算，而且购买时长越长，折扣越多。
   - **按量付费**：无需预先支付费用，计算实例和存储空间（实际数据量）均按小时计费，并从账户中按小时扣除。如果您只需短期使用该集群，可以选择**按量付费**，用完即可释放，节省费用。

7. 设置以下参数：
   - **克隆源类型**：选择**备份集**。
   - **克隆源备份集**：请确认是否为您要恢复的备份集。
   - **地域**：无需修改，与原集群相同。
   - **可用区**：无需修改。
   - **网络类型**：无需修改。
   - **VPC网络**和**VPC交换机**：建议保持不变，即原集群所在的VPC网络和交换机。
   - **数据库引擎**：无需修改。
   - **节点规格**：不同规格有不同的最大存储容量和性能，具体请参见节点规格。
   - **节点个数**：无需修改。系统将自动创建一个与主节点规格相同的只读节点。
   - **集群名称**：如果留空，系统将为您自动生成一个集群名称。创建集群后还可以修改集群名称。
   - **购买时长**：预付费集群需要填写此参数。
   - **集群数量**：默认为1，无法修改。
8. 阅读并勾选**《云数据库 POLARDB服务协议》**，然后完成支付。

相关主题

#unique_117

相关API

API	描述
#unique_39	创建POLARDB集群。 **说明：** 克隆集群时，参数**CreationOption**取值需要为**CloneFromPolarDB**。

15.3 备份功能FAQ

本文介绍PolarDB MySQL备份功能的常见问题。

- Q：一级备份大小是否为所有快照备份之和？

  A：一级备份大小不是所有快照备份之和，即并非下图中的②，而是①所示的大小。

- Q：为什么一级备份的总大小比单个备份还要小？

  A：PolarDB的一级备份有两个容量数据，一个是每个备份的逻辑大小，一个是全部备份的物理大小。PolarDB的一级备份采用快照链的机制，相同的数据块只会记录一份，因此总物理大小要小于逻辑大小，有时候甚至会小于单个备份逻辑大小。

- Q：PolarDB备份有哪些费用？

  A：一级备份、二级备份以及日志备份的存储空间费用。其中一级备份和日志备份默认开启，并赠送一定的免费空间。二级备份默认关闭。

- Q：一级备份的费用怎么算？

  A：每小时费用 = [一级备份总大小 - （数据库存储用量 x 50%）] x 每小时价格。例如，PolarDB数据库的一级备份总大小为700G，数据库存储用量为1000GB，那么每小时费用为（700G-500G）* 0.003元/GB = 0.6元。

- Q：存储包能否抵扣备份空间的费用？

  A：目前存储包只能抵扣数据存储占用的空间，不能抵扣备份占用的空间。

相关文档

#unique_119

16 数据安全/加密

16.1 设置SSL加密

为了提高链路安全性，您可以启用SSL（Secure Sockets Layer）加密，并安装SSL CA证书到需要的应用服务。SSL在传输层对网络连接进行加密，能提升通信数据的安全性和完整性，但会同时增加网络连接响应时间。

注意事项

- SSL的证书有效期为1年，请及时更新证书有效期并重新下载和配置CA证书，否则使用加密连接的客户端程序将无法正常连接。
- 由于SSL加密的固有缺陷，启用SSL加密会显著增加CPU使用率，建议您仅在外网链路有加密需求的时候启用SSL加密。内网链路相对较安全，一般无需对链路加密。
- 关闭SSL加密会重启集群，请谨慎操作。

开启SSL加密和下载证书

1. 登录PolarDB管理控制台。
2. 在页面左上角，选择集群所在地域。

3. 找到目标集群，单击集群ID。
4. 在左侧菜单栏中单击**配置与管理** > **安全管理**。

5. 在**SSL配置**页签，单击**SSL状态**右侧滑块，开启SSL加密。

6. 在**设置SSL**对话框中，单击**确定**。

7. SSL状态变为**已开通**后，单击**下载证书**。

下载的文件为压缩包，包含如下三个文件：

- p7b文件：用于Windows系统中导入CA证书。
- pem文件：用于其他系统或应用中导入CA证书。
- jks文件：Java中的truststore证书存储文件，密码统一为apsaradb，用于Java程序中导入CA证书链。

> **说明：**
> 在Java中使用JKS证书文件时，jdk7和jdk8需要修改默认的jdk安全配置，在连接PolarDB数据库的服务器的jre/lib/security/java.security文件中，修改如下两项配置：
>
> ```
> jdk.tls.disabledAlgorithms=SSLv3, RC4, DH keySize < 224
> jdk.certpath.disabledAlgorithms=MD2, RSA keySize < 1024
> ```
>
> 若不修改jdk安全配置，会报如下错误。其它类似报错，一般也都由Java安全配置导致。
>
> ```
> javax.net.ssl.SSLHandshakeException: DHPublicKey does not comply to algorithm constraints
> ```

更新证书有效期

如果您修改了SSL连接地址或证书有效期即将到期，您需要更新证书有效期，以下内容将为您介绍如何更新证书有效期。

1. 登录PolarDB管理控制台。

2. 在页面左上角，选择集群所在地域。

3. 找到目标集群，单击集群ID。

4. 在左侧菜单栏中单击**配置与管理** > **安全管理**。

5. 在**SSL配置**页签中单击**更新有效期**。

6. 在**设置SSL**对话框单击**确定**。

> **说明:**
> 更新有效期操作将会重启集群，重启前请做好业务安排，谨慎操作。

7. 更新有效期后，重新下载和配置证书。

> **说明:**
> 下载证书请参见开启SSL加密和下载证书步骤七。

关闭SSL加密

> **说明：**
>
> - 关闭SSL加密会重启集群，建议您在业务低峰期操作。
> - SSL加密关闭后，数据库访问性能会有一定程度提升，但安全性上有削弱，故非安全环境下不建议关闭SSL加密。

1. 登录PolarDB管理控制台。
2. 在页面左上角，选择集群所在地域。

3. 找到目标集群，单击集群ID。
4. 在左侧菜单栏中单击**配置与管理** > **安全管理**。
5. 在**SSL配置**页签中单击**SSL状态**右侧滑块，关闭SSL加密。

6. 在**设置SSL**对话框，单击**确定**。

常见问题

Q：SSL证书到期后不更新会有什么影响？会影响实例运行或数据安全吗？

A：SSL证书到期后不更新，会导致使用加密连接的客户端程序无法正常连接实例，不会影响实例运行或数据安全。

相关API

API	描述
#unique_122	调用DescribeDBClusterSSL接口查询PolarDB集群SSL设置。
#unique_123	调用ModifyDBClusterSSL接口设置PolarDB集群SSL加密的开通、关闭或更新CA证书。

16.2 设置透明数据加密TDE

PolarDB-P提供了透明数据加密TDE（Transparent Data Encryption）功能。TDE可对数据文件执行实时I/O加密和解密，数据在写入磁盘之前进行加密，从磁盘读入内存时进行解密。TDE不会增加数据文件的大小，开发人员无需更改任何应用程序，即可使用TDE功能。

前提条件

- 集群版本为PolarDB-P。
- 已开通KMS。如果您未开通KMS，可以免费开启秘钥管理服务。

背景信息

TDE通过在数据库层执行静止数据加密，阻止可能的攻击者绕过数据库直接从存储中读取敏感信息。经过数据库身份验证的应用和用户可以继续透明地访问应用数据（不需要更改应用代码或配置），而尝试读取表空间文件中的敏感数据的OS用户以及尝试读取磁盘或备份信息的不法之徒将不允许访问明文数据。

PolarDB-O TDE加密使用的密钥由密钥管理服务（KMS）产生和管理，PolarDB不提供加密所需的密钥和证书。您不仅可以使用阿里云自动生成的密钥，也可以使用自带的密钥材料生成数据密钥，然后授权PolarDB使用。

注意事项

- TDE开通后无法关闭。
- 仅支持在创建集群的过程中开启TDE。
- 开通TDE后，如果是I/O密集型（I/O bound）场景，可能会对数据库性能产生一定影响。

- 使用已有自定义密钥时，需要注意：
  - 禁用密钥、设置密钥删除计划或者删除密钥材料都会造成密钥不可用。
  - 撤销授权关系后，重启PolarDB集群会导致PolarDB集群不可用。
  - 需要使用主账号或者具有AliyunSTSAssumeRoleAccess权限的账号。

操作步骤

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 在**集群列表**页面，单击**创建新集群**。
4. 在**PolarDB购买页面**，配置PolarDB购买信息，选择**开启TDE**。

   说明：
   更多关于购买页面的信息，请参见创建PolarDB-P集群。

5. 单击页面右侧的**立即购买**。
6. 在确认订单页面，确认订单信息，阅读和勾选**服务协议**，单击**去支付**。

   说明：
   完成支付后，集群将在十分钟左右创建成功。

查看TDE状态

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 找到目标集群，单击集群ID。
4. 在左侧导航栏单击**配置与管理** > **安全管理**。

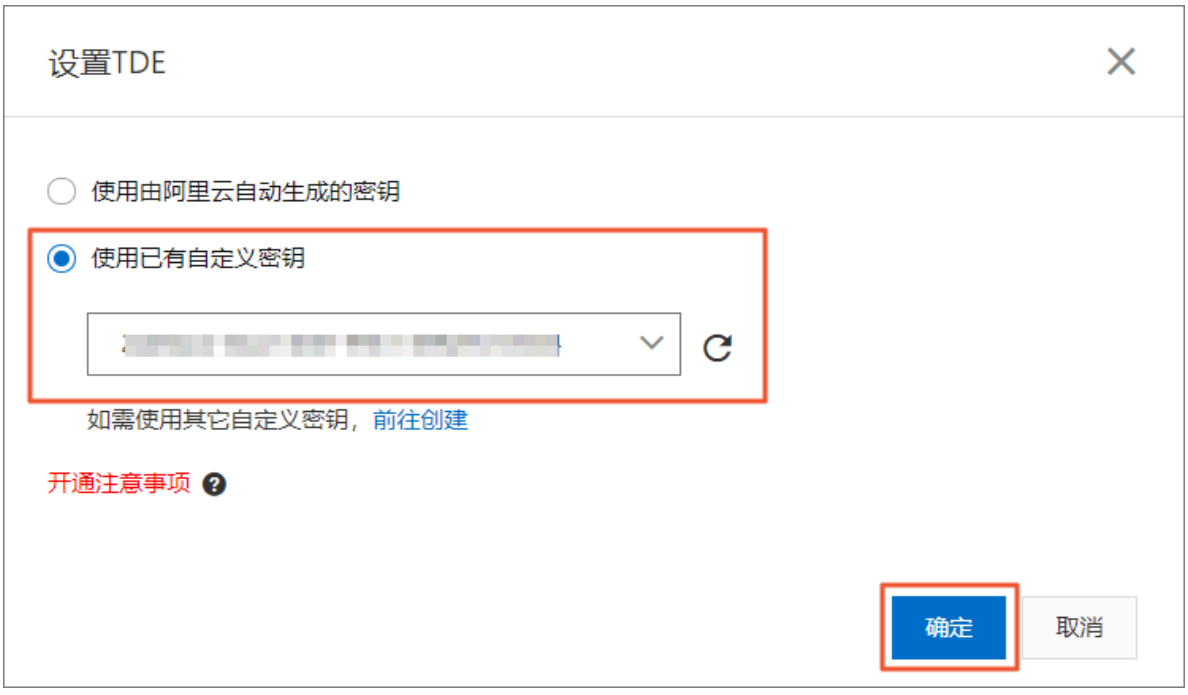
5. 在**TDE配置**页签，查看**TDE状态**。

切换为自定义密钥

1. 在左侧导航栏单击**配置与管理** > **安全管理**。
2. 在页签，单击**TDE状态**右侧的。

3. 在**设置TDE**对话框中，选择**使用已有自定义密钥**。

说明:
如果没有自定义密钥，需要单击**前往创建**，在密钥管理服务控制台创建密钥并导入自带的密钥材料。详情请参见管理密钥。



4. 单击**确定**。

常见问题

- 开启TDE后，常用数据库工具（Navicat等）还能正常使用吗？

  答：可以正常使用。

- 加密后查看数据为什么还是明文的？

  答：查询数据时会解密并读取到内存，所以是明文显示。开启TDE后存储的数据是加密的。

相关API

API	描述
#unique_57	创建PolarDB集群，开启透明数据加密TDE。  **说明:** DBType参数需要为PostgreSQL或Oracle。

17 诊断与优化

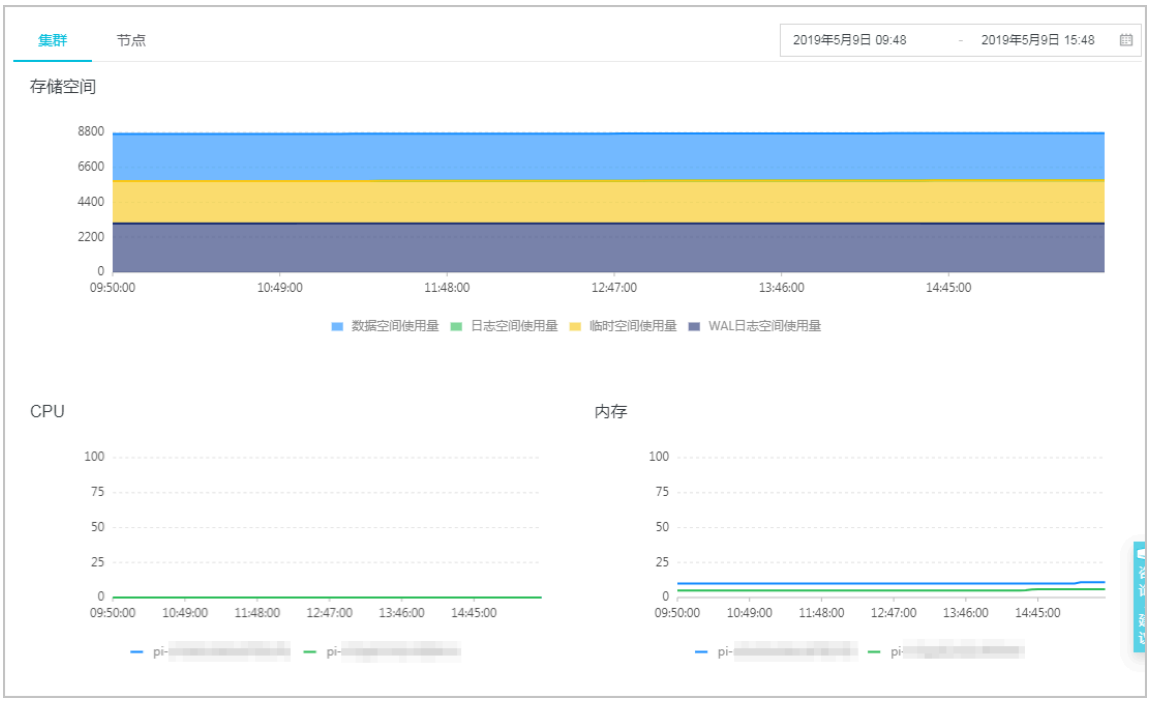
17.1 性能监控与报警

为方便您掌握实例的运行状态，POLARDB控制台提供了丰富的性能监控项。

性能监控

1. 进入POLARDB控制台。
2. 选择地域。
3. 找到目标集群，单击**集群名称**列的**集群ID**。
4. 在左侧导航栏中选择**诊断与优化** > **性能监控**。
5. 根据您的需求，可以查看**集群**或**节点**的监控信息。详细说明请参见监控项说明。

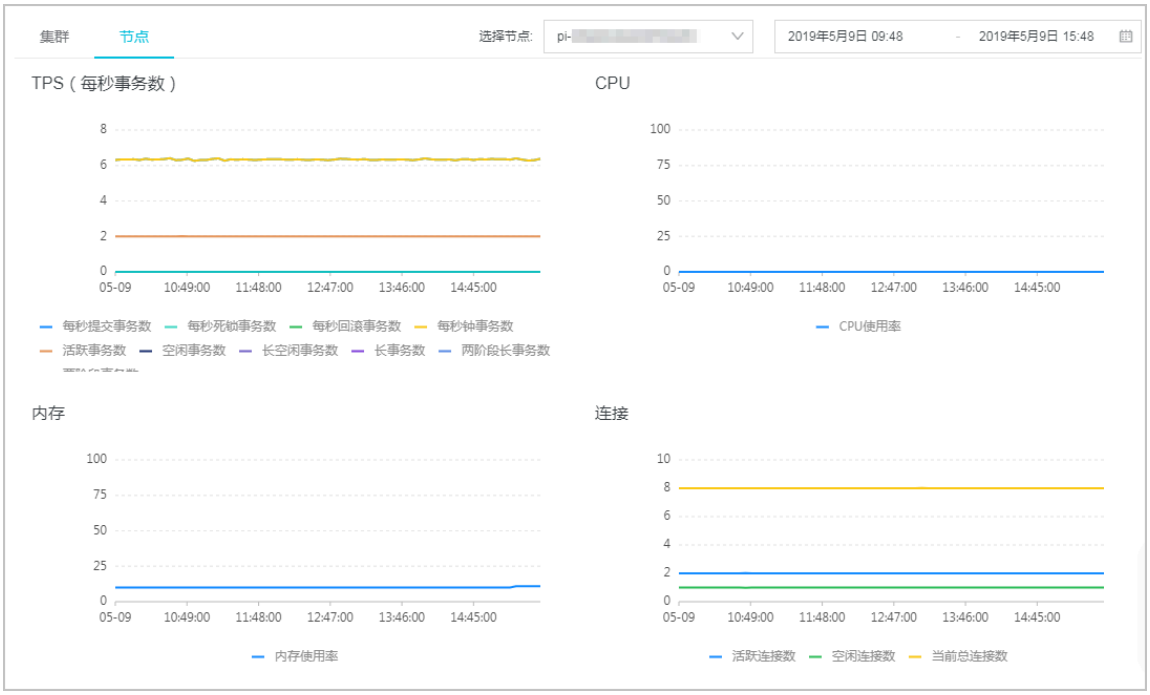
- 集群性能监控：单击**集群**，在右侧设置时间段后单击**确定**。



- 节点性能监控：单击**节点**，在右侧选择节点并设置时间段后单击**确定**。

说明：

节点页面下方单击**显示更多**，会显示更多监控项。



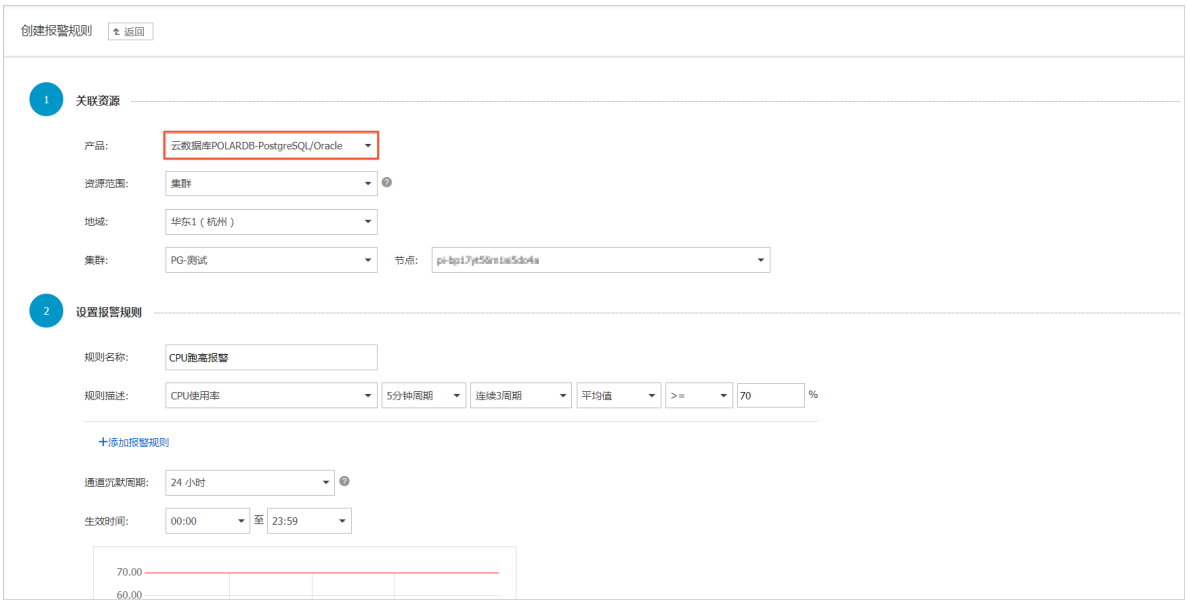
监控项说明

类别	监控项	说明
集群	存储空间	展示数据空间、日志空间、临时空间和WAL日志空间的使用量。
	CPU	展示各节点的CPU使用率。
	内存	展示各节点的内存使用率。
节点	TPS	展示所选择节点的每秒事务数，包括每秒提交事务数、每秒死锁事务数、每秒回滚事务数等等。
	CPU	展示所选择节点的CPU使用率。
	内存	展示所选择节点的内存使用率。
	连接	展示所选择节点的当前总连接数、活跃连接数和空闲连接数。
	扫描行数	展示所选择节点每秒插入、读取、更新、删除、返回的行数。
	数据库最大年龄	数据库最旧和最新的两个事务之间的事务ID差值。
	I/O吞吐量	展示所选择节点的总I/O吞吐量、读I/O吞吐量、写I/O吞吐量。
	IOPS	展示所选择节点的每秒读写次数，包括每秒读写总次数、每秒读次数、每秒写次数。
	缓存	展示所选择节点每秒缓存读取次数和每秒磁盘读取次数。
	缓存命中率	展示所选择节点的缓存命中率。

类别	监控项	说明
	临时文件	展示所选择节点的临时文件数量和总大小。

设置报警

1. 进入云监控控制台。
2. 在左侧导航栏中，选择**报警服务 > 报警规则**。
3. 在**报警规则列表**页面，单击**创建报警规则**，进入**创建报警规则**页面。



4. 在产品下拉列表中，选择**云数据库POLARDB-PostgreSQL/Oracle**，选择资源范围，设置报警规则和通知方式后，单击**确认**即可。

> **说明:**
> 报警规则相关说明，请参见#unique_128。

17.2 性能洞察

性能洞察（Performance Insights）专注于POLARDB集群负载监控、关联分析、性能调优的利器，以简单直观的方式帮助用户迅速评估数据库负载，找到性能问题的源头，提升数据库的稳定性。

典型使用场景

性能洞察可以在以下场景中，为您提供帮助。

- 概要分析集群性能指标。

  帮助您监控集群的关键性能指标，从宏观角度帮助您确认数据库集群负载情况和变化趋势。根据集群关键性能指标趋势图，可以帮助您发现集群负载来源以及负载分布的时间规律。

- 轻松评估数据库负载。

  您无需综合分析复杂繁多的性能指标趋势图，平均活跃会话趋势图中展示了所有核心性能信息，这些信息帮助您轻松地评估数据库负载来源和瓶颈类型，例如是高CPU使用率，还是锁定等待，又或者是I/O延迟等，并且可以直接定位具体是哪些SQL语句。

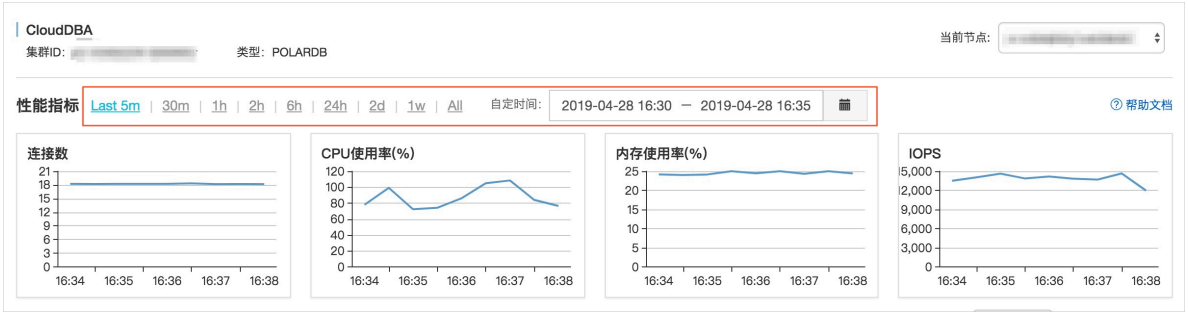
  > **说明:**
  > 平均活跃会话（Average Active Sessions，AAS），是指用户POLARDB集群一段时间内的平均活跃会话数，AAS的数量变化趋势反映了用户POLARDB集群负载的变化情况。因此，性能洞察功能使用AAS来做为POLARDB集群负载高低的衡量指标。

- 简单查找性能问题源头。

  结合AAS趋势图和多维度负载详情进行分析，您可以迅速确定性能问题是集群规格配置导致的，或者是数据库本身设计导致的，并找到是哪些SQL语句导致了性能问题。

操作步骤

1. 登录POLARDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，选择**诊断与优化** > **性能洞察**。
5. 选择过滤条件。

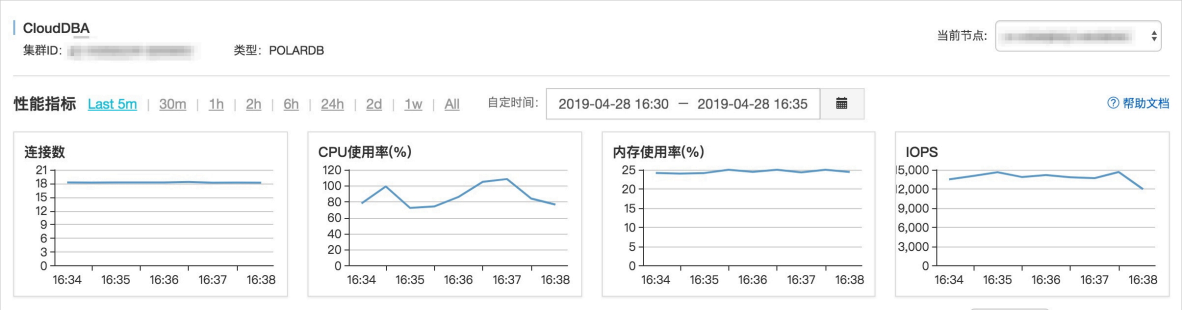


页面介绍

- 关键性能指标趋势图

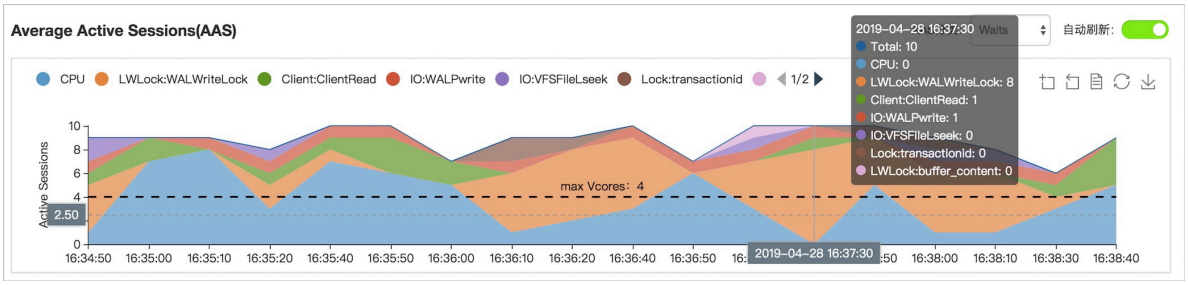
  用户可以通过关键性能指标的趋势图确认集群负载的情况和资源瓶颈。

  您还可以切换时间段或者选择自定义时间范围，来获取相应时间段的关键性能指标趋势图。



- 平均活跃会话（AAS）

  通过关键性能指标的趋势图，宏观确认数据库的负载情况后，可以进一步确认负载来源。



> **说明：**
> max Vcores是指用户POLARDB集群最多可以使用的CPU核数，这个值的大小决定了集群CPU的处理能力。

从实时AAS变化趋势图中，可以清楚地发现POLARDB集群中的负载来源和时间，以及变化规律。

- 多维度负载源详情

  通过分析性能洞察中的实时AAS变化趋势，掌握了集群负载变化的规律，接下来可以从多个维度找出影响性能的具体SQL语句，以及相关联的用户、主机、数据库等。

从以上截图的下半部分，我们可以方便地找出与AAS变化趋势关联负载对应的SQL查询语句，以及每个语句对AAS的使用占比情况。

性能洞察支持6个维度的AAS分类，您可以通过右侧的**AAS分类**下拉框来切换。

类别	说明
SQL	业务TOP 10 SQL的AAS变化趋势。
Waits	活跃会话资源等待的AAS变化趋势。
Users	登录用户的AAS变化趋势。
Hosts	客户端主机名或者主机IP AAS变化趋势。
Databases	业务所在数据库的AAS变化趋势。
Status	活跃会话状态的AAS变化趋势。

18 SQL洞察

SQL洞察功能为您的数据库提供安全审计、性能诊断等增值服务。

费用说明

- 试用版：免费使用，审计日志仅保存一天，即只能查询一天范围内的数据，不支持数据导出等高级功能，不保障数据完整性。
- 30天或以上： 详情请参见#unique_28。

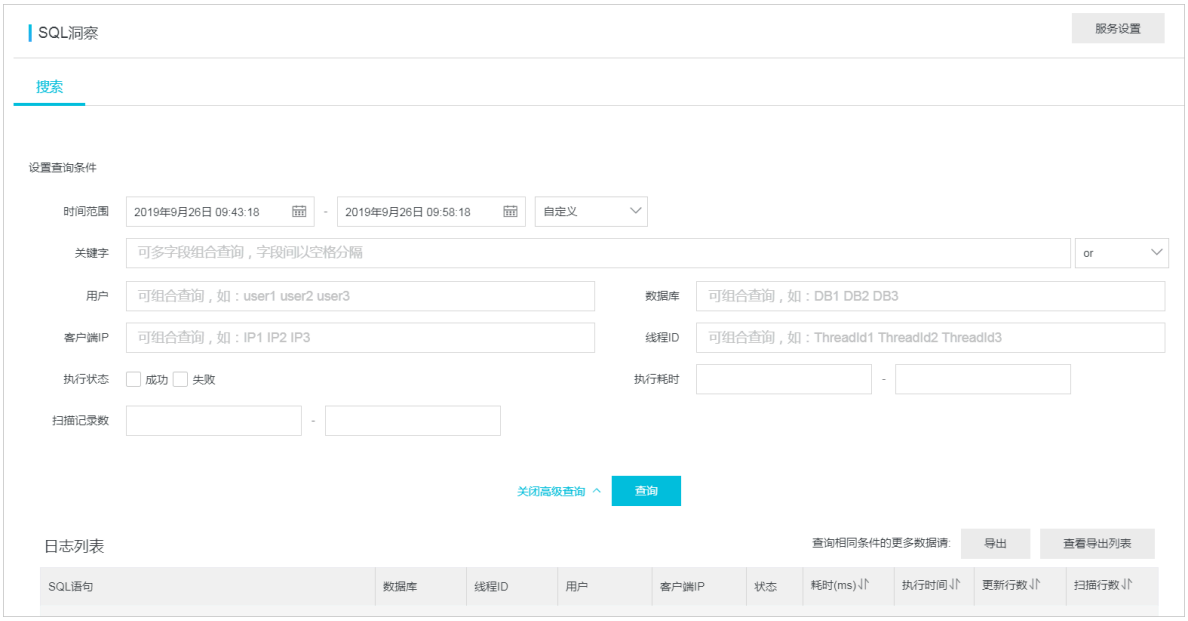
功能说明

- SQL审计日志

  记录对数据库执行的所有操作。通过审计日志记录，您可以对数据库进行故障分析、行为分析、安全审计等操作。

- 增强搜索

  可以按照数据库、用户、客户端IP、线程ID、执行耗时、执行状态等进行多维度检索，并支持导出和下载搜索结果。



开通SQL洞察

1. 登录PolarDB控制台。
2. 在控制台左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，选择**日志与审计 > SQL洞察**。

5. 单击**立即开通**。

6. 选择SQL审计日志的保存时长，单击**开通服务**。

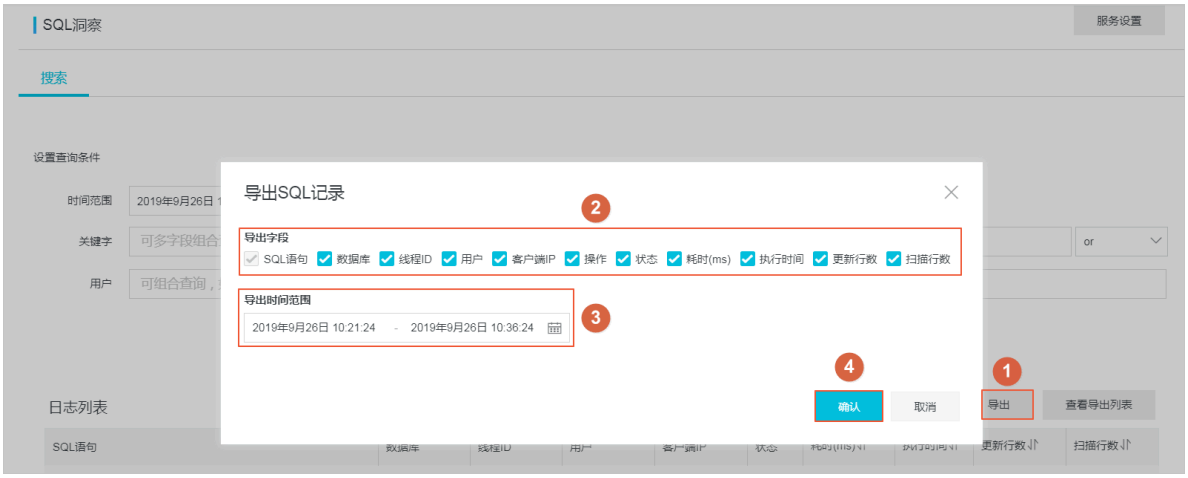
修改SQL日志的存储时长

1. 登录PolarDB控制台。
2. 在控制台左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，选择**日志与审计** > **SQL洞察**。
5. 单击右上角**服务设置**。
6. 修改存储时长，单击**确认**。

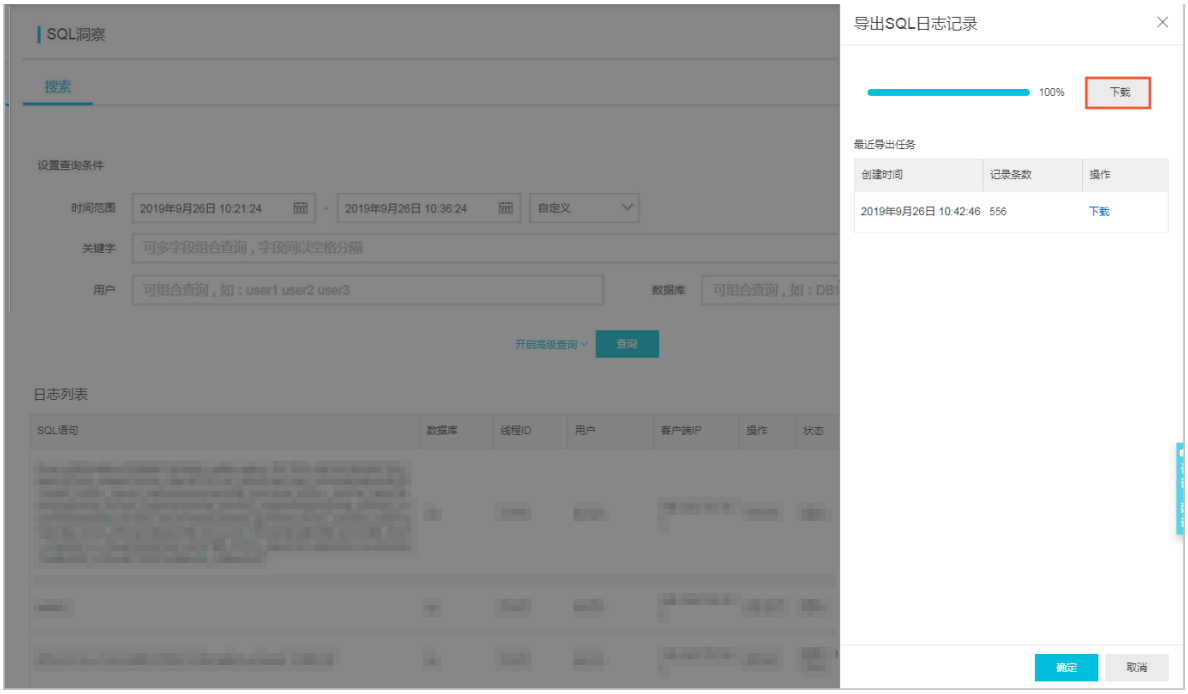
导出SQL记录

1. 登录PolarDB控制台。
2. 在控制台左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，选择**日志与审计** > **SQL洞察**。
5. 单击右侧**导出**。

6. 在弹出的对话框中，选择**导出字段**和**导出时间范围**，单击**确认**。



7. 导出完成后，在**导出SQL日志记录**中，下载已导出的文件并妥善保存。



关闭SQL洞察

> **说明：**
>
> SQL洞察功能关闭后，SQL审计日志会被清空。请将SQL审计日志导出后，再关闭SQL洞察功能。

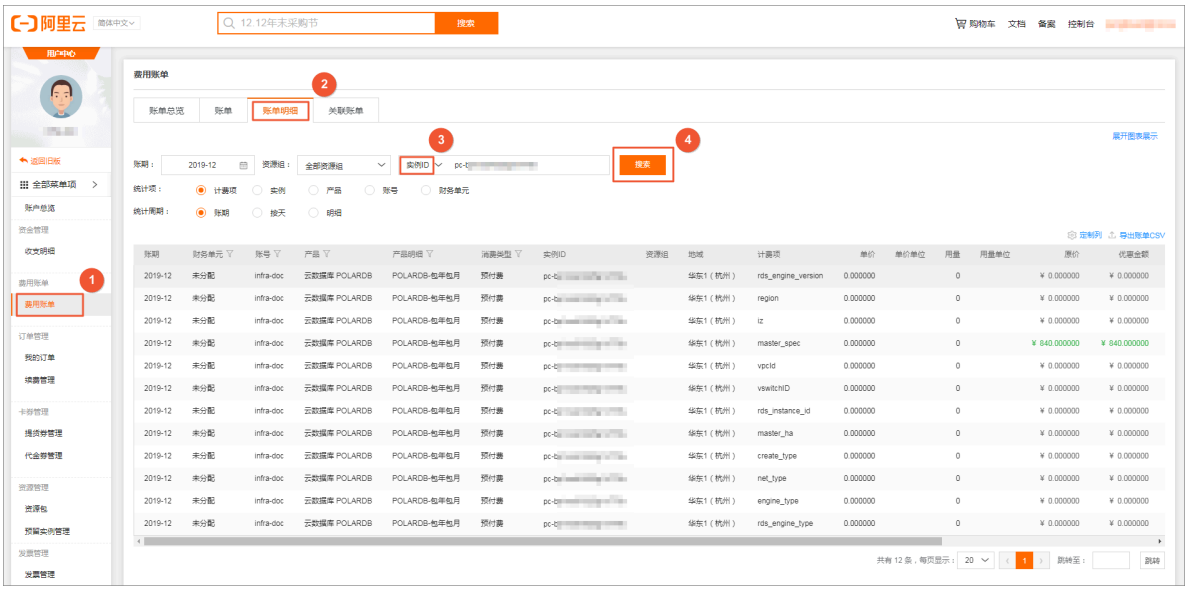
1. 登录PolarDB控制台。
2. 在控制台左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，选择**日志与审计** > **SQL洞察**。
5. 单击右上角**服务设置**。

6. 修改存储时长，单击**确认**。
7. 单击滑块关闭SQL洞察。

查看审计日志的大小和消费明细

1. 登录阿里云管理控制台。
2. 在页面右上角，选择**费用 > 用户中心**。
3. 在左侧导航栏中，选择**费用账单 > 费用账单** 。
4. 选择**账单明细**页签，设置搜索**实例ID**，并搜索目标实例。

> **说明:**
> 若要查询超过12个月前的记录，请提交工单。



5. 查看**计费项**列为**sql_explorer**的费用明细。

19 插件

19.1 使用oss_fdw读写外部数据文本文件

阿里云支持通过oss_fdw插件将OSS中的数据加载到POLARDB for PostgreSQL数据库中，也支持将POLARDB for PostgreSQL数据库中的数据写入OSS中。

oss_fdw 参数

oss_fdw和其他fdw接口一样，对外部数据OSS中的数据进行封装。用户可以像使用数据表一样通过oss_fdw读取OSS中存放的数据。oss_fdw提供独有的参数用于连接和解析OSS上的文件数据。

说明：

- 目前oss_fdw支持读取和写入OSS中文件的格式为：text/csv、gzip格式的text/csv文件。
- oss_fdw各参数的值需使用双引号（""）引起来，且不含无用空格。

CREATE SERVER 参数

- ossendpoint：是内网访问OSS的地址，也称为host。
- id oss：账号id。
- key oss：账号key。
- bucket： OSSBucket，需要先创建OSS账号再设置该参数。

针对导入模式和导出模式，提供下列容错相关参数。网络条件较差时，可以调整以下参数，以保障导入和导出成功。

- oss_connect_timeout：设置链接超时，单位秒，默认是10秒。
- oss_dns_cache_timeout：设置DNS超时，单位秒，默认是60秒。
- oss_speed_limit：设置能容忍的最小速率，默认是1024，即1K。
- oss_speed_time：设置能容忍最小速率的最长时间，默认是15秒。

如果使用了oss_speed_limit和oss_speed_time的默认值，表示如果连续15秒的传输速率小于1K，则超时。

CREATE FOREIGN TABLE参数

- filepath： OSS中带路径的文件名。
  - 文件名包含文件路径，但不包含bucket。
  - 该参数匹配OSS对应路径上的多个文件，支持将多个文件加载到数据库。
  - 文件命名为filepath和filepath.x 支持被导入到数据库，x要求从1开始，且连续。例如，filepath、filepath.1、filepath.2、filepath.3、filepath.5 ，前4个文件会被匹配和导入，但是 filepath.5将无法导入。
- dir：OSS中的虚拟文件目录。
  - dir需要以/结尾。
  - dir指定的虚拟文件目录中的所有文件（不包含子文件夹和子文件夹下的文件）都会被匹配和导入到数据库。
- prefix：指定数据文件对应路径名的前缀，不支持正则表达式，且与 filepath、dir 互斥，三者只能设置其中一个。
- format：指定文件的格式，目前只支持csv。
- encoding：文件中数据的编码格式，支持常见的pg编码，如utf8。
- parse_errors：容错模式解析，以行为单位，忽略文件分析过程中发生的错误。
- delimiter：指定列的分割符。
- quote：指定文件的引用字符。
- escape：指定文件的逃逸字符。
- null：指定匹配对应字符串的列为null，例如null 'test'，即列值为'test'的字符串为null。
- force_not_null：指定某些列的值不为null。例如，force_not_null 'id'表示：如果id列的值为空，则该值为空字符串，而不是null。
- compressiontype：设置读取和写入OSS上文件的格式：
  - none：默认的文件类型，即没有压缩的文本格式。
  - gzip：读取文件的格式为gzip压缩格式。
- compressionlevel：设置写入OSS的压缩格式的压缩等级，范围1到9，默认6。

说明：

- filepath和dir需要在OPTIONS参数中指定。
- filepath和dir必须指定两个参数中的其中一个，且不能同时指定。
- 导出模式目前只支持虚拟文件夹的匹配模式，即只支持dir，不支持filepath。

CREATE FOREIGN TABLE的导出模式参数

- oss_flush_block_size：单次刷出到OSS的buffer大小，默认32MB，可选范围1到128MB。
- oss_file_max_size：写入OSS的最大文件大小，超出之后会切换到另一个文件续写。默认1024MB，可选范围8到4000 MB。
- num_parallel_worker：写OSS数据的压缩模式中并行压缩线程的个数，范围1到8，默认并发数3。

辅助函数

FUNCTION oss_fdw_list_file (relname text, schema text DEFAULT 'public')

- 用于获得某个外部表所匹配的OSS上的文件名和文件的大小。
- 文件大小的单位是字节。

```
select * from oss_fdw_list_file('t_oss');
           name            |  size
-------------------------------+-----------
 oss_test/test.gz.1  | 739698350
 oss_test/test.gz.2  | 739413041
 oss_test/test.gz.3  | 739562048
(3 rows)
```

辅助功能

oss_fdw.rds_read_one_file：在读模式下，指定某个外表匹配的文件。设置后，该外部表在数据导入中只匹配这个被设置的文件。

例如，set oss_fdw.rds_read_one_file = 'oss_test/example16.csv.1';

```
set oss_fdw.rds_read_one_file = 'oss_test/test.gz.2';
select * from oss_fdw_list_file('t_oss');
           name            |  size
-------------------------------+-----------
  oss_test/test.gz.2  | 739413041
(1 rows)
```

oss_fdw用例

```
# 创建插件
create extension oss_fdw;
# 创建 server
CREATE SERVER ossserver FOREIGN DATA WRAPPER oss_fdw OPTIONS
    (host 'oss-cn-hangzhou.aliyuncs.com' ,  id 'xxx',  key 'xxx',  bucket 'mybucket');
# 创建 oss 外部表
CREATE FOREIGN TABLE ossexample
   (date text,  time text,  open float,
    high float,  low float,  volume int)
    SERVER ossserver
    OPTIONS ( filepath 'osstest/example.csv',  delimiter ',' ,
       format 'csv',  encoding 'utf8',  PARSE_ERRORS '100');
# 创建表，数据就装载到这张表中
create table example
      (date text,  time text,  open float,
```

```
        high float,  low float,  volume int);
# 数据从 ossexample 装载到 example 中。
insert into example select * from ossexample;
# 可以看到
# oss_fdw 能够正确估计 oss 上的文件大小，正确的规划查询计划。
explain insert into example select * from ossexample;
                        QUERY PLAN
-----------------------------------------------------------------------
 Insert on example  (cost=0.00..1.60 rows=6 width=92)
   ->  Foreign Scan on ossexample  (cost=0.00..1.60 rows=6 width=92)
       Foreign OssFile: osstest/example.csv.0
       Foreign OssFile Size: 728
(4 rows)
# 表 example 中的数据写出到 OSS 中。
insert into ossexample select * from example;
explain insert into ossexample select * from example;
                     QUERY PLAN
---------------------------------------------------------------
 Insert on ossexample  (cost=0.00..16.60 rows=660 width=92)
   ->  Seq Scan on example  (cost=0.00..16.60 rows=660 width=92)
(2 rows)
```

oss_fdw 注意事项

- oss_fdw是在PostgreSQL FOREIGN TABLE框架下开发的外部表插件。
- 数据导入的性能和POLARDB for PostgreSQL集群的资源（CPU IO MEM MET）相关，也和OSS相关。
- 为保证数据导入的性能，请确保云数据库POLARDB for PostgreSQL与OSS所在Region相同，相关信息请参考OSS endpoint 信息。
- 如果读取外表的SQL时触发 ERROR: oss endpoint userendpoint not in aliyun white list，建议使用阿里云各可用区公共 endpoint。如果问题仍无法解决，请通过工单反馈。

错误处理

导入或导出出错时，日志中会出现下列错误提示信息：

- code：出错请求的HTTP状态码。
- error_code：OSS的错误码。
- error_msg：OSS的错误信息。
- req_id：标识该次请求的UUID。当您无法解决问题时，可以凭req_id来请求OSS开发工程师的帮助。

请参考以下链接中的文档了解和处理各类错误，超时相关的错误可以使用oss_ext相关参数处理。

- OSS help 页面
- PostgreSQL CREATE FOREIGN TABLE 手册
- OSS 错误处理
- OSS 错误响应

id和key隐藏

CREATE SERVER中的id和key信息如果不做任何处理，用户可以使用select * from pg_foreign_server看到明文信息，会暴露用户的id和key。我们通过对id和key进行对称加密实现对id和key的隐藏（不同的实例使用不同的密钥，最大限度保护用户信息），但无法使用类似GP一样的方法，增加一个数据类型，会导致老实例不兼容。

最终的加密后的信息如下：

```
postgres=# select * from pg_foreign_server ;
  srvname  | srvowner | srvfdw | srvtype | srvversion | srvacl |
                srvoptions
-----------+----------+--------+---------+------------+--------
+-------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------
 ossserver |       10 |  16390 |         |            |        | {host=oss-cn-hangzhou-zmf.aliyuncs.com，
id=MD5xxxxxxxx，key=MD5xxxxxxxx，bucket=067862}
```

加密后的信息将会以MD5开头（总长度为len，len%8==3），这样导出之后再导入不会再次加密，但是用户不能创建MD5开头的key和id。

19.2 使用pg_pathman插件

本文介绍pg_pathman插件的一些常见用法。

背景信息

为了提高分区表的性能，PolarDB PostgreSQL引入了pg_pathman插件。该插件一款分区管理插件，提供了分区优化机制。

创建pg_pathman插件扩展

```
test=# create extension pg_pathman;
CREATE EXTENSION
```

查看已安装的扩展

以下命令可以查看已安装的扩展，还可以查看到pg_pathman 的具体版本。

```
test=# \dx
                List of installed extensions
   Name    | Version |   Schema   |         Description
------------+---------+------------+------------------------------
 pg_pathman | 1.5     | public     | Partitioning tool for PostgreSQL
 plpgsql    | 1.0     | pg_catalog | PL/pgSQL procedural language
(2 rows)
```

插件升级

PolarDB PostgreSQL会定期对插件进行升级，以提供更优质的数据库服务。而当您需要升级插件版本时，需要：

- 升级对应集群到最新版本。
- 执行SQL版本更新命令，如下所示：

```
ALTER EXTENSION pg_pathman UPDATE;
SET pg_pathman.enable = t;
```

插件特性

- 目前支持HASH分区、RANGE分区。
- 支持自动分区管理（通过函数接口创建分区，自动将主表数据迁移到分区表），或手工分区管理（通过函数实现，将已有的表绑定到分区表，或者从分区表剥离）。
- 支持的分区字段类型包括int、float、date以及其他常用类型，包括自定义的domain。
- 有效的分区表查询计划（JOINs、subselects 等）。
- 使用RuntimeAppend & RuntimeMergeAppend 自定义计划节点实现了动态分区选择。
- PartitionFilter：一种有效的插入触发器替换方法。
- 支持自动新增分区（目前仅支持RANGE分区表）。
- 支持copy from/to直接读取或写入分区表，提高效率。
- 支持分区字段的更新，需要添加触发器，如果不需要更新分区字段，则不建议添加这个触发器，会产生一定的性能影响。
- 允许用户自定义回调函数，在创建分区时会自动触发。
- 非堵塞式创建分区表，以及后台自动将主表数据非堵塞式迁移到分区表。
- 支持FDW，通过配置参数pg_pathman.insert_into_fdw=(disabled | postgres | any_fdw)支持postgres_fdw或任意FDW。

插件用法

- 相关视图和表
- 分区管理
- 高级分区管理

更多用法，请参见https://github.com/postgrespro/pg_pathman。

相关视图和表

pg_pathman使用函数来维护分区表，并且创建了一些视图，可以查看分区表的状态，具体如下：

1. pathman_config

```
CREATE TABLE IF NOT EXISTS pathman_config (
   partrel       REGCLASS NOT NULL PRIMARY KEY,  -- 主表oid
   attname        TEXT NOT NULL,  -- 分区列名
   parttype       INTEGER NOT NULL,  -- 分区类型(hash or range)
   range_interval  TEXT,  -- range分区的interval
```

```
    CHECK (parttype IN (1, 2)) /* check for allowed part types */ );
```

2. pathman_config_params

```
CREATE TABLE IF NOT EXISTS pathman_config_params (
    partrel       REGCLASS NOT NULL PRIMARY KEY,  -- 主表oid
    enable_parent  BOOLEAN NOT NULL DEFAULT TRUE,  -- 是否在优化器中过滤主表
    auto          BOOLEAN NOT NULL DEFAULT TRUE,  -- insert时是否自动扩展不存在的分区
    init_callback  REGPROCEDURE NOT NULL DEFAULT 0);  -- create partition时的回调函数
oid
```

3. pathman_concurrent_part_tasks

```
-- helper SRF function
CREATE OR REPLACE FUNCTION show_concurrent_part_tasks()
RETURNS TABLE (
    userid     REGROLE,
    pid        INT,
    dbid       OID,
    relid      REGCLASS,
    processed  INT,
    status     TEXT)
AS 'pg_pathman', 'show_concurrent_part_tasks_internal'
LANGUAGE C STRICT;

CREATE OR REPLACE VIEW pathman_concurrent_part_tasks
AS SELECT * FROM show_concurrent_part_tasks();
```

4. pathman_partition_list

```
-- helper SRF function
CREATE OR REPLACE FUNCTION show_partition_list()
RETURNS TABLE (
    parent     REGCLASS,
    partition  REGCLASS,
    parttype   INT4,
    partattr   TEXT,
    range_min  TEXT,
    range_max  TEXT)
AS 'pg_pathman', 'show_partition_list_internal'
LANGUAGE C STRICT;

CREATE OR REPLACE VIEW pathman_partition_list
AS SELECT * FROM show_partition_list();
```

分区管理

1. RANGE分区

   有四个管理函数用来创建范围分区。其中两个可以指定起始值、间隔、分区个数，其函数定义如下：

```
create_range_partitions(relation      REGCLASS,  -- 主表OID
                        attribute     TEXT,      -- 分区列名
                        start_value   ANYELEMENT,  -- 开始值
                        p_interval    ANYELEMENT,  -- 间隔；任意类型，适合任意类型的分区表
                        p_count       INTEGER DEFAULT NULL,   --  分多少个区
                        partition_data BOOLEAN DEFAULT TRUE)   --  是否立即将数据从主表迁移到
分区, 不建议这么使用, 建议使用非堵塞式的迁移( 调用partition_table_concurrently() )
```

```
create_range_partitions(relation      REGCLASS, -- 主表OID
                attribute     TEXT,     -- 分区列名
                start_value    ANYELEMENT, -- 开始值
                p_interval     INTERVAL,   -- 间隔；interval 类型，用于时间分区表
                p_count       INTEGER DEFAULT NULL,   --  分多少个区
                partition_data BOOLEAN DEFAULT TRUE)   --  是否立即将数据从主表迁移到
分区, 不建议这么使用, 建议使用非堵塞式的迁移( 调用partition_table_concurrently() )
```

另外两个可以指定起始值、终值、间隔，其定义如下：

```
create_partitions_from_range(relation      REGCLASS, -- 主表OID
                    attribute     TEXT,     -- 分区列名
                    start_value    ANYELEMENT, -- 开始值
                    end_value      ANYELEMENT, -- 结束值
                    p_interval     ANYELEMENT, -- 间隔；任意类型，适合任意类型的分区表
                    partition_data BOOLEAN DEFAULT TRUE)   --  是否立即将数据从主表迁移
到分区, 不建议这么使用, 建议使用非堵塞式的迁移( 调用partition_table_concurrently() )

create_partitions_from_range(relation      REGCLASS, -- 主表OID
                    attribute     TEXT,     -- 分区列名
                    start_value    ANYELEMENT, -- 开始值
                    end_value      ANYELEMENT, -- 结束值
                    p_interval     INTERVAL,   -- 间隔；interval 类型，用于时间分区表
                    partition_data BOOLEAN DEFAULT TRUE)   --  是否立即将数据从主表迁移
到分区, 不建议这么使用, 建议使用非堵塞式的迁移( 调用partition_table_concurrently() )
```

示例如下所示：

```
创建需要分区的主表
postgres=# create table part_test(id int, info text, crt_time timestamp not null);  -- 分区
列必须有not null约束
CREATE TABLE

插入一批测试数据，模拟已经有数据了的主表
postgres=# insert into part_test select id,md5(random()::text),clock_timestamp() + (id
||' hour')::interval from generate_series(1,10000) t(id);
INSERT 0 10000
postgres=# select * from part_test limit 10;
 id |               info               |          crt_time
----+----------------------------------+----------------------------
  1 | 36fe1adedaa5b848caec4941f87d443a | 2016-10-25 10:27:13.206713
  2 | c7d7358e196a9180efb4d0a10269c889 | 2016-10-25 11:27:13.206893
  3 | 005bdb063550579333264b895df5b75e | 2016-10-25 12:27:13.206904
  4 | 6c900a0fc50c6e4da1ae95447c89dd55 | 2016-10-25 13:27:13.20691
  5 | 857214d8999348ed3cb0469b520dc8e5 | 2016-10-25 14:27:13.206916
  6 | 4495875013e96e625afbf2698124ef5b | 2016-10-25 15:27:13.206921
  7 | 82488cf7e44f87d9b879c70a9ed407d4 | 2016-10-25 16:27:13.20693
  8 | a0b92547c8f17f79814dfbb12b8694a0 | 2016-10-25 17:27:13.206936
  9 | 2ca09e0b85042b476fc235e75326b41b | 2016-10-25 18:27:13.206942
 10 | 7eb762e1ef7dca65faf413f236dff93d | 2016-10-25 19:27:13.206947
(10 rows)

注意:
1. 分区列必须有not null约束
2. 分区个数必须能覆盖已有的所有记录

创建分区，每个分区包含1个月的跨度数据
postgres=# select
create_range_partitions('part_test'::regclass,           -- 主表OID
                'crt_time',                     -- 分区列名
                '2016-10-25 00:00:00'::timestamp,  -- 开始值
                interval '1 month',             -- 间隔；interval 类型，用于时间分区表
```

```
                  24,                           -- 分多少个区
                  false) ;                       -- 不迁移数据
NOTICE:  sequence "part_test_seq" does not exist, skipping
 create_range_partitions
-------------------------
                      24
(1 row)
postgres-# \d+ part_test
                              Table "public.part_test"
  Column  |            Type             | Modifiers | Storage  | Stats target | Description
----------+-----------------------------+-----------+----------+--------------
+-------------
 id       | integer                     |           | plain    |              |
 info     | text                        |           | extended |              |
 crt_time | timestamp without time zone | not null  | plain    |              |
Child tables: part_test_1,
              part_test_10,
              part_test_11,
              part_test_12,
              part_test_13,
              part_test_14,
              part_test_15,
              part_test_16,
              part_test_17,
              part_test_18,
              part_test_19,
              part_test_2,
              part_test_20,
              part_test_21,
              part_test_22,
              part_test_23,
              part_test_24,
              part_test_3,
              part_test_4,
              part_test_5,
              part_test_6,
              part_test_7,
              part_test_8,
              part_test_9


由于不迁移数据，所以数据还在主表
postgres=# select count(*) from only part_test;
 count
-------
 10000
(1 row)


使用非堵塞式的迁移接口
partition_table_concurrently(relation   REGCLASS,            -- 主表OID
                        batch_size INTEGER DEFAULT 1000,  -- 一个事务批量迁移多少记录
                        sleep_time FLOAT8 DEFAULT 1.0)    -- 获得行锁失败时，休眠多久再次获
取，重试60次退出任务。


postgres=# select partition_table_concurrently('part_test'::regclass,
                        10000,
                        1.0);
NOTICE:  worker started, you can stop it with the following command: select
stop_concurrent_part_task('part_test');
 partition_table_concurrently
------------------------------
```

```
(1 row)


迁移结束后，主表数据已经没有了，全部在分区中
postgres=# select count(*) from only part_test;
 count
-------
     0
(1 row)


数据迁移完成后，建议禁用主表，这样执行计划就不会出现主表了
postgres=# select set_enable_parent('part_test'::regclass, false);
 set_enable_parent
-------------------

(1 row)

postgres=# explain select * from part_test where crt_time = '2016-10-25 00:00:00'::
timestamp;
                            QUERY PLAN
------------------------------------------------------------------------------------
 Append  (cost=0.00..16.18 rows=1 width=45)
   ->  Seq Scan on part_test_1  (cost=0.00..16.18 rows=1 width=45)
         Filter: (crt_time = '2016-10-25 00:00:00'::timestamp without time zone)
(3 rows)
```

说明:

在RANGE分区表使用过程中，建议您：

- 分区列必须有not null约束。
- 分区个数必须能覆盖已有的所有记录。
- 使用非堵塞式迁移接口。
- 数据迁移完成后，禁用主表。

2. HASH分区

有一个管理函数用来创建范围分区，可以指定起始值、间隔、分区个数，具体如下：

```
create_hash_partitions(relation       REGCLASS,  -- 主表OID
                attribute       TEXT,     -- 分区列名
                partitions_count INTEGER,   -- 打算创建多少个分区
                partition_data   BOOLEAN DEFAULT TRUE)   --  是否立即将数据从主表迁移到
分区, 不建议这么使用, 建议使用非堵塞式的迁移( 调用partition_table_concurrently() )
```

示例如下所示：

```
创建需要分区的主表
postgres=# create table part_test(id int, info text, crt_time timestamp not null);    -- 分
区列必须有not null约束
CREATE TABLE

插入一批测试数据，模拟已经有数据了的主表
postgres=# insert into part_test select id,md5(random()::text),clock_timestamp() + (id
||' hour')::interval from generate_series(1,10000) t(id);
INSERT 0 10000
```

```
postgres=# select * from part_test limit 10;
 id |              info              |         crt_time
----+--------------------------------+---------------------------
  1 | 29ce4edc70dbfbe78912beb7c4cc95c2 | 2016-10-25 10:47:32.873879
  2 | e0990a6fb5826409667c9eb150fef386 | 2016-10-25 11:47:32.874048
  3 | d25f577a01013925c203910e34470695 | 2016-10-25 12:47:32.874059
  4 | 501419c3f7c218e562b324a1bebfe0ad | 2016-10-25 13:47:32.874065
  5 | 5e5e22bdf110d66a5224a657955ba158 | 2016-10-25 14:47:32.87407
  6 | 55d2d4fd5229a6595e0dd56e13d32be4 | 2016-10-25 15:47:32.874076
  7 | 1dfb9a783af55b123c7a888afe1eb950 | 2016-10-25 16:47:32.874081
  8 | 41eeb0bf395a4ab1e08691125ae74bff | 2016-10-25 17:47:32.874087
  9 | 83783d69cc4f9bb41a3978fe9e13d7fa | 2016-10-25 18:47:32.874092
 10 | affc9406d5b3412ae31f7d7283cda0dd | 2016-10-25 19:47:32.874097
(10 rows)

注意:
1. 分区列必须有not null约束

创建128个分区
postgres=# select
create_hash_partitions('part_test'::regclass,            -- 主表OID
                    'crt_time',                    -- 分区列名
                    128,                         -- 打算创建多少个分区
                    false) ;                      -- 不迁移数据
 create_hash_partitions
------------------------
                   128
(1 row)

postgres=# \d+ part_test
                          Table "public.part_test"
  Column  |          Type           | Modifiers | Storage  | Stats target | Description
----------+-------------------------------+-----------+----------+--------------
+-------------
 id      | integer                  |         | plain    |            |
 info    | text                     |         | extended |            |
 crt_time | timestamp without time zone | not null  | plain    |            |
Child tables: part_test_0,
              part_test_1,
              part_test_10,
              part_test_100,
              part_test_101,
              part_test_102,
              part_test_103,
              part_test_104,
              part_test_105,
              part_test_106,
              part_test_107,
              part_test_108,
              part_test_109,
              part_test_11,
              part_test_110,
              part_test_111,
              part_test_112,
              part_test_113,
              part_test_114,
              part_test_115,
              part_test_116,
              part_test_117,
              part_test_118,
              part_test_119,
              part_test_12,
              part_test_120,
              part_test_121,
```

```
        part_test_122,
        part_test_123,
        part_test_124,
        part_test_125,
        part_test_126,
        part_test_127,
        part_test_13,
        part_test_14,
        part_test_15,
        part_test_16,
        part_test_17,
        part_test_18,
        part_test_19,
        part_test_2,
        part_test_20,
        part_test_21,
        part_test_22,
        part_test_23,
        part_test_24,
        part_test_25,
        part_test_26,
        part_test_27,
        part_test_28,
        part_test_29,
        part_test_3,
        part_test_30,
        part_test_31,
        part_test_32,
        part_test_33,
        part_test_34,
        part_test_35,
        part_test_36,
        part_test_37,
        part_test_38,
        part_test_39,
        part_test_4,
        part_test_40,
        part_test_41,
        part_test_42,
        part_test_43,
        part_test_44,
        part_test_45,
        part_test_46,
        part_test_47,
        part_test_48,
        part_test_49,
        part_test_5,
        part_test_50,
        part_test_51,
        part_test_52,
        part_test_53,
        part_test_54,
        part_test_55,
        part_test_56,
        part_test_57,
        part_test_58,
        part_test_59,
        part_test_6,
        part_test_60,
        part_test_61,
        part_test_62,
        part_test_63,
        part_test_64,
        part_test_65,
```

```
          part_test_66,
          part_test_67,
          part_test_68,
          part_test_69,
          part_test_7,
          part_test_70,
          part_test_71,
          part_test_72,
          part_test_73,
          part_test_74,
          part_test_75,
          part_test_76,
          part_test_77,
          part_test_78,
          part_test_79,
          part_test_8,
          part_test_80,
          part_test_81,
          part_test_82,
          part_test_83,
          part_test_84,
          part_test_85,
          part_test_86,
          part_test_87,
          part_test_88,
          part_test_89,
          part_test_9,
          part_test_90,
          part_test_91,
          part_test_92,
          part_test_93,
          part_test_94,
          part_test_95,
          part_test_96,
          part_test_97,
          part_test_98,
          part_test_99


由于不迁移数据，所以数据还在主表
postgres=# select count(*) from only part_test;
 count
-------
 10000
(1 row)


使用非堵塞式的迁移接口
partition_table_concurrently(relation   REGCLASS,           -- 主表OID
                    batch_size INTEGER DEFAULT 1000,  -- 一个事务批量迁移多少记录
                    sleep_time FLOAT8 DEFAULT 1.0)    -- 获得行锁失败时，休眠多久再次获
取，重试60次退出任务。


postgres=# select partition_table_concurrently('part_test'::regclass,
                    10000,
                    1.0);
NOTICE:  worker started, you can stop it with the following command: select
stop_concurrent_part_task('part_test');
 partition_table_concurrently
------------------------------

(1 row)
```

```
迁移结束后，主表数据已经没有了，全部在分区中
postgres=# select count(*) from only part_test;
 count
-------
     0
(1 row)


数据迁移完成后，建议禁用主表，这样执行计划就不会出现主表了
postgres=# select set_enable_parent('part_test'::regclass, false);
 set_enable_parent
-------------------

(1 row)

只查单个分区
postgres=# explain select * from part_test where crt_time = '2016-10-25 00:00:00'::
timestamp;
                            QUERY PLAN
------------------------------------------------------------------------------------
 Append  (cost=0.00..1.91 rows=1 width=45)
   ->  Seq Scan on part_test_122  (cost=0.00..1.91 rows=1 width=45)
         Filter: (crt_time = '2016-10-25 00:00:00'::timestamp without time zone)
(3 rows)

分区表约束如下
很显然pg_pathman自动完成了转换，如果是传统的继承，select * from part_test where
crt_time = '2016-10-25 00:00:00'::timestamp; 这种写法是不能筛选分区的。
postgres=# \d+ part_test_122
                        Table "public.part_test_122"
  Column  |            Type             | Modifiers | Storage  | Stats target | Description
----------+-----------------------------+-----------+----------+--------------
+-------------
 id       | integer                     |           | plain    |              |
 info     | text                        |           | extended |              |
 crt_time | timestamp without time zone | not null  | plain    |              |
Check constraints:
    "pathman_part_test_122_3_check" CHECK (get_hash_part_idx(timestamp_hash(
crt_time), 128) = 122)
Inherits: part_test
```

说明:

在HASH分区表使用过程中，建议您:

- 分区列必须有not null约束。
- 使用非堵塞式迁移接口。
- 数据迁移完成后，禁用主表。
- pg_pathman不会受制于表达式的写法，所以`select * from part_test where crt_time = '2016-10-25 00:00:00'::timestamp;`这样的写法也能用于HASH分区的。
- HASH分区列不局限于int类型的列，会使用HASH函数自动转换。

3. 数据迁移到分区

   如果创建分区表时，未将主表数据迁移到分区，那么可以使用非堵塞式的迁移接口，将数据迁移到分区。用法如下：

   ```
   with tmp as (delete from 主表 limit xx nowait returning *) insert into 分区 select * from tmp

   或者使用 select array_agg(ctid) from 主表 limit xx for update nowati 进行标示 然后执行delete和insert。
   ```

   函数接口如下：

   ```
   partition_table_concurrently(relation   REGCLASS,           -- 主表OID
                       batch_size INTEGER DEFAULT 1000,  -- 一个事务批量迁移多少记录
                       sleep_time FLOAT8 DEFAULT 1.0)    -- 获得行锁失败时，休眠多久再次获取，重试60次退出任务。
   ```

   示例如下所示：

   ```
   postgres=# select partition_table_concurrently('part_test'::regclass,
                       10000,
                       1.0);
   NOTICE:  worker started, you can stop it with the following command: select stop_concurrent_part_task('part_test');
    partition_table_concurrently
   ------------------------------

   (1 row)
   ```

   如果停止迁移任务，调用如下函数接口：

   ```
   stop_concurrent_part_task(relation REGCLASS)
   ```

   查看后台的数据迁移任务。

   ```
   postgres=# select * from pathman_concurrent_part_tasks;
    userid | pid | dbid | relid | processed | status
   --------+-----+------+-------+-----------+--------
   (0 rows)
   ```

4. 分裂范围分区

   如果某个分区太大，想分裂为两个分区，可以使用如下方法（目前仅支持RANGE分区表）：

   ```
   split_range_partition(partition     REGCLASS,          -- 分区oid
                   split_value    ANYELEMENT,        -- 分裂值
                   partition_name TEXT DEFAULT NULL)   -- 分裂后新增的分区表名
   ```

   示例如下所示：

   ```
   postgres=# \d+ part_test
                           Table "public.part_test"
    Column |         Type          | Modifiers | Storage  | Stats target | Description
   ----------+---------------------------+-----------+----------+--------------
   +-------------
   ```

```
 id      | integer                  |         | plain    |             |
 info    | text                     |         | extended |             |
 crt_time | timestamp without time zone | not null | plain    |             |
Child tables: part_test_1,
              part_test_10,
              part_test_11,
              part_test_12,
              part_test_13,
              part_test_14,
              part_test_15,
              part_test_16,
              part_test_17,
              part_test_18,
              part_test_19,
              part_test_2,
              part_test_20,
              part_test_21,
              part_test_22,
              part_test_23,
              part_test_24,
              part_test_3,
              part_test_4,
              part_test_5,
              part_test_6,
              part_test_7,
              part_test_8,
              part_test_9

postgres=# \d+ part_test_1
                          Table "public.part_test_1"
  Column  |          Type           | Modifiers | Storage  | Stats target | Description
----------+-----------------------------+-----------+----------+--------------
+-------------
 id      | integer                  |         | plain    |             |
 info    | text                     |         | extended |             |
 crt_time | timestamp without time zone | not null | plain    |             |
Check constraints:
    "pathman_part_test_1_3_check" CHECK (crt_time >= '2016-10-25 00:00:00'::
timestamp without time zone AND crt_time < '2016-11-25 00:00:00'::timestamp
without time zone)
Inherits: part_test
```

分裂

```
postgres=# select split_range_partition('part_test_1'::regclass,            -- 分区oid
                  '2016-11-10 00:00:00'::timestamp,     -- 分裂值
                  'part_test_1_2');                     -- 分区表名
           split_range_partition
-----------------------------------------------
 {"2016-10-25 00:00:00","2016-11-25 00:00:00"}
(1 row)
```

分裂后的两个表如下：

```
postgres=# \d+ part_test_1
                          Table "public.part_test_1"
  Column  |          Type           | Modifiers | Storage  | Stats target | Description
----------+-----------------------------+-----------+----------+--------------
+-------------
 id      | integer                  |         | plain    |             |
 info    | text                     |         | extended |             |
 crt_time | timestamp without time zone | not null | plain    |             |
```

```
Check constraints:
    "pathman_part_test_1_3_check" CHECK (crt_time >= '2016-10-25 00:00:00'::
timestamp without time zone AND crt_time < '2016-11-10 00:00:00'::timestamp
without time zone)
Inherits: part_test

postgres=# \d+ part_test_1_2
                          Table "public.part_test_1_2"
  Column  |          Type           | Modifiers | Storage  | Stats target | Description
----------+-------------------------------+-----------+----------+--------------
+-------------
 id       | integer                 |           | plain    |              |
 info     | text                    |           | extended |              |
 crt_time | timestamp without time zone | not null  | plain    |              |
Check constraints:
    "pathman_part_test_1_2_3_check" CHECK (crt_time >= '2016-11-10 00:00:00'::
timestamp without time zone AND crt_time < '2016-11-25 00:00:00'::timestamp
without time zone)
Inherits: part_test
```

数据会自动迁移到另一个分区。

```
postgres=# select count(*) from part_test_1;
 count
-------
   373
(1 row)

postgres=# select count(*) from part_test_1_2;
 count
-------
   360
(1 row)
```

继承关系如下：

```
postgres=# \d+ part_test
                            Table "public.part_test"
  Column  |          Type           | Modifiers | Storage  | Stats target | Description
----------+-------------------------------+-----------+----------+--------------
+-------------
 id       | integer                 |           | plain    |              |
 info     | text                    |           | extended |              |
 crt_time | timestamp without time zone | not null  | plain    |              |
Child tables: part_test_1,
              part_test_10,
              part_test_11,
              part_test_12,
              part_test_13,
              part_test_14,
              part_test_15,
              part_test_16,
              part_test_17,
              part_test_18,
              part_test_19,
              part_test_1_2,   -- 新增的表
              part_test_2,
              part_test_20,
              part_test_21,
              part_test_22,
              part_test_23,
              part_test_24,
```

```
            part_test_3,
            part_test_4,
            part_test_5,
            part_test_6,
            part_test_7,
            part_test_8,
            part_test_9
```

5. 合并范围分区

   目前仅支持RANGE 分区，调用如下接口：

```
指定两个需要合并分区，必须为相邻分区
merge_range_partitions(partition1 REGCLASS, partition2 REGCLASS)
```

   示例如下所示：

```
postgres=# select merge_range_partitions('part_test_2'::regclass, 'part_test_12'::
regclass) ;
ERROR:  merge failed, partitions must be adjacent
CONTEXT:  PL/pgSQL function merge_range_partitions_internal(regclass,regclass,
regclass,anyelement) line 27 at RAISE
SQL statement "SELECT public.merge_range_partitions_internal($1, $2, $3, NULL::
timestamp without time zone)"
PL/pgSQL function merge_range_partitions(regclass,regclass) line 44 at EXECUTE
不是相邻分区，报错

相邻分区可以合并
postgres=# select merge_range_partitions('part_test_1'::regclass, 'part_test_1_2'::
regclass) ;
 merge_range_partitions
------------------------

(1 row)
```

   合并后，会删掉其中一个分区表。

```
postgres=# \d part_test_1_2
Did not find any relation named "part_test_1_2".

postgres=# \d part_test_1
           Table "public.part_test_1"
  Column |            Type             | Modifiers
---------+-----------------------------+-----------
 id      | integer                     |
 info    | text                        |
 crt_time | timestamp without time zone | not null
Check constraints:
    "pathman_part_test_1_3_check" CHECK (crt_time >= '2016-10-25 00:00:00'::
timestamp without time zone AND crt_time < '2016-11-25 00:00:00'::timestamp
without time zone)
Inherits: part_test

postgres=# select count(*) from part_test_1;
 count
-------
   733
```

```
(1 row)
```

6. 向后添加范围分区

   如果已经对主表进行了分区，将来需要增加分区的话，有几种方法，一种是向后新增分区（即在末尾追加分区）。

   新增分区时，会使用初次创建该分区表时的interval作为间隔。可以在pathman_config中查询每个分区表初次创建时的interval，如下：

```
postgres=# select * from pathman_config;
 partrel | attname | parttype | range_interval
----------+----------+----------+----------------
 part_test | crt_time |       2 | 1 mon
(1 row)
```

   添加分区接口（目前不支持指定表空间）

```
append_range_partition(parent        REGCLASS,          -- 主表OID
                       partition_name TEXT DEFAULT NULL,  -- 新增的分区表名, 默认不需要输入
                       tablespace     TEXT DEFAULT NULL)  -- 新增的分区表放到哪个表空间, 默认不
需要输入
```

   示例如下所示：

```
postgres=# select append_range_partition('part_test'::regclass);
 append_range_partition
------------------------
 public.part_test_25
(1 row)

postgres=# \d+ part_test_25
                          Table "public.part_test_25"
 Column |            Type             | Modifiers | Storage  | Stats target | Description
----------+-----------------------------+-----------+----------+--------------
+-------------
 id      | integer                     |           | plain    |              |
 info    | text                        |           | extended |              |
 crt_time | timestamp without time zone | not null  | plain    |              |
Check constraints:
    "pathman_part_test_25_3_check" CHECK (crt_time >= '2018-10-25 00:00:00'::
timestamp without time zone AND crt_time < '2018-11-25 00:00:00'::timestamp
without time zone)
Inherits: part_test

postgres=# \d+ part_test_24
                          Table "public.part_test_24"
 Column |            Type             | Modifiers | Storage  | Stats target | Description
----------+-----------------------------+-----------+----------+--------------
+-------------
 id      | integer                     |           | plain    |              |
 info    | text                        |           | extended |              |
 crt_time | timestamp without time zone | not null  | plain    |              |
Check constraints:
    "pathman_part_test_24_3_check" CHECK (crt_time >= '2018-09-25 00:00:00'::
timestamp without time zone AND crt_time < '2018-10-25 00:00:00'::timestamp
without time zone)
```

```
Inherits: part_test
```

7. 向前添加范围分区

   在头部追加分区，接口如下：

```
prepend_range_partition(parent        REGCLASS,
                        partition_name TEXT DEFAULT NULL,
                        tablespace     TEXT DEFAULT NULL)
```

   示例如下所示：

```
postgres=# select prepend_range_partition('part_test'::regclass);
 prepend_range_partition
-------------------------
 public.part_test_26
(1 row)

postgres=# \d+ part_test_26
                          Table "public.part_test_26"
  Column  |          Type           | Modifiers | Storage  | Stats target | Description
----------+-----------------------------+-----------+----------+--------------
+-------------
 id       | integer                 |           | plain    |              |
 info     | text                    |           | extended |              |
 crt_time | timestamp without time zone | not null  | plain    |              |
Check constraints:
    "pathman_part_test_26_3_check" CHECK (crt_time >= '2016-09-25 00:00:00'::
timestamp without time zone AND crt_time < '2016-10-25 00:00:00'::timestamp
without time zone)
Inherits: part_test

postgres=# \d+ part_test_1
                          Table "public.part_test_1"
  Column  |          Type           | Modifiers | Storage  | Stats target | Description
----------+-----------------------------+-----------+----------+--------------
+-------------
 id       | integer                 |           | plain    |              |
 info     | text                    |           | extended |              |
 crt_time | timestamp without time zone | not null  | plain    |              |
Check constraints:
    "pathman_part_test_1_3_check" CHECK (crt_time >= '2016-10-25 00:00:00'::
timestamp without time zone AND crt_time < '2016-11-25 00:00:00'::timestamp
without time zone)
Inherits: part_test
```

8. 添加分区

   指定分区起始值的方式添加分区，只要创建的分区和已有分区不会存在数据交叉就可以创建成功。也就是说使用这种方法，不要求强制创建连续的分区，例如已有分区覆盖了2010-2015的范围，您可以直接创建一个2020年的分区表，不需要覆盖2015到2020的范围。接口如下：

```
add_range_partition(relation      REGCLASS,  -- 主表OID
                    start_value   ANYELEMENT, -- 起始值
                    end_value     ANYELEMENT, -- 结束值
                    partition_name TEXT DEFAULT NULL, -- 分区名
```

```
                tablespace     TEXT DEFAULT NULL)  -- 分区创建在哪个表空间下
```

示例如下所示：

```
postgres=# select add_range_partition('part_test'::regclass,    -- 主表OID
                '2020-01-01 00:00:00'::timestamp，-- 起始值
                '2020-02-01 00:00:00'::timestamp); -- 结束值
 add_range_partition
---------------------
 public.part_test_27
(1 row)

postgres=# \d+ part_test_27
                        Table "public.part_test_27"
  Column  |          Type           | Modifiers | Storage  | Stats target | Description
----------+-----------------------------+-----------+----------+--------------
+-------------
 id       | integer                 |           | plain    |              |
 info     | text                    |           | extended |              |
 crt_time | timestamp without time zone | not null  | plain    |              |
Check constraints:
    "pathman_part_test_27_3_check" CHECK (crt_time >= '2020-01-01 00:00:00'::
timestamp without time zone AND crt_time < '2020-02-01 00:00:00'::timestamp
without time zone)
Inherits: part_test
```

9. 删除分区

删除单个范围分区，接口如下：

```
drop_range_partition(partition TEXT,  -- 分区名称
                delete_data BOOLEAN DEFAULT TRUE)  -- 是否删除分区数据，如果false，表示
分区数据迁移到主表。

Drop RANGE partition and all of its data if delete_data is true.
```

示例如下所示：

```
删除分区， 数据迁移到主表
postgres=# select drop_range_partition('part_test_1',false);
NOTICE:  733 rows copied from part_test_1
 drop_range_partition
----------------------
 part_test_1
(1 row)

postgres=# select drop_range_partition('part_test_2',false);
NOTICE:  720 rows copied from part_test_2
 drop_range_partition
----------------------
 part_test_2
(1 row)

postgres=# select count(*) from part_test;
 count
-------
 10000
(1 row)

删除分区，分区数据也删除，不迁移到主表
postgres=# select drop_range_partition('part_test_3',true);
```

```
 drop_range_partition
----------------------
 part_test_3
(1 row)

postgres=# select count(*) from part_test;
 count
-------
  9256
(1 row)

postgres=# select count(*) from only part_test;
 count
-------
  1453
(1 row)
```

删除所有分区，并且指定是否要将数据迁移到主表。接口如下：

```
drop_partitions(parent      REGCLASS,
                delete_data BOOLEAN DEFAULT FALSE)

Drop partitions of the parent table (both foreign and local relations).
If delete_data is false, the data is copied to the parent table first.
Default is false.
```

示例如下所示：

```
postgres=# select drop_partitions('part_test'::regclass, false);  -- 删除所有分区表，并将数据迁移到主表
NOTICE:  function public.part_test_upd_trig_func() does not exist, skipping
NOTICE:  744 rows copied from part_test_4
NOTICE:  672 rows copied from part_test_5
NOTICE:  744 rows copied from part_test_6
NOTICE:  720 rows copied from part_test_7
NOTICE:  744 rows copied from part_test_8
NOTICE:  720 rows copied from part_test_9
NOTICE:  744 rows copied from part_test_10
NOTICE:  744 rows copied from part_test_11
NOTICE:  720 rows copied from part_test_12
NOTICE:  744 rows copied from part_test_13
NOTICE:  507 rows copied from part_test_14
NOTICE:  0 rows copied from part_test_15
NOTICE:  0 rows copied from part_test_16
NOTICE:  0 rows copied from part_test_17
NOTICE:  0 rows copied from part_test_18
NOTICE:  0 rows copied from part_test_19
NOTICE:  0 rows copied from part_test_20
NOTICE:  0 rows copied from part_test_21
NOTICE:  0 rows copied from part_test_22
NOTICE:  0 rows copied from part_test_23
NOTICE:  0 rows copied from part_test_24
NOTICE:  0 rows copied from part_test_25
NOTICE:  0 rows copied from part_test_26
NOTICE:  0 rows copied from part_test_27
 drop_partitions
-----------------
              24
(1 row)

postgres=# select count(*) from part_test;
 count
```

```
-------
 9256
(1 row)

postgres=# \dt part_test_4
No matching relations found.
```

10.绑定分区（已有的表加入分区表）

将已有的表，绑定到已有的某个分区主表。已有的表与主表要保持一致的结构，包括dropped columns （查看pg_attribute的一致性）。接口如下：

```
attach_range_partition(relation    REGCLASS,   -- 主表OID
                       partition   REGCLASS,   -- 分区表OID
                       start_value ANYELEMENT, -- 起始值
                       end_value   ANYELEMENT) -- 结束值
```

示例如下所示：

```
postgres=# create table part_test_1 (like part_test including all);
CREATE TABLE
postgres=# \d+ part_test
                          Table "public.part_test"
  Column  |            Type             | Modifiers | Storage  | Stats target | Description
----------+-----------------------------+-----------+----------+--------------
+-------------
 id       | integer                     |           | plain    |              |
 info     | text                        |           | extended |              |
 crt_time | timestamp without time zone | not null  | plain    |              |

postgres=# \d+ part_test_1
                        Table "public.part_test_1"
  Column  |            Type             | Modifiers | Storage  | Stats target | Description
----------+-----------------------------+-----------+----------+--------------
+-------------
 id       | integer                     |           | plain    |              |
 info     | text                        |           | extended |              |
 crt_time | timestamp without time zone | not null  | plain    |              |

postgres=# select attach_range_partition('part_test'::regclass, 'part_test_1'::regclass,
 '2019-01-01 00:00:00'::timestamp, '2019-02-01 00:00:00'::timestamp);
 attach_range_partition
------------------------
 part_test_1
(1 row)

绑定分区时，
自动创建继承关系，自动创建约束
postgres=# \d+ part_test_1
                        Table "public.part_test_1"
  Column  |            Type             | Modifiers | Storage  | Stats target | Description
----------+-----------------------------+-----------+----------+--------------
+-------------
 id       | integer                     |           | plain    |              |
 info     | text                        |           | extended |              |
 crt_time | timestamp without time zone | not null  | plain    |              |
Check constraints:
    "pathman_part_test_1_3_check" CHECK (crt_time >= '2019-01-01 00:00:00'::
timestamp without time zone AND crt_time < '2019-02-01 00:00:00'::timestamp
without time zone)
```

```
Inherits: part_test
```

11.解绑分区（将分区变成普通表）

将分区从主表的继承关系中删除，不删数据，删除继承关系，删除约束。接口如下：

```
detach_range_partition(partition REGCLASS) -- 指定分区名，转换为普通表
```

示例如下所示：

```
postgres=# select count(*) from part_test;
 count
-------
  9256
(1 row)

postgres=# select count(*) from part_test_2;
 count
-------
   733
(1 row)

postgres=# select detach_range_partition('part_test_2');
 detach_range_partition
------------------------
 part_test_2
(1 row)

postgres=# select count(*) from part_test_2;
 count
-------
   733
(1 row)

postgres=# select count(*) from part_test;
 count
-------
  8523
(1 row)
```

12.永久禁止分区表pg_pathman插件

您可以针对单个分区主表禁用pg_pathman。接口函数如下：

```
disable_pathman_for(relation TEXT)

Permanently disable pg_pathman partitioning mechanism for the specified parent
table and remove the insert trigger if it exists.
All partitions and data remain unchanged.

postgres=# \sf disable_pathman_for
CREATE OR REPLACE FUNCTION public.disable_pathman_for(parent_relid regclass)
 RETURNS void
 LANGUAGE plpgsql
 STRICT
AS $function$
BEGIN
        PERFORM public.validate_relname(parent_relid);

        DELETE FROM public.pathman_config WHERE partrel = parent_relid;
        PERFORM public.drop_triggers(parent_relid);
```

```
    /* Notify backend about changes */
    PERFORM public.on_remove_partitions(parent_relid);
END
$function$
```

示例如下所示：

```
postgres=# select disable_pathman_for('part_test');
NOTICE:  drop cascades to 23 other objects
DETAIL:  drop cascades to trigger part_test_upd_trig on table part_test_3
drop cascades to trigger part_test_upd_trig on table part_test_4
drop cascades to trigger part_test_upd_trig on table part_test_5
drop cascades to trigger part_test_upd_trig on table part_test_6
drop cascades to trigger part_test_upd_trig on table part_test_7
drop cascades to trigger part_test_upd_trig on table part_test_8
drop cascades to trigger part_test_upd_trig on table part_test_9
drop cascades to trigger part_test_upd_trig on table part_test_10
drop cascades to trigger part_test_upd_trig on table part_test_11
drop cascades to trigger part_test_upd_trig on table part_test_12
drop cascades to trigger part_test_upd_trig on table part_test_13
drop cascades to trigger part_test_upd_trig on table part_test_14
drop cascades to trigger part_test_upd_trig on table part_test_15
drop cascades to trigger part_test_upd_trig on table part_test_16
drop cascades to trigger part_test_upd_trig on table part_test_17
drop cascades to trigger part_test_upd_trig on table part_test_18
drop cascades to trigger part_test_upd_trig on table part_test_19
drop cascades to trigger part_test_upd_trig on table part_test_20
drop cascades to trigger part_test_upd_trig on table part_test_21
drop cascades to trigger part_test_upd_trig on table part_test_22
drop cascades to trigger part_test_upd_trig on table part_test_23
drop cascades to trigger part_test_upd_trig on table part_test_24
drop cascades to trigger part_test_upd_trig on table part_test_25
 disable_pathman_for
---------------------

(1 row)

postgres=# \d+ part_test
                          Table "public.part_test"
  Column  |          Type           | Modifiers | Storage  | Stats target | Description
----------+-------------------------------+-----------+----------+--------------
+-------------
 id       | integer                 |           | plain    |              |
 info     | text                    |           | extended |              |
 crt_time | timestamp without time zone | not null  | plain    |              |
Child tables: part_test_10,
              part_test_11,
              part_test_12,
              part_test_13,
              part_test_14,
              part_test_15,
              part_test_16,
              part_test_17,
              part_test_18,
              part_test_19,
              part_test_20,
              part_test_21,
              part_test_22,
              part_test_23,
              part_test_24,
              part_test_25,
              part_test_26,
```

```
            part_test_27,
            part_test_28,
            part_test_29,
            part_test_3,
            part_test_30,
            part_test_31,
            part_test_32,
            part_test_33,
            part_test_34,
            part_test_35,
            part_test_4,
            part_test_5,
            part_test_6,
            part_test_7,
            part_test_8,
            part_test_9

postgres=# \d+ part_test_10
                        Table "public.part_test_10"
  Column  |          Type           | Modifiers | Storage  | Stats target | Description
----------+-------------------------------+-----------+----------+--------------
+-------------
 id       | integer                 |         | plain    |              |
 info     | text                    |         | extended |              |
 crt_time | timestamp without time zone | not null  | plain    |              |
Check constraints:
    "pathman_part_test_10_3_check" CHECK (crt_time >= '2017-06-25 00:00:00'::
timestamp without time zone AND crt_time < '2017-07-25 00:00:00'::timestamp
without time zone)
Inherits: part_test
```

禁用pg_pathman插件后，继承关系和约束不会变化，只是pg_pathman插件不介入custom scan执行计划。禁用pg_pathman插件后的执行计划如下：

```
postgres=# explain select * from part_test where crt_time='2017-06-25 00:00:00'::
timestamp;
                              QUERY PLAN
-----------------------------------------------------------------------------------
 Append  (cost=0.00..16.00 rows=2 width=45)
   ->  Seq Scan on part_test  (cost=0.00..0.00 rows=1 width=45)
         Filter: (crt_time = '2017-06-25 00:00:00'::timestamp without time zone)
   ->  Seq Scan on part_test_10  (cost=0.00..16.00 rows=1 width=45)
         Filter: (crt_time = '2017-06-25 00:00:00'::timestamp without time zone)
(5 rows)
```

注意：

disable_pathman_for没有可逆操作，请慎使用。

高级分区管理

1. 禁用主表

   当主表的数据全部迁移到分区后，可以禁用主表。接口函数如下：

```
set_enable_parent(relation REGCLASS, value BOOLEAN)


Include/exclude parent table into/from query plan.
```

```
In original PostgreSQL planner parent table is always included into query plan even if
it's empty which can lead to additional overhead.

You can use disable_parent() if you are never going to use parent table as a storage.

Default value depends on the partition_data parameter that was specified during
 initial partitioning in create_range_partitions() or create_partitions_from_range()
functions.

If the partition_data parameter was true then all data have already been migrated to
partitions and parent table disabled.

Otherwise it is enabled.
```

示例如下所示：

```
select set_enable_parent('part_test', false);
```

2. 自动扩展分区

范围分区表，允许自动扩展分区。如果新插入的数据不在已有的分区范围内，会自动创建分区。

```
set_auto(relation REGCLASS, value BOOLEAN)

Enable/disable auto partition propagation (only for RANGE partitioning).

It is enabled by default.
```

示例如下所示：

```
postgres=# \d+ part_test
                                Table "public.part_test"
  Column  |            Type             | Modifiers | Storage  | Stats target | Description
----------+-----------------------------+-----------+----------+--------------
+-------------
 id       | integer                     |           | plain    |              |
 info     | text                        |           | extended |              |
 crt_time | timestamp without time zone | not null  | plain    |              |
Child tables: part_test_10,
              part_test_11,
              part_test_12,
              part_test_13,
              part_test_14,
              part_test_15,
              part_test_16,
              part_test_17,
              part_test_18,
              part_test_19,
              part_test_20,
              part_test_21,
              part_test_22,
              part_test_23,
              part_test_24,
              part_test_25,
              part_test_26,
              part_test_3,
              part_test_4,
              part_test_5,
              part_test_6,
              part_test_7,
```

```
            part_test_8,
            part_test_9

postgres=# \d+ part_test_26
                        Table "public.part_test_26"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
----------+-----------------------------+-----------+----------+--------------
+-------------
 id      | integer                 |         | plain    |              |
 info    | text                    |         | extended |              |
 crt_time | timestamp without time zone | not null | plain    |              |
Check constraints:
    "pathman_part_test_26_3_check" CHECK (crt_time >= '2018-09-25 00:00:00'::
timestamp without time zone AND crt_time < '2018-10-25 00:00:00'::timestamp
without time zone)
Inherits: part_test

postgres=# \d+ part_test_25
                        Table "public.part_test_25"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
----------+-----------------------------+-----------+----------+--------------
+-------------
 id      | integer                 |         | plain    |              |
 info    | text                    |         | extended |              |
 crt_time | timestamp without time zone | not null | plain    |              |
Check constraints:
    "pathman_part_test_25_3_check" CHECK (crt_time >= '2018-08-25 00:00:00'::
timestamp without time zone AND crt_time < '2018-09-25 00:00:00'::timestamp
without time zone)
Inherits: part_test

插入一个不在已有分区范围的值，会根据创建分区时的interval自动扩展若干个分区，这个操作可能很久。
postgres=# insert into part_test values (1,'test','2222-01-01'::timestamp);

插入结束后，扩展了好多分区，原因是插入的值跨度范围太大了。

postgres=# \d+ part_test
                          Table "public.part_test"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
----------+-----------------------------+-----------+----------+--------------
+-------------
 id      | integer                 |         | plain    |              |
 info    | text                    |         | extended |              |
 crt_time | timestamp without time zone | not null | plain    |              |
Child tables: part_test_10,
            part_test_100,
            part_test_1000,
            part_test_1001,
       .....................................
       很多
```

说明：

不建议开启自动扩展范围分区，不合理的自动扩展可能会消耗大量的时间。

3. 回调函数（创建每个分区时都会触发）

回调函数是在每创建一个分区时会自动触发调用的函数。例如，可以用在DDL逻辑复制中，将DDL语句记录下来，存放到表中。回调函数如下：

```
set_init_callback(relation REGCLASS, callback REGPROC DEFAULT 0)

Set partition creation callback to be invoked for each attached or created partition (
both HASH and RANGE).

The callback must have the following signature:

part_init_callback(args JSONB) RETURNS VOID.

Parameter arg consists of several fields whose presence depends on partitioning type:

/* RANGE-partitioned table abc (child abc_4) */
{
   "parent":    "abc",
   "parttype":  "2",
   "partition": "abc_4",
   "range_max": "401",
   "range_min": "301"
}

/* HASH-partitioned table abc (child abc_0) */
{
   "parent":    "abc",
   "parttype":  "1",
   "partition": "abc_0"
}
```

示例如下所示：

```
回调函数
postgres=# create or replace function f_callback_test(jsonb) returns void as
$$
declare
begin
  create table if not exists rec_part_ddl(id serial primary key, parent name, parttype int
, partition name, range_max text, range_min text);
  if ($1->>'parttype')::int = 1 then
    raise notice 'parent: %, parttype: %, partition: %', $1->>'parent', $1->>'parttype', $1-
>>'partition';
    insert into rec_part_ddl(parent, parttype, partition) values (($1->>'parent')::name,
 ($1->>'parttype')::int, ($1->>'partition')::name);
  elsif ($1->>'parttype')::int = 2 then
    raise notice 'parent: %, parttype: %, partition: %, range_max: %, range_min: %', $1-
>>'parent', $1->>'parttype', $1->>'partition', $1->>'range_max', $1->>'range_min';
    insert into rec_part_ddl(parent, parttype, partition, range_max, range_min) values
 (($1->>'parent')::name, ($1->>'parttype')::int, ($1->>'partition')::name, $1->>'
range_max', $1->>'range_min');
  end if;
end;
$$ language plpgsql strict;

测试表
postgres=# create table tt(id int, info text, crt_time timestamp not null);
CREATE TABLE

设置测试表的回调函数
```

```
select set_init_callback('tt'::regclass, 'f_callback_test'::regproc);

创建分区
postgres=# select
create_range_partitions('tt'::regclass,                -- 主表OID
                    'crt_time',                    -- 分区列名
                    '2016-10-25 00:00:00'::timestamp,  -- 开始值
                    interval '1 month',              -- 间隔；interval 类型，用于时间分区表
                    24,                           -- 分多少个区
                    false) ;
 create_range_partitions
-------------------------
                      24
(1 row)

检查回调函数是否已调用
postgres=# select * from rec_part_ddl;
 id | parent | parttype | partition |     range_max      |     range_min
----+--------+----------+-----------+--------------------+---------------------
  1 | tt     |        2 | tt_1      | 2016-11-25 00:00:00 | 2016-10-25 00:00:00
  2 | tt     |        2 | tt_2      | 2016-12-25 00:00:00 | 2016-11-25 00:00:00
  3 | tt     |        2 | tt_3      | 2017-01-25 00:00:00 | 2016-12-25 00:00:00
  4 | tt     |        2 | tt_4      | 2017-02-25 00:00:00 | 2017-01-25 00:00:00
  5 | tt     |        2 | tt_5      | 2017-03-25 00:00:00 | 2017-02-25 00:00:00
  6 | tt     |        2 | tt_6      | 2017-04-25 00:00:00 | 2017-03-25 00:00:00
  7 | tt     |        2 | tt_7      | 2017-05-25 00:00:00 | 2017-04-25 00:00:00
  8 | tt     |        2 | tt_8      | 2017-06-25 00:00:00 | 2017-05-25 00:00:00
  9 | tt     |        2 | tt_9      | 2017-07-25 00:00:00 | 2017-06-25 00:00:00
 10 | tt     |        2 | tt_10     | 2017-08-25 00:00:00 | 2017-07-25 00:00:00
 11 | tt     |        2 | tt_11     | 2017-09-25 00:00:00 | 2017-08-25 00:00:00
 12 | tt     |        2 | tt_12     | 2017-10-25 00:00:00 | 2017-09-25 00:00:00
 13 | tt     |        2 | tt_13     | 2017-11-25 00:00:00 | 2017-10-25 00:00:00
 14 | tt     |        2 | tt_14     | 2017-12-25 00:00:00 | 2017-11-25 00:00:00
 15 | tt     |        2 | tt_15     | 2018-01-25 00:00:00 | 2017-12-25 00:00:00
 16 | tt     |        2 | tt_16     | 2018-02-25 00:00:00 | 2018-01-25 00:00:00
 17 | tt     |        2 | tt_17     | 2018-03-25 00:00:00 | 2018-02-25 00:00:00
 18 | tt     |        2 | tt_18     | 2018-04-25 00:00:00 | 2018-03-25 00:00:00
 19 | tt     |        2 | tt_19     | 2018-05-25 00:00:00 | 2018-04-25 00:00:00
 20 | tt     |        2 | tt_20     | 2018-06-25 00:00:00 | 2018-05-25 00:00:00
 21 | tt     |        2 | tt_21     | 2018-07-25 00:00:00 | 2018-06-25 00:00:00
 22 | tt     |        2 | tt_22     | 2018-08-25 00:00:00 | 2018-07-25 00:00:00
 23 | tt     |        2 | tt_23     | 2018-09-25 00:00:00 | 2018-08-25 00:00:00
 24 | tt     |        2 | tt_24     | 2018-10-25 00:00:00 | 2018-09-25 00:00:00
(24 rows)
```

19.3 使用中文分词

本文为您介绍POLARDB for PostgreSQL如何启用中文分词以及自定义中文分词词典。

启用中文分词

可以使用下面的命令，启用中文分词：

```
CREATE EXTENSION zhparser;
CREATE TEXT SEARCH CONFIGURATION testzhcfg (PARSER = zhparser);
ALTER TEXT SEARCH CONFIGURATION testzhcfg ADD MAPPING FOR n,v,a,i,e,l WITH simple;
--可选的参数设定
alter role all set zhparser.multi_short=on;
--简单测试
```

```
SELECT * FROM ts_parse('zhparser', 'hello world! 2010年保障房建设在全国范围内获全面启动，从中央到地方纷纷加大了保障房的建设和投入力度。2011年，保障房进入了更大规模的建设阶段。住房城乡建设部党组书记、部长姜伟新去年底在全国住房城乡建设工作会议上表示，要继续推进保障性安居工程建设。');
SELECT to_tsvector('testzhcfg',' “今年保障房新开工数量虽然有所下调，但实际的年度在建规模以及竣工规模会超以往年份，相对应的对资金的需求也会创历史纪录。” 陈国强说。在他看来，与2011年相比，2012年的保障房建设在资金配套上的压力将更为严峻。');
SELECT to_tsquery('testzhcfg', '保障房资金压力');
```

利用分词进行全文索引的方法如下：

```
--为T1表的name字段创建全文索引
create index idx_t1 on t1 using gin (to_tsvector('zhcfg',upper(name) ));
--使用全文索引
 select * from t1 where to_tsvector('zhcfg',upper(t1.name)) @@ to_tsquery('zhcfg','(防火)') ;
```

自定义中文分词词典

自定义中文分词词典，示例如下：

```
-- 确实的分词结果
SELECT to_tsquery('testzhcfg', '保障房资金压力');
-- 往自定义分词词典里面插入新的分词
insert into pg_ts_custom_word values ('保障房资');
-- 使新的分词生效
select zhprs_sync_dict_xdb();
-- 退出此连接
\c
-- 重新查询，可以得到新的分词结果
SELECT to_tsquery('testzhcfg', '保障房资金压力');
```

使用自定义分词的注意事项如下：

- 最多支持一百万条自定义分词，超出部分不做处理，必须保证分词数量在这个范围之内。自定义分词与缺省的分词词典将共同产生作用。
- 每个词的最大长度为128字节，超出部分将会截取。
- 增删改分词之后必须执行`select zhprs_sync_dict_xdb();`并且重新建立连接才会生效。

20 错误码

本文将为您介绍PolarDB-P错误码。

错误码格式

PolarDB-P服务器发出的所有消息都遵循SQL标准对“SQLSTATE”码的约定，被赋予一个5个字符的错误码。如果应用程序想要判断发生了什么错误，应该检查返回消息的错误码，而非检测消息的文本。不同发布版本之间的错误码一般不会发生改变，但其错误文本可能变化。

根据SQL标准，错误码的前两位表示错误类别，后三位表示当前错误类别下的特定情况。应用程序即便无法识别特定的错误码，依然可以通过错误码的前两位识别出错误类型。

以下列出所有的错误类和错误码，对于每个错误类有一个最后三位为000的错误码，该错误码用于表示当前错误类下没有赋予特定错误情况的错误。

错误码列表

说明:

条件名（Condition Name）表示在存储过程中可以使用的条件名，在存储过程中该条件名不区分大小写。

存储过程不识别warning类型的条件名，对应的错误类为 00 , 01 和 02。

表 20-1: Class 00 — Successful Completion

Error Code	Condition Name
00000	successful_completion

表 20-2: Class 01 — Warning

Error Code	Condition Name
01000	warning
0100C	dynamic_result_sets_returned
01008	implicit_zero_bit_padding
01003	null_value_eliminated_in_set_function
01007	privilege_not_granted
01006	privilege_not_revoked
01004	string_data_right_truncation

Error Code	Condition Name
01P01	deprecated_feature

表 20-3: Class 02 — No Data (this is also a warning class per the SQL standard)

Error Code	Condition Name
02000	no_data
02001	no_additional_dynamic_result_sets_returned

表 20-4: lass 03 — SQL Statement Not Yet Complete

Error Code	Condition Name
03000	sql_statement_not_yet_complete

表 20-5: Class 08 — Connection Exception

Error Code	Condition Name
08000	connection_exception
08003	connection_does_not_exist
08006	connection_failure
08001	sqlclient_unable_to_establish_sqlconnection
08004	sqlserver_rejected_establishment_of_sqlconnection
08007	transaction_resolution_unknown
08P01	protocol_violation

表 20-6: Class 09 — Triggered Action Exception

Error Code	Condition Name
09000	triggered_action_exception

表 20-7: Class 0A — Feature Not Supported

Error Code	Condition Name
0A000	feature_not_supported

表 20-8: Class 0B — Invalid Transaction Initiation

Error Code	Condition Name
0B000	invalid_transaction_initiation

表 20-9: Class 0F — Locator Exception

Error Code	Condition Name
0F000	locator_exception
0F001	invalid_locator_specification

表 20-10: Class 0L — Invalid Grantor

Error Code	Condition Name
0L000	invalid_grantor
0LP01	invalid_grant_operation

表 20-11: Class 0P — Invalid Role Specification

Error Code	Condition Name
0P000	invalid_role_specification

表 20-12: Class 0Z — Diagnostics Exception

Error Code	Condition Name
0Z000	diagnostics_exception
0Z002	stacked_diagnostics_accessed_without_active_handler

表 20-13: Class 20 — Case Not Found

Error Code	Condition Name
20000	case_not_found

表 20-14: Class 21 — Cardinality Violation

Error Code	Condition Name
21000	cardinality_violation

表 20-15: Class 22 — Data Exception

Error Code	Condition Name
22000	data_exception
2202E	array_subscript_error
22021	character_not_in_repertoire
22008	datetime_field_overflow
22012	division_by_zero
22005	error_in_assignment
2200B	escape_character_conflict
22022	indicator_overflow
22015	interval_field_overflow
2201E	invalid_argument_for_logarithm
22014	invalid_argument_for_ntile_function
22016	invalid_argument_for_nth_value_function
2201F	invalid_argument_for_power_function
2201G	invalid_argument_for_width_bucket_function
22018	invalid_character_value_for_cast
22007	invalid_datetime_format
22019	invalid_escape_character
2200D	invalid_escape_octet
22025	invalid_escape_sequence
22P06	nonstandard_use_of_escape_character
22010	invalid_indicator_parameter_value
22023	invalid_parameter_value
22013	invalid_preceding_or_following_size
2201B	invalid_regular_expression
2201W	invalid_row_count_in_limit_clause
2201X	invalid_row_count_in_result_offset_clause
2202H	invalid_tablesample_argument
2202G	invalid_tablesample_repeat

Error Code	Condition Name
22009	invalid_time_zone_displacement_value
2200C	invalid_use_of_escape_character
2200G	most_specific_type_mismatch
22004	null_value_not_allowed
22002	null_value_no_indicator_parameter
22003	numeric_value_out_of_range
2200H	sequence_generator_limit_exceeded
22026	string_data_length_mismatch
22001	string_data_right_truncation
22011	substring_error
22027	trim_error
22024	unterminated_c_string
2200F	zero_length_character_string
22P01	floating_point_exception
22P02	invalid_text_representation
22P03	invalid_binary_representation
22P04	bad_copy_file_format
22P05	untranslatable_character
2200L	not_an_xml_document
2200M	invalid_xml_document
2200N	invalid_xml_content
2200S	invalid_xml_comment
2200T	invalid_xml_processing_instruction

表 20-16: Class 23 — Integrity Constraint Violation

Error Code	Condition Name
23000	integrity_constraint_violation
23001	restrict_violation
23502	not_null_violation
23503	foreign_key_violation

Error Code	Condition Name
23505	unique_violation
23514	check_violation
23P01	exclusion_violation

表 20-17: Class 24 — Invalid Cursor State

Error Code	Condition Name
24000	invalid_cursor_state

表 20-18: Class 25 — Invalid Transaction State

Error Code	Condition Name
25000	invalid_transaction_state
25001	active_sql_transaction
25002	branch_transaction_already_active
25008	held_cursor_requires_same_isolation_level
25003	inappropriate_access_mode_for_branch_transaction
25004	inappropriate_isolation_level_for_branch_transaction
25005	no_active_sql_transaction_for_branch_transaction
25006	read_only_sql_transaction
25007	schema_and_data_statement_mixing_not_supported
25P01	no_active_sql_transaction
25P02	in_failed_sql_transaction
25P03	idle_in_transaction_session_timeout

表 20-19: Class 26 — Invalid SQL Statement Name

Error Code	Condition Name
26000	invalid_sql_statement_name

表 20-20: Class 27 — Triggered Data Change Violation

Error Code	Condition Name
27000	triggered_data_change_violation

表 20-21: Class 28 — Invalid Authorization Specification

Error Code	Condition Name
28000	invalid_authorization_specification
28P01	invalid_password

表 20-22: Class 2B — Dependent Privilege Descriptors Still Exist

Error Code	Condition Name
2B000	dependent_privilege_descriptors_still_exist
2BP01	dependent_objects_still_exist

表 20-23: Class 2D — Invalid Transaction Termination

Error Code	Condition Name
2D000	invalid_transaction_termination

表 20-24: Class 2F — SQL Routine Exception

Error Code	Condition Name
2F000	sql_routine_exception
2F005	function_executed_no_return_statement
2F002	modifying_sql_data_not_permitted
2F003	prohibited_sql_statement_attempted
2F004	reading_sql_data_not_permitted

表 20-25: Class 34 — Invalid Cursor Name

Error Code	Condition Name
34000	invalid_cursor_name

表 20-26: Class 38 — External Routine Exception

Error Code	Condition Name
38000	external_routine_exception
38001	containing_sql_not_permitted
38002	modifying_sql_data_not_permitted

Error Code	Condition Name
38003	prohibited_sql_statement_attempted
38004	reading_sql_data_not_permitted

表 20-27: Class 39 — External Routine Invocation Exception

Error Code	Condition Name
39000	external_routine_invocation_exception
39001	invalid_sqlstate_returned
39004	null_value_not_allowed
39P01	trigger_protocol_violated
39P02	srf_protocol_violated
39P03	event_trigger_protocol_violated

表 20-28: Class 3B — Savepoint Exception

Error Code	Condition Name
3B000	savepoint_exception
3B001	invalid_savepoint_specification

表 20-29: Class 3D — Invalid Catalog Name

Error Code	Condition Name
3D000	invalid_catalog_name

表 20-30: Class 3F — Invalid Schema Name

Error Code	Condition Name
3F000	invalid_schema_name

表 20-31: Class 40 — Transaction Rollback

Error Code	Condition Name
40000	transaction_rollback
40002	transaction_integrity_constraint_violation
40001	serialization_failure

Error Code	Condition Name
40003	statement_completion_unknown
40P01	deadlock_detected

表 20-32: Class 42 — Syntax Error or Access Rule Violation

Error Code	Condition Name
42000	syntax_error_or_access_rule_violation
42601	syntax_error
42501	insufficient_privilege
42846	cannot_coerce
42803	grouping_error
42P20	windowing_error
42P19	invalid_recursion
42830	invalid_foreign_key
42602	invalid_name
42622	name_too_long
42939	reserved_name
42804	datatype_mismatch
42P18	indeterminate_datatype
42P21	collation_mismatch
42P22	indeterminate_collation
42809	wrong_object_type
428C9	generated_always
42703	undefined_column
42883	undefined_function
42P01	undefined_table
42P02	undefined_parameter
42704	undefined_object
42701	duplicate_column
42P03	duplicate_cursor
42P04	duplicate_database

Error Code	Condition Name
42723	duplicate_function
42P05	duplicate_prepared_statement
42P06	duplicate_schema
42P07	duplicate_table
42712	duplicate_alias
42710	duplicate_object
42702	ambiguous_column
42725	ambiguous_function
42P08	ambiguous_parameter
42P09	ambiguous_alias
42P10	invalid_column_reference
42611	invalid_column_definition
42P11	invalid_cursor_definition
42P12	invalid_database_definition
42P13	invalid_function_definition
42P14	invalid_prepared_statement_definition
42P15	invalid_schema_definition
42P16	invalid_table_definition
42P17	invalid_object_definition

表 20-33: Class 44 — WITH CHECK OPTION Violation

Error Code	Condition Name
44000	with_check_option_violation

表 20-34: Class 53 — Insufficient Resources

Error Code	Condition Name
53000	insufficient_resources
53100	disk_full
53200	out_of_memory
53300	too_many_connections

Error Code	Condition Name
53400	configuration_limit_exceeded

表 20-35: Class 54 — Program Limit Exceeded

Error Code	Condition Name
54000	program_limit_exceeded
54001	statement_too_complex
54011	too_many_columns
54023	too_many_arguments

表 20-36: Class 55 — Object Not In Prerequisite State

Error Code	Condition Name
55000	object_not_in_prerequisite_state
55006	object_in_use
55P02	cant_change_runtime_param
55P03	lock_not_available

表 20-37: Class 57 — Operator Intervention

Error Code	Condition Name
57000	operator_intervention
57014	query_canceled
57P01	admin_shutdown
57P02	crash_shutdown
57P03	cannot_connect_now
57P04	database_dropped

表 20-38: Class 58 — System Error (errors external to PostgreSQL itself)

Error Code	Condition Name
58000	system_error
58030	io_error
58P01	undefined_file

Error Code	Condition Name	
58P02	duplicate_file	

表 20-39: Class 72 — Snapshot Failure

Error Code	Condition Name
72000	snapshot_too_old

表 20-40: Class F0 — Configuration File Error

Error Code	Condition Name
F0000	config_file_error
F0001	lock_file_exists

表 20-41: Class HV — Foreign Data Wrapper Error (SQL/MED)

Error Code	Condition Name
HV000	fdw_error
HV005	fdw_column_name_not_found
HV002	fdw_dynamic_parameter_value_needed
HV010	fdw_function_sequence_error
HV021	fdw_inconsistent_descriptor_information
HV024	fdw_invalid_attribute_value
HV007	fdw_invalid_column_name
HV008	fdw_invalid_column_number
HV004	fdw_invalid_data_type
HV006	fdw_invalid_data_type_descriptors
HV091	fdw_invalid_descriptor_field_identifier
HV00B	fdw_invalid_handle
HV00C	fdw_invalid_option_index
HV00D	fdw_invalid_option_name
HV090	fdw_invalid_string_length_or_buffer_length
HV00A	fdw_invalid_string_format
HV009	fdw_invalid_use_of_null_pointer

Error Code	Condition Name
HV014	fdw_too_many_handles
HV001	fdw_out_of_memory
HV00P	fdw_no_schemas
HV00J	fdw_option_name_not_found
HV00K	fdw_reply_handle
HV00Q	fdw_schema_not_found
HV00R	fdw_table_not_found
HV00L	fdw_unable_to_create_execution
HV00M	fdw_unable_to_create_reply
HV00N	fdw_unable_to_establish_connection

表 20-42: Class P0 — PL/pgSQL Error

Error Code	Condition Name
P0000	plpgsql_error
P0001	raise_exception
P0002	no_data_found
P0003	too_many_rows
P0004	assert_failure

表 20-43: Class XX — Internal Error

Error Code	Condition Name
XX000	internal_error
XX001	data_corrupted
XX002	index_corrupted

21 常见错误处理方法

本文为您介绍PolarDB-P常见错误的处理方式。

连接异常

连接异常即应用程序或者客户端与数据库的连接出现异常，例如已经创建的连接，提示连接不存在或连接超时，无法与数据库实例建立连接等。连接异常经常发生在网络闪断，或者数据库服务重启时，您需要在应用程序中就此类异常，添加重试逻辑。如果重试多次仍无法创建连接，请及时通过工单咨询。

数据异常

数据异常通常表现为函数参数无效，数组下标错误，除零，数据格式无效，转移字符无效等，具体的错误内容均可通过错误码和condition name来判断。此类错误通常需要根据报错的具体内容，定位SQL的错误位置，修复SQL并重试。

语法错误

此类错误说明SQL中存在语法问题，如使用未定义的列、函数、表、参数，有重复的列、数据库、函数、表或者别名等，通常错误信息中会告知错误的具体位置和错误类型，您可据此修复语法错误。

资源不足

资源不足通常表现为磁盘满，内存超限（OOM，out of memory），连接太多或者特定资源的使用超过配置所限。此类异常通常可以通过升级实例规格予以解决。不过仍然需要具体场景具体分析，如连接太多，可能由于应用同时有太多连接导致超过实例最大连接数；也可能有慢SQL或数据库实例资源不足（如CPU或内存），导致大量连接堆积。您需根据具体情况，适当减少连接数或者排查并调优慢SQL。