ALIBABA CLOUD

阿里云

PolarDB PostgreSQL

用户指南

文档版本：20220609

阿里云

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

目录

1.概述

PolarDB是阿里巴巴自研的新一代云原生关系型数据库，在存储计算分离架构下，利用了软硬件结合的优势，为用户提供具备极致弹性、高性能、海量存储、安全可靠的数据库服务。100%兼容MySQL 5.6/5.7/8.0，PostgreSQL 11，高度兼容Oracle。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
  一个集群版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。更多信息请参见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。

PolarDB控制台地址：PolarDB控制台。

2.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 PostgreSQL迁移至PolarDB PostgreSQL引擎
从自建数据库迁移至PolarDB	从自建PostgreSQL迁移至PolarDB PostgreSQL引擎

3.数据迁移

3.1. 从自建PostgreSQL迁移至

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

前提条件

PolarDB PostgreSQL引擎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 PostgreSQL引擎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。

```
-- PostgreSQL database cluster dump
--

SET default_transaction_read_only = off;

SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;

--
-- Roles
--

CREATE ROLE data1;
ALTER ROLE data1 WITH NOSUPERUSER INHERIT CREATEROLE CREATEDB LOGIN NOREPLICATION NOBYPASSRLS PASSWORD 'md5[redacted]';
CREATE ROLE manisha;
ALTER ROLE manisha WITH NOSUPERUSER INHERIT NOCREATEROLE NOCREATEDB LOGIN NOREPLICATION NOBYPASSRLS PASSWORD 'md[redacted]';
CREATE ROLE postgres;
ALTER ROLE postgres WITH SUPERUSER INHERIT CREATEROLE CREATEDB LOGIN REPLICATION BYPASSRLS PASSWORD 'md5[redacted]';
CREATE ROLE testuser;
ALTER ROLE testuser WITH NOSUPERUSER INHERIT NOCREATEROLE CREATEDB LOGIN NOREPLICATION NOBYPASSRLS;



--
-- PostgreSQL database cluster dump complete
--
```

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 PostgreSQL引擎

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 PostgreSQL引擎实例中。

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

参数说明：

- <username>：登录PolarDB PostgreSQL引擎数据库的账号。
- <hostname>：PolarDB PostgreSQL引擎实例的主地址（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。
- <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 testdb -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 PostgreSQL引擎实例中。

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

参数说明：

- <username>：登录PolarDB PostgreSQL引擎数据库的账号。
- <hostname>：PolarDB PostgreSQL引擎实例的主地址（私网），详情请参考查看或申请连接地址。
- <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，请参考修改密码。

等待数据迁移完成即可。

3.2. 从RDS PostgreSQL迁移至

本文介绍通过 `pg_dump` 和 `pg_restore` 命令将RDS PostgreSQL数据库迁移至PolarDB PostgreSQL引擎中。
迁移的源库为自建PostgreSQL数据库时，请参考从自建PostgreSQL迁移至PolarDB PostgreSQL引擎。

前提条件

PolarDB PostgreSQL引擎实例的存储空间应大于RDS 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 PostgreSQL引擎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 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 PostgreSQL引擎

1. 在ECS上连接PolarDB PostgreSQL引擎数据库。

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

参数说明：

- <username>：登录PolarDB PostgreSQL引擎数据库的账号。
- <hostname>：PolarDB PostgreSQL引擎实例的主地址（私网），详情请参考查看或申请连接地址。
- <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 PostgreSQL实例数据库中的角色信息，在目标PolarDB PostgreSQL引擎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 PostgreSQL引擎实例中。

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

参数说明：

- <username>：登录PolarDB PostgreSQL引擎数据库的账号。
- <hostname>：PolarDB PostgreSQL引擎实例的主地址（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 p
ostgresdump
```

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，请参考管理数据库账号。

等待数据迁移完成即可。

4.查看并管理计划内事件

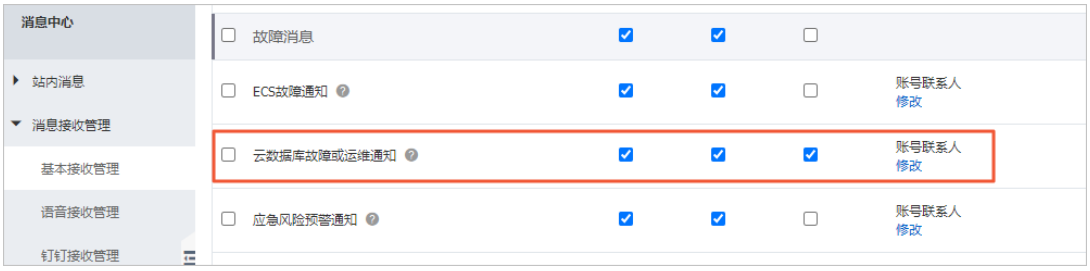
PolarDB计划内的运维事件（例如数据库软件升级、硬件维护与升级）除了会通过短信、语音、邮件或站内信通知您，还会在控制台上进行通知。您可以在计划内事件中，查看具体的事件类型、任务ID、集群名称、切换时间等，也可以手动修改切换时间。

注意事项

- 当您有未处理的运维事件时，可以在控制台左侧导航栏的**事件中心 > 计划内事件**中看到事件提醒。

- 云数据库的计划内事件（如数据库软件升级、硬件维护与升级等）通常会至少在执行前的3天通知您，通知方式为短信、语音、邮件、站内信或控制台等。您需要登录消息中心，确保**云数据库故障或运维通知**的通知开关处于开启状态并设置消息接收人（推荐设置为数据库运维人员），否则您将无法收到相应的通知信息。

  消息中心通知设置



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. 在左侧导航栏中，单击**事件中心 > 计划内事件**。

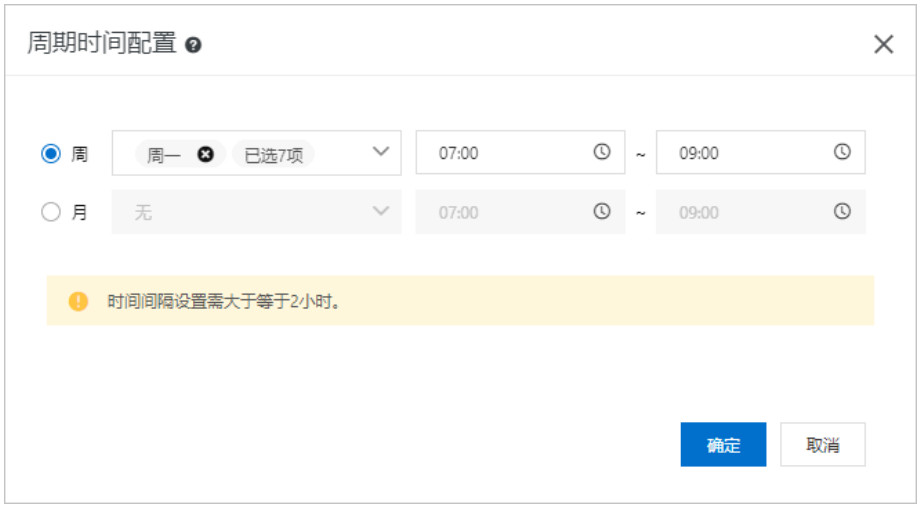
   > **说明** 强制要求预约时间的运维事件会弹窗提醒，请尽快完成预约。

4. （可选）在**计划内事件**页面，可以进行周期时间配置。

i. 在页面右侧，单击**周期时间配置**。

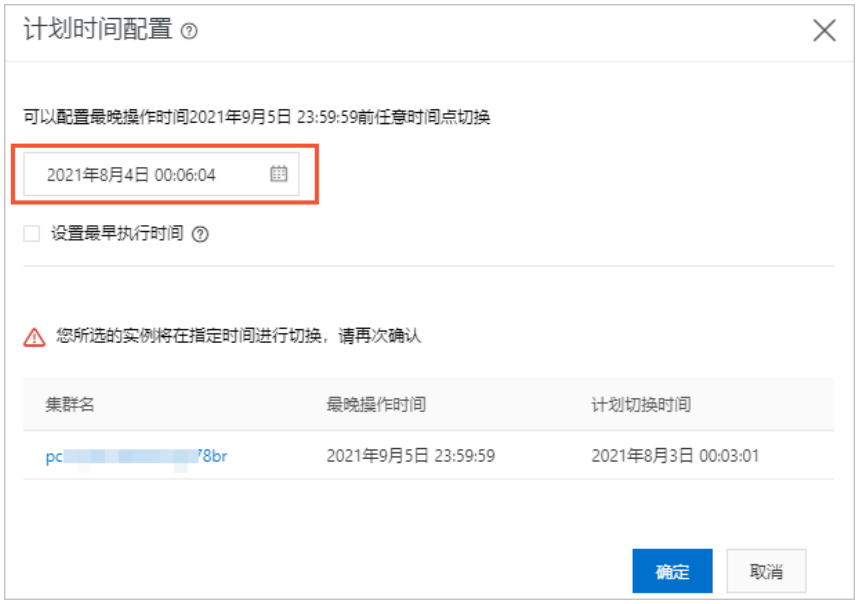
> **说明** **周期时间配置**是数据库主动运维事件的全局配置项（不包含紧急风险修复类事件）。周期时间设置后，新生成的主动运维事件的计划切换时间会自动命中周期时间；如果不设置周期时间，新生成的主动运维事件的计划切换时间会自动命中集群的可维护窗口时间。

ii. 在弹出的对话框中，设置周期时间并单击**确定**。



5. 在**计划内事件**页面，可以查看事件的详细信息，如需修改事件的切换时间，请选中目标集群，单击**计划时间配置**。

6. 在**计划时间配置**对话框中，设置**计划切换时间**并单击**确定**。



说明

- 您可以勾选**设置最早执行时间**，系统将自动填充最近的预约切换日期和预约切换时间。**确定**后集群开始切换准备，进入待处理状态；取消勾选后，可自定义修改预约切换日期和时间。
- **计划切换时间**不能晚于**最晚开始时间**。

事件的原因与影响

升级类型	事件原因	影响类型	影响说明
热升级	实例迁移	实例闪断	进入 后，将产生下述影响： • 一般情况下，实例小版本升级采用热升级模式。实例或实例中涉及切换的分片将发生连接闪断及30秒以内的只读状态（用于等待数据完全同步），请在业务低峰期执行，并确保应用程序具备重连机制。 • 短暂影响该实例在DMS和DTS中的使用，操作完成后自动恢复正常。 计划切换时间
	主备切换		
	实例参数调整		
	主机风险修复		
	SSL证书更新		
	备份模式升级		
	小版本升级	实例闪断	进入 后，将产生下述影响： • 一般情况下，实例小版本升级采用热升级模式。实例或实例中涉及切换的分片将发生连接闪断及30秒以内的只读状态（用于等待数据完全同步），请在业务低峰期执行，并确保应用程序具备重连机制。 • 短暂影响该实例在DMS和DTS中的使用，操作完成后自动恢复正常。
		小版本号间的差异	不同的小版本号（内核版本号）更新的内容有所区别，您需要关注升级后的小版本和当前小版本的差异，具体请参见新版本更新说明。
	代理小版本升级	实例闪断	进入 后，将产生下述影响： • 一般情况下，实例小版本升级采用热升级模式。实例或实例中涉及切换的分片将发生连接闪断及30秒以内的只读状态（用于等待数据完全同步），请在业务低峰期执行，并确保应用程序具备重连机制。 • 短暂影响该实例在DMS和DTS中的使用，操作完成后自动恢复正常。

升级类型	事件原因	影响类型	影响说明
		小版本号间的差异	不同的小版本号更新的内容有所区别，您需要关注升级后的小版本和当前小版本的差异。
	网络升级	实例闪断	进入 后，将产生下述影响： • 一般情况下，实例小版本升级采用热升级模式。实例或实例中涉及切换的分片将发生连接闪断及30秒以内的只读状态（用于等待数据完全同步），请在业务低峰期执行，并确保应用程序具备重连机制。 • 短暂影响该实例在DMS和DTS中的使用，操作完成后自动恢复正常。
		VIP直连影响	部分网络升级过程中可能涉及跨可用区迁移，实例的虚拟IP（VIP）地址会发生改变，如果客户端使用VIP连接云数据库将会引起连接中断。 **说明** 为避免影响，您应当使用实例提供的域名形式的连接地址，同时关闭应用及其所属服务器的DNS缓存。
	存储网关升级	I/O 抖动	可能出现短暂的I/O抖动或SQL时延增加，影响的时间不超过3秒。
冷升级	大版本实例升级	实例闪断	进入 后，将产生下述影响： • 实例或实例中涉及切换的分片通常会有2分钟以内的连接闪断。对于表文件较多的实例、升级前有正在执行的大事务实例或升级前CPU使用率较高的实例，冷升级过程中的闪断时间可能会超过2分钟。请在业务低峰期执行，并确保应用程序具备重连机制。 • 如遇到需要冷升级的情况，请关注版本间的差异，并根据业务情况选择合适的升级时间。 计划切换时间
	公测版本升级		

相关API

API	描述
DescribePendingMaintenanceActions	查看不同任务类型下待处理事件的数量。
ModifyPendingMaintenanceAction	修改待处理事件的任务切换时间。
DescribePendingMaintenanceAction	查询待处理事件的详情。

5.设置集群白名单

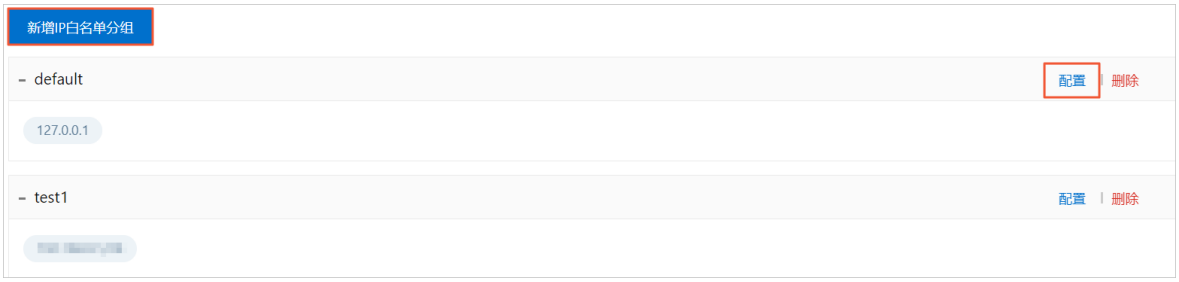
创建PolarDB PostgreSQL引擎数据库集群后，您需要设置PolarDB集群的IP白名单，并创建集群的初始账号，才能连接和使用该集群。

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 > 集群白名单**。
5. 在**集群白名单**页面，您可以**新增IP白名单分组**或**配置**已有白名单。



   - 新增白名单分组
     a. 单击**新增IP白名单分组**。

b. 在**新增IP白名单分组**对话框，输入分组名称和允许访问的IP白名单地址。

> **说明** IP白名单分组名称需满足如下要求：
> - 由小写字母、数字、下划线（_）组成。
> - 由字母开头、字母或数字结尾。
> - 长度为2~120个字符。

- 配置白名单

  a. 单击目标IP白名单分组名称右侧的**配置**。

b. 在**配置白名单**对话框，输入允许访问的IP白名单地址。

> **说明**
> - 每个集群都默认包含一个 `default` 的白名单分组，且只包含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。

6. 单击**确定**即可。

> **说明** 目前支持创建50个IP白名单，所有IP白名单累积支持添加1000个IP地址或地址段。

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

- 已添加ECS的IP地址到IP白名单中，但是还是无法访问。
  答：
  i. 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。
  ii. 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ClassicLink打通经典网络到专有网络的网络。
  iii. 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。

- 什么原因导致公网连接失败？
  答：
  i. 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。
  ii. 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。请参考查看或申请连接地址确定真正的公网地址。
- 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。
- 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。

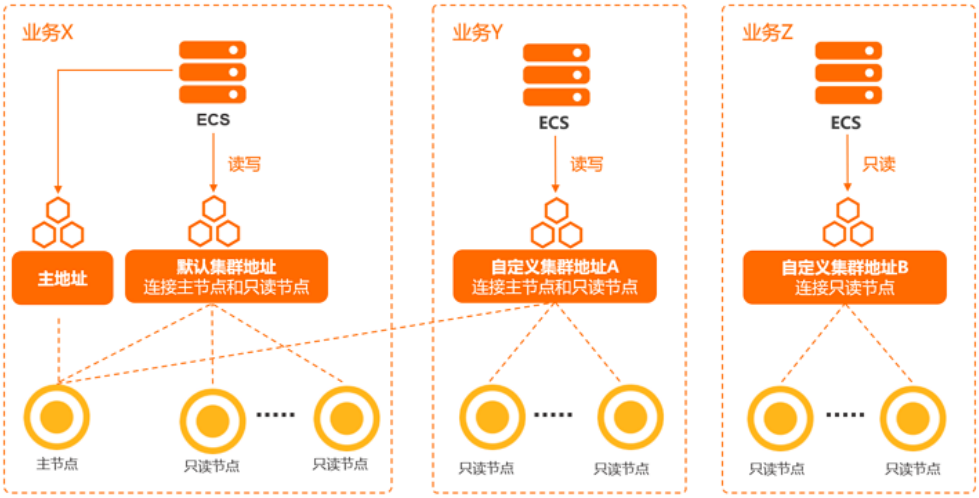
```
1
2
3 CREATE USER 'alitest'@'192.168.1.101' ;
4
5
6 select * from mysql.user where user='alitest';
```

6.集群访问

6.1. 查看或申请连接地址

在连接PolarDB集群时，您需要填写PolarDB集群的连接地址。PolarDB为集群地址和主地址分别提供了私网和公网的连接地址，本文将介绍如何在控制台查看和申请连接地址。

集群地址和主地址



地址类型	地址说明	支持的网络类型
集群地址（推荐）	• 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。 **说明** PolarDB包含一个默认的集群地址，您还可以根据业务需求创建多个自定义的集群地址，自定义集群地址可以连接到指定的节点，以及设置读写模式等。具体信息，请参见创建自定义集群地址。	• 私网 • 公网
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，主访问地址会自动切换到新的主节点。	

私网地址和公网地址

网络类型	说明	使用场景
私网	• 通过私网的连接地址访问可以发挥PolarDB的最佳性能。 • 创建集群时会默认生成一个私网的连接地址，该地址支持修改但无法释放，关于如何修改，请参见修改连接地址。	例如： • 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访问数据库集群。
公网	• 您可以申请或释放公网的连接地址。具体操作，请参见申请连接地址和释放连接地址。 • 公网即因特网，通过公网访问将无法实现PolarDB最佳性能。	例如：通过公网访问数据库集群进行维护操作。

查看连接地址和端口

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**链接地址**区域，您可以通过以下任意一种方式来查看连接地址和端口信息。
   - 方式一

     单击**链接地址**区域右上角的图标切换视图，即可查看目标连接地址和端口信息。

   - 方式二

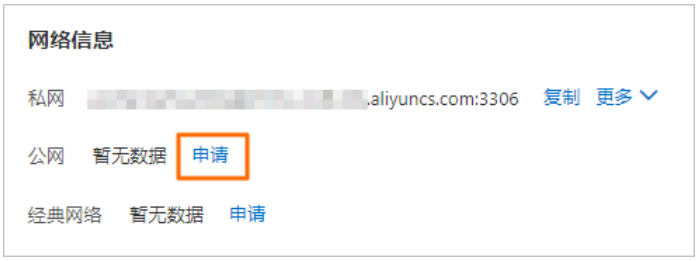
     单击目标集群地址右侧的**编辑配置**，即可在弹出的对话框中查看**网络信息**，包括连接地址和端口号。

> **说明**
>
> - 如果您之前是通过域名连接到数据库，当数据库迁移上云后，想要保留原来的数据库域名，可以单击**绑定私有域名**进行绑定。仅**私网**的连接地址支持设置私有域名绑定。具体内容，请参见私有域名。
> - 连接地址的默认端口号为1521，不可变更。

申请连接地址

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**链接地址**区域，您可以通过以下任意一种方式来查看连接地址和端口信息。
5. 单击**申请**。
   - 方式一：
     a. 单击**链接地址**区域右上角图标切换视图。
     b. 单击**申请**。

   - 方式二：
     a. 单击目标集群地址右侧的**编辑配置**。

b. 在弹出的对话框中的**网络信息**区域，单击**申请**。

> **说明**
> - 仅支持申请**公网**的连接地址。
> - 创建集群时会默认生成一个**私网**地址，无需申请。

6. 在弹出的对话框中，设置连接地址前缀，单击**确定**。

> **说明** 连接地址前缀需满足如下条件：
> - 由小写字母、数字、中划线（-）组成，6~30个字符。
> - 以字母开头，以数字或字母结尾。

下一步

连接数据库集群

相关API

API	描述
DescribeDBClusterEndpoints	查询集群的地址信息。
CreateDBEndpointAddress	创建集群的公网地址。
ModifyDBEndpointAddress	修改集群默认访问地址。
DeleteDBEndpointAddress	释放集群地址。

6.2. 修改或释放连接地址

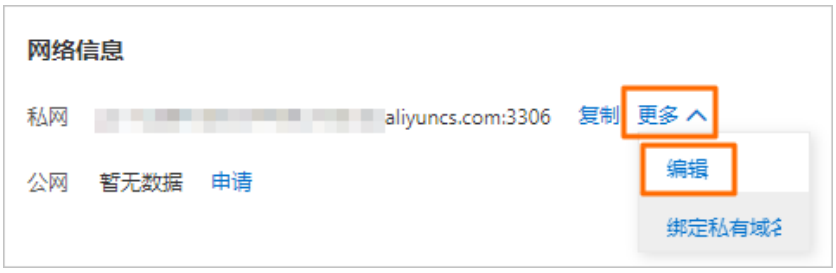
在连接PolarDB集群时，您需要填写PolarDB集群的连接地址。PolarDB为集群地址和主地址分别提供了私网和公网的连接地址，本文将介绍如何在控制台修改和释放PolarDB的连接地址。

修改连接地址

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**的**链接地址**区域，您可以通过以下任意一种方式来查看修改连接地址。
   - 方式一
     a. 单击**链接地址**区域右上角的图标切换视图。

b. 找到目标连接地址，单击**编辑**。

- 方式二：
  a. 单击目标集群地址右侧的**编辑配置**。
  b. 在弹出的对话框中的**网络信息**区域，单击**更多 > 编辑**。



5. 在弹出的对话框中，设置**公网**或**私网**连接地址前缀。

> 注意
> - 连接地址前缀需满足如下条件：
>   - 由小写字母、数字、中划线（-）组成，6~30个字符。
>   - 以字母开头，以数字或字母结尾。
> - 如果该连接地址正在启用SSL，修改会导致实例重启。
> - 如果该连接地址正在启用SSL，则修改后的连接地址全长不能超过64个字符。

6. 单击**确定**。

释放连接地址

> 警告
> - 释放连接地址前，请确认您的应用程序已使用新的连接地址进行访问。
> - 连接地址释放后无法恢复，只能重新**申请**，详情请参见查看或申请连接地址。

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**的**链接地址**区域，您可以通过以下任意一种方式来释放连接地址。
   - 方式一
     a. 单击**链接地址**区域右上角的图标切换视图。

b. 找到目标连接地址，单击**释放**。

- 方式二：
  a. 单击目标集群地址右侧的**编辑配置**。
  b. 在弹出的对话框中的**网络信息**区域，单击**更多 > 释放**。

> **说明** 仅公网的连接地址支持被释放。

5. 单击**确定**。

相关API

API	描述
DescribeDBClusterEndpoints	查询集群的地址信息。
CreateDBEndpointAddress	创建集群的公网地址。
ModifyDBEndpointAddress	修改集群默认访问地址。
DeleteDBEndpointAddress	释放集群地址。

6.3. 连接数据库集群

本文介绍如何通过DMS和客户端连接到PolarDB PostgreSQL引擎集群。

前提条件

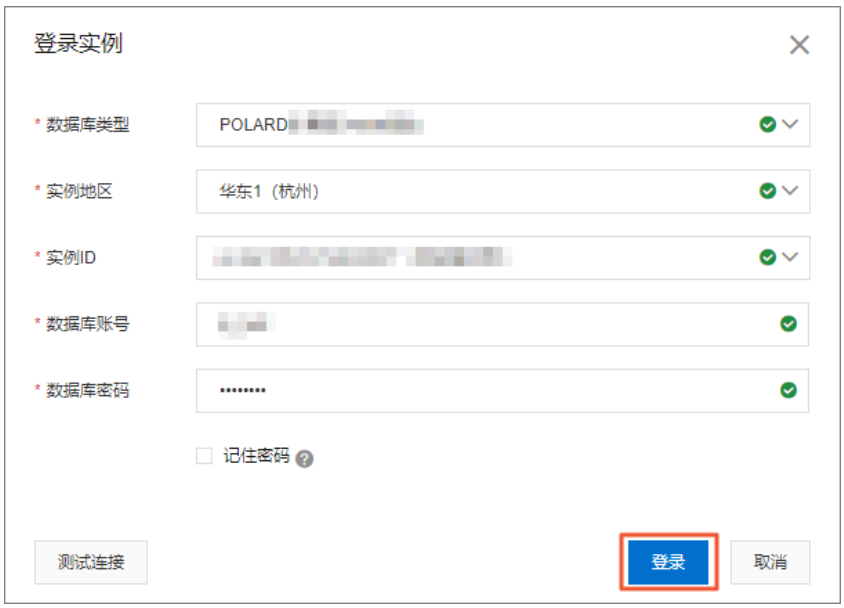
- 已创建数据库集群的高权限账号或普通账号。具体操作请参见创建数据库账号。
- 已经将需要访问PolarDB集群的主机IP地址添加到白名单，如何添加白名单请参见设置集群白名单。

通过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. 找到目标集群，单击集群ID。

4. 单击**基本信息**页面右上角的**登录数据库**。
5. 在弹出的对话框中，输入PolarDB集群中创建的**数据库账号**和**数据库密码**。



6. 单击**登录**。

> **说明** 如果您是首次使用DMS连接PolarDB集群，系统会提示您授权白名单，单击**确认**后即可完成授权。

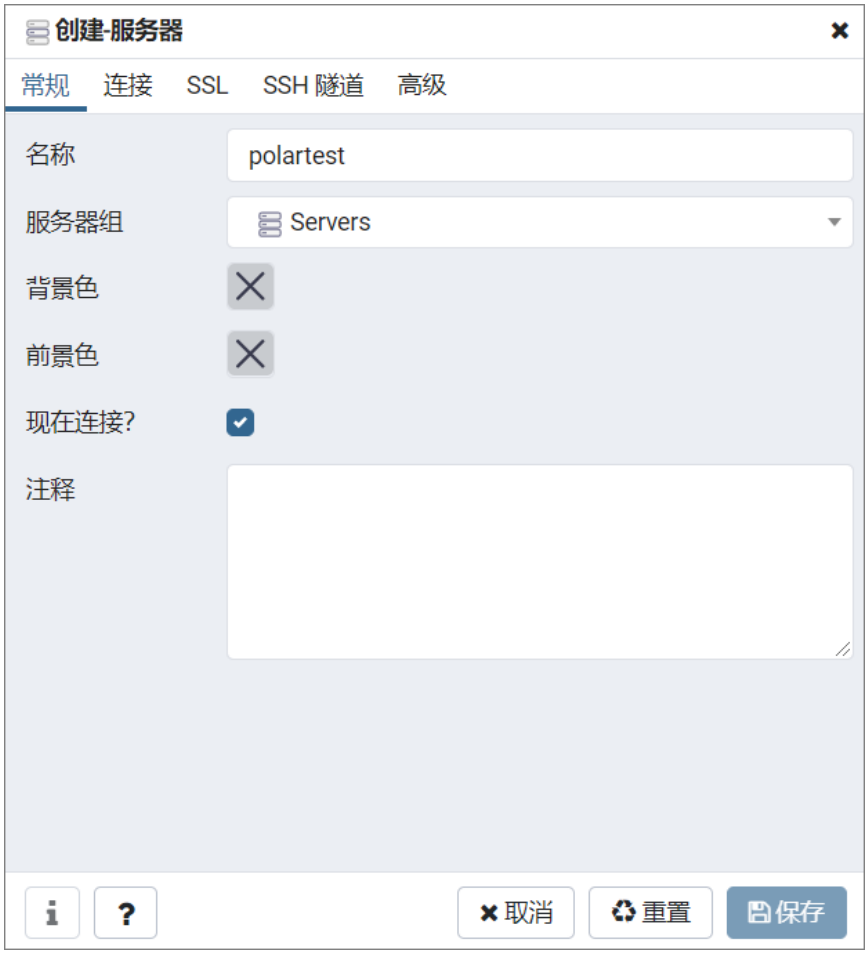
7. 登录后刷新DMS页面，在左侧导航栏中，单击**已登录实例**。
8. 找到目标数据库，双击目标数据库即可切换到目标数据库。

通过客户端连接PolarDB

您也可以通过pgAdmin 4客户端连接PolarDB数据库集群。

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，自定义服务器**名称**。

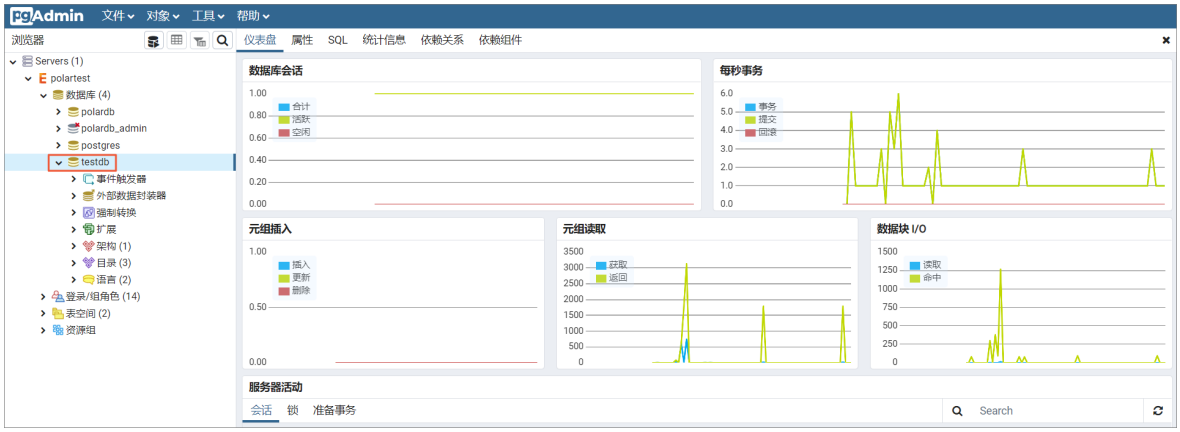


4. 选择Connection标签页，输入要连接的集群信息，参数说明如下。

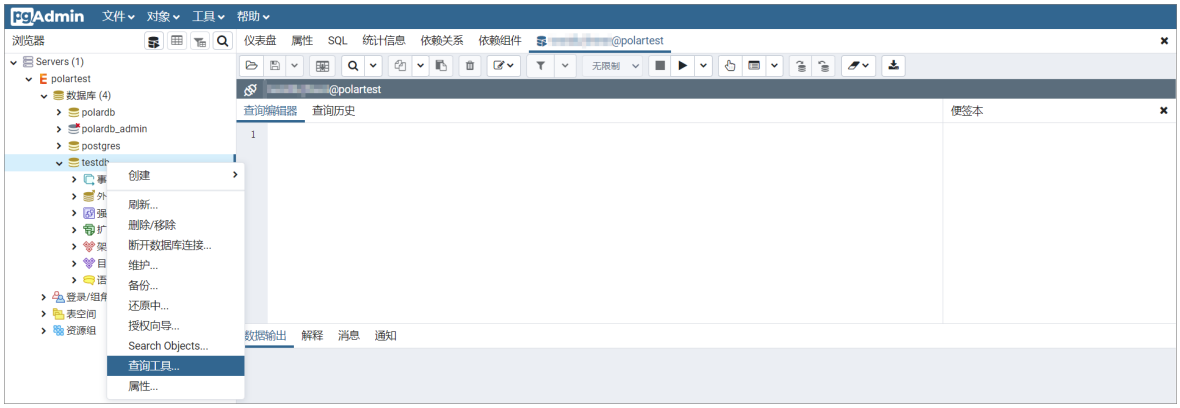
参数说明

参数	说明
主机名称/地址	输入PolarDB集群的连接地址。查看PolarDB集群的地址及端口信息的步骤如下： i. 登录PolarDB控制台。 ii. 在控制台左上角，选择集群所在地域。 iii. 单击目标集群ID。 iv. 在**链接地址**区域查看PolarDB地址。
端口	需输入PolarDB PostgreSQL引擎集群端口，默认为1921。
维护数据库	输入维护数据库，默认为**postgres**。
用户名	PolarDB集群的账号，创建账号请参见创建数据库账号。
密码	PolarDB集群账号所对应的密码。

5. 单击**保存**。
6. 若连接信息无误，单击目标数据库后出现类似如下界面，则表示连接成功。



7. 右键单击目标数据库，选择**查询工具...**，打开如下页面后，即可对数据库进行增删改查等操作。

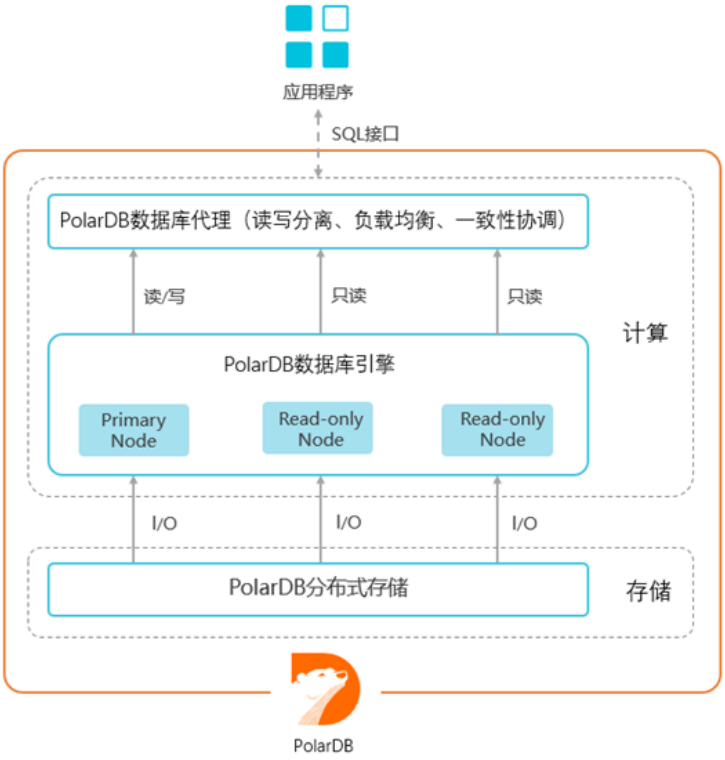


6.4. 集群地址

6.4.1. PolarDB数据库代理概述

本文为您介绍PolarDB数据库代理及其支持的相关功能。

PolarDB架构及PolarDB数据库代理介绍



PolarDB集群版是一个由多节点构成的数据库集群，包括一个主节点，多个只读节点。对外默认提供两个地址，分别为主地址和集群地址。其中，集群地址功能由PolarDB数据库代理提供，集群地址分为只读和可读可写两种读写模式，可读可写模式支持读写分离，只读模式支持按连接数负载。

读写分离

PolarDB集群版自带读写分离功能。应用程序只需连接一个集群地址，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（当前未完成的请求数）发往主节点或只读节点，详情请参见读写分离。

FAQ

- Q：为什么刚插入的语句，立即查的时候查不到？
  A：读写分离的架构下，主节点和只读节点之间复制会有延迟，但PolarDB支持会话一致性，即同一个会话内保证能读到之前的更新。
- Q：为什么只读节点没有压力？
  A：默认情况下事务中的请求都会路由到主节点，若是用Sysbench做压测，0.5版本的Sysbench可以加上 `--oltp-skip-trx=on` ，1.0版本的Sysbench可以加上 `--skip-trx=on` 去掉事务，若业务上因为事务较多导致只读库负载过低，可以提交工单开启读写分离下的事务拆分功能。
- Q：为什么某个节点的请求数比别的节点多？
  A：当前是根据负载来分发请求的，负载小的节点接收的请求数会更多。
- Q：是否支持0毫秒延迟的读取？
  A：PolarDB集群的主节点和只读节点在正常负载情况下，具有毫秒级的延迟，读写分离连接地址暂时不支持在数据写入后0毫秒的读取。如果要求0毫秒延迟的读取，可使用主地址（动态指向PolarDB主节点）将读写请求发给主节点。
- Q：新增的只读节点会自动加入到读写分离吗？
  A：新增只读节点之后新建的读写分离连接才会转发请求到该只读节点。若需要使新增只读节点之前建立的读写分离连接也转发请求到新增的只读节点，则您需要通过重启应用等操作断开该连接并重新建立连接。

相关API

API	描述
CreateDBEndpointAddress	创建PolarDB集群的公网地址。
CreateDBClusterEndpoint	创建PolarDB自定义集群地址。

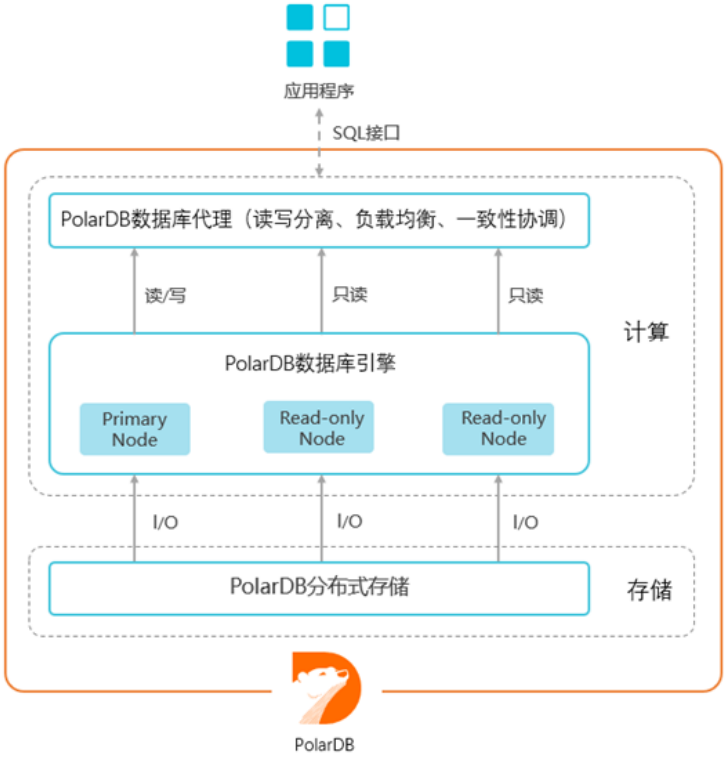
API	描述
DescribeDBClusterEndpoints	查询PolarDB集群的地址信息。
ModifyDBClusterEndpoint	修改PolarDB集群地址属性。
ModifyDBEndpointAddress	修改PolarDB集群的连接地址（如自定义集群地址）。
DeleteDBEndpointAddress	释放PolarDB集群地址（除了自定义集群地址的私网地址）。
DeleteDBClusterEndpoint	释放PolarDB自定义集群地址。

6.4.2. 一致性级别

PolarDB提供了最终一致性和会话读一致性两种一致性级别，满足您在不同场景下对一致性级别的要求。

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。



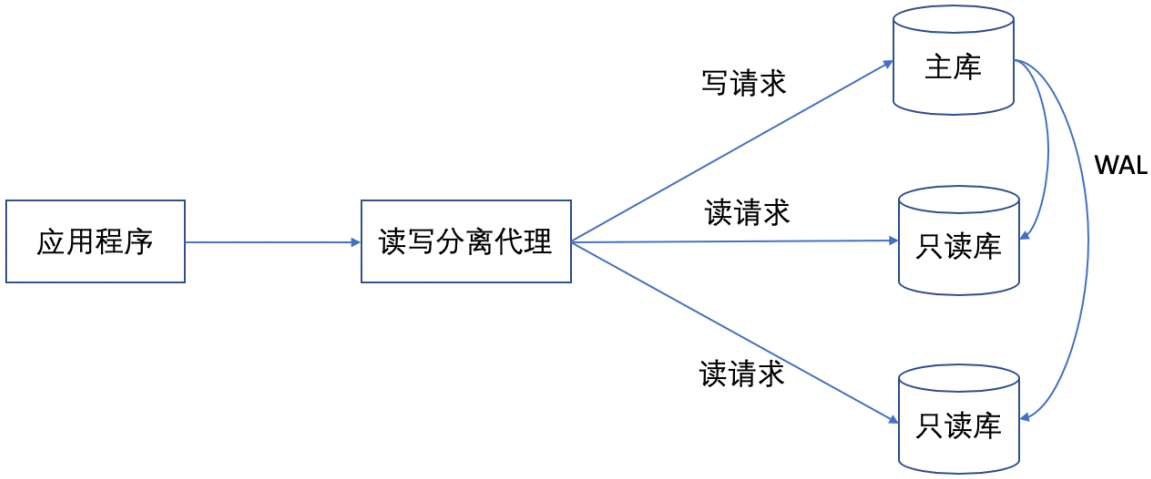
PostgreSQL读写分离解决和引入

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

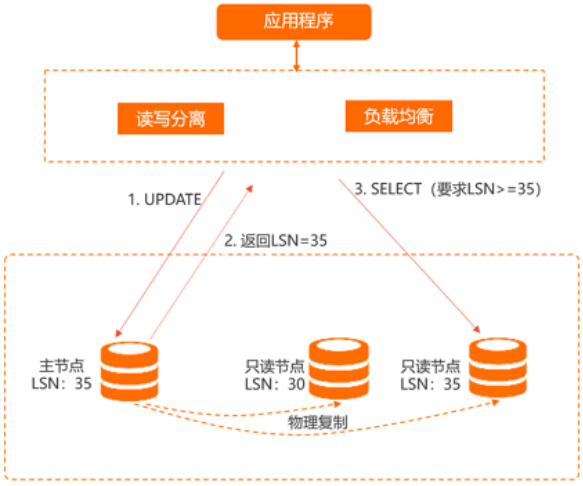
PolarDB是读写分离的架构，传统的读写分离都只提供最终一致性的保证，主从复制延迟会导致从不同节点查询到的结果不同，比如一个会话内连续执行以下QUERY：

```
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;
```

6.4.3. 读写分离

PolarDB PostgreSQL引擎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。
  含写操作的prepare语句只发到主库，相应的execute也只发到主库；纯读操作的prepare语句广播到所有节点，相应的execute根据负载均衡进行路由，从而实现查询请求的负载均衡。
- 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

详情请参见事务拆分。

高级选项-一致性级别

详情请参见一致性级别。

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
CreateDBEndpointAddress	创建PolarDB集群的公网地址。
DescribeDBClusterEndpoints	查询PolarDB集群的地址信息。
ModifyDBClusterEndpoint	修改PolarDB集群地址属性。
ModifyDBEndpointAddress	修改PolarDB集群默认访问地址前缀。
DeleteDBEndpointAddress	释放PolarDB集群地址（除了自定义集群地址的私网地址）。

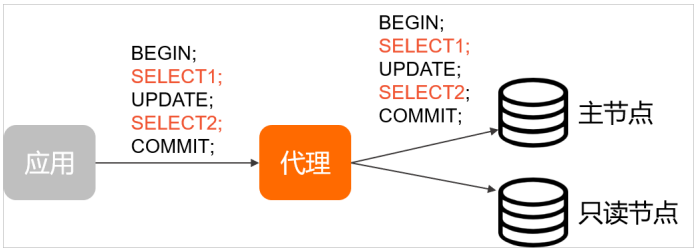
6.4.4. 事务拆分

本文将为您介绍PolarDB PostgreSQL引擎集群事务拆分的功能原理以及如何开启事务拆分。

背景信息

当您使用PolarDB PostgreSQL引擎可读可写模式集群地址时，读写请求会由代理（Proxy）分发到主节点和只读节点。为了保证一个会话连接中事务读写一致性，代理会将这个会话中所有在事务中的请求都发送到主节点。

例如，某些数据库客户端驱动（例如JDBC）默认将请求封装在事务中，因此应用的请求都会被发送到主节点，导致主节点压力大，而只读节点几乎没有压力，如下图所示。

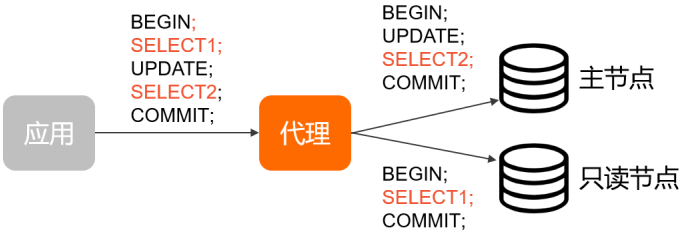


为了解决上述问题，PolarDB提供了事务拆分功能，旨在保证业务中读写一致性的前提下，将事务中读请求发送到只读节点，以减轻主节点的压力。

功能介绍

事务拆分基础服务

代理会将事务中第一个写请求前的读请求发送到只读节点，从而减轻主节点的负载，由于事务中未提交（COMMIT）的数据在只读节点上处于不可见的状态，为了保障事务中读写一致性，第一个写请求后的所有读写请求仍路由到主节点。如需开启事务拆分基础服务，请参见开启事务拆分。



功能优势

不需要改动应用的代码或配置就可以将事务中的读压力从主节点转移到只读节点，从而提高主节点的稳定性。

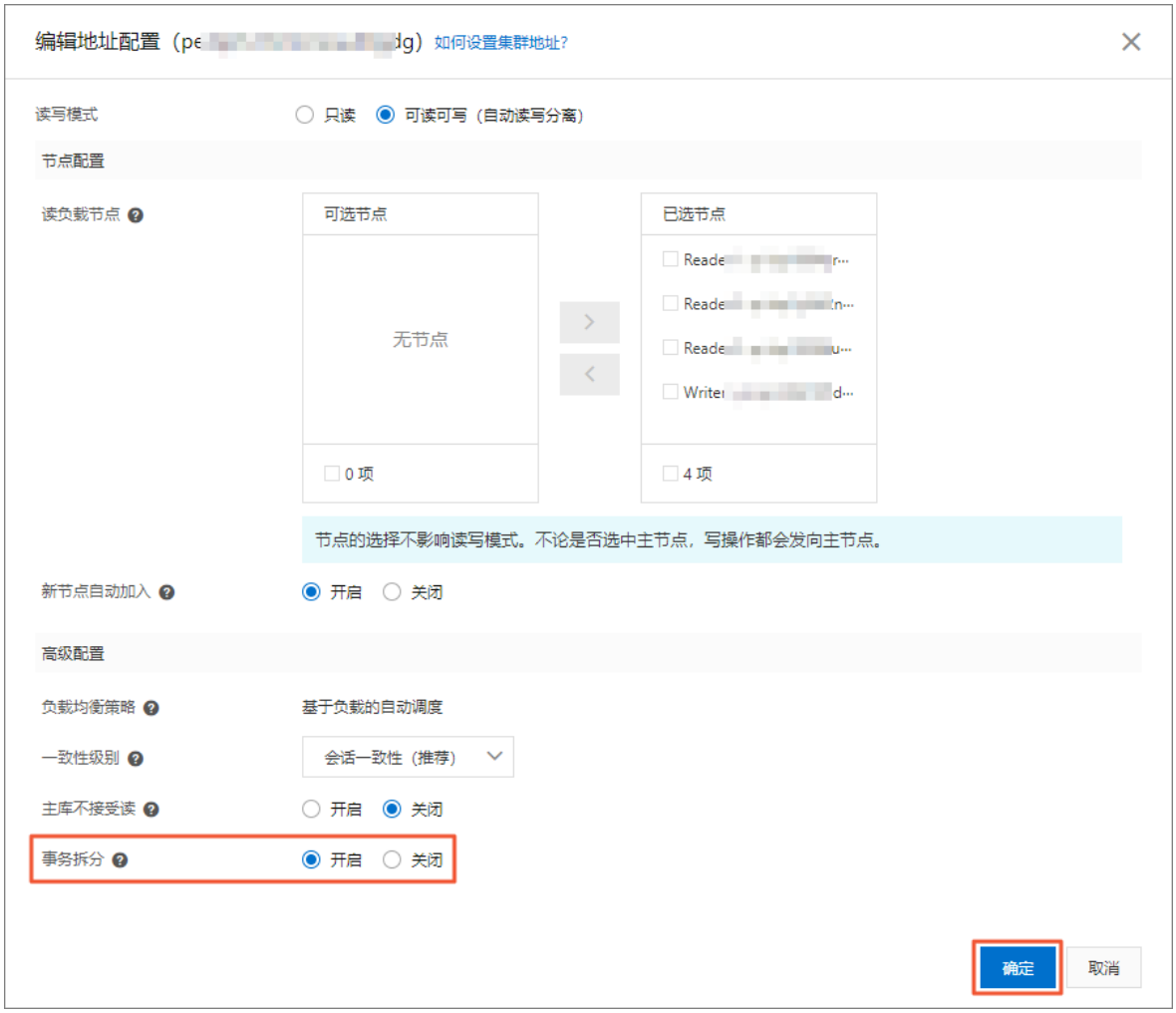
注意事项

- 仅支持对读已提交（Read Committed）事务隔离级别的事务拆分。
- 对于事务拆分基础服务，如果一致性级别不是**最终一致性**，只有只读节点与主节点数据同步成功后，事务中第一次写请求前的读请求才会发送到只读节点，否则依旧发送到主节点。关于一致性级别的具体信息，请参见一致性级别。

开启事务拆分

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.
5. 单击**事务拆分**右侧的**开启**。

> **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**。

相关API

API	描述
ModifyDBClusterEndpoint	修改PolarDB的集群地址属性，包括读写模式、新节点是否自动加入本地址、一致性级别、事务拆分、主库不接受读等。

6.4.5. 自定义函数与表的路由规则

数据库代理提供函数或表的自定义路由功能，通过此功能可以让自定义函数的读操作路由到只读库（原默认路由到主库），或者让自定义表读的操作路由到主库（原默认路由到只读库）。

前提条件

本章节所有操作需要使用高权限账号通过主地址连接到数据库中进行。

创建插件

请提交工单创建polar_proxy_utils插件。

插入记录

执行以下命令，插入记录。

```
polar_add_proxy_routing_strategy(_name, _type, rw_mode);
```

> **说明** 其中：
>
> - _name：需要添加的路由的表名或者函数名。
> - _type：标记此次添加的为表名或者函数名。t：表示表名；f：表示函数名。
> - rw_mode：涉及表名或者函数名的查询请求，要路由到主库还是只读库。w：表示路由到主库；r：表示路由到只读库。

例如，执行 `polar_add_proxy_routing_strategy('lol', 't', 'w');` 命令。

```
postgres=#
postgres=#
postgres=#
postgres=# select polar_list_proxy_routing_strategy();
 polar_list_proxy_routing_strategy
-----------------------------------
(0 rows)

postgres=# select polar_add_proxy_routing_strategy('lol', 't', 'w');
 polar_add_proxy_routing_strategy
-----------------------------------

(1 row)

postgres=# select polar_list_proxy_routing_strategy();
       polar_list_proxy_routing_strategy
-------------------------------------------------
 (lol,table,write,"2020-03-24 04:12:56.245526")
(1 row)
```

- 添加这条记录之前，执行 `select * from lol` 会路由到只读库。
- 添加这条记录之后，执行 `select * from lol` 就会路由到主库。

展示白名单信息

执行以下命令，展示白名单信息。

```
select polar_list_proxy_routing_strategy();
```

```
postgres=# select polar_list_proxy_routing_strategy();
       polar_list_proxy_routing_strategy
-------------------------------------------------
 (lol,table,write,"2020-03-24 04:12:56.245526")
 (abc,function,read,"2020-03-24 04:15:23.854227")
(2 rows)
```

删除一条白名单记录

执行以下命令，删除一条白名单记录。

```
select polar_delete_proxy_routing_strategy(_name, _type);
```

> **说明** 其中：
>
> - _name：需要删除的路由的表名或者函数名。
> - _type：标记此次删除的是表名还是函数名。t：表示表名；f：表示函数名。

例如，执行 `select polar_delete_proxy_routing_strategy('lol', 't');` 命令。

```
postgres=#
postgres=# select polar_list_proxy_routing_strategy();
        polar_list_proxy_routing_strategy
-------------------------------------------------
 (lol,table,write,"2020-03-24 04:12:56.245526")
 (abc,function,read,"2020-03-24 04:15:23.854227")
(2 rows)

postgres=# select polar_delete_proxy_routing_strategy('lol', 't');
 polar_delete_proxy_routing_strategy
-------------------------------------

(1 row)

postgres=# select polar_list_proxy_routing_strategy();
        polar_list_proxy_routing_strategy
-------------------------------------------------
 (abc,function,read,"2020-03-24 04:15:23.854227")
(1 row)
```

- 删除这条记录之前，执行 `select * from lol` 会路由到主库。
- 删除这条记录之后，执行 `select * from lol` 就会路由到只读库。

删除所有白名单记录

执行以下命令，删除所有白名单记录。

```
select polar_truncate_proxy_routing_strategy();
```

```
postgres=# select polar_list_proxy_routing_strategy();
        polar_list_proxy_routing_strategy
-------------------------------------------------
 (abc,function,read,"2020-03-24 04:15:23.854227")
 (lol,table,write,"2020-03-24 04:20:17.186641")
(2 rows)

postgres=# select polar_truncate_proxy_routing_strategy();
 polar_truncate_proxy_routing_strategy
---------------------------------------

(1 row)

postgres=# select polar_list_proxy_routing_strategy();
 polar_list_proxy_routing_strategy
-----------------------------------
(0 rows)
```

6.4.6. 创建自定义集群地址

本文将为您介绍如何配置数据库代理，包括如何开启读写分离、事务拆分和一致性级别等功能，您可以通过创建或修改集群地址进行配置。

创建自定义集群地址

您可以在创建自定义集群地址时，选择开启读写分离、事务拆分和一致性级别。

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。

集群配置表

配置项		说明
网络信息		PolarDB为每个集群地址默认提供了公网连接地址，若需要修改该地址或申请私网和经典网络连接地址，请参见修改连接地址和申请连接地址。
集群设置	**读写模式**	本地址的读写模式，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，只对新建连接生效，已有的连接保持原来的模式。
	地址名称	输入集群地址的名称。
服务节点	**可选节点**和**已选节点**	从左侧**可选节点**框内，选中想要加入本地址用于处理读请求的节点，单击图标，将其移动到右侧**已选节点**框中。 **说明** ○ **可选节点**包括主节点和所有只读节点。 ○ 节点的选择不影响读写模式。读写模式为**可读可写（自动读写分离）**时，无论是否选中主节点，写请求都只会发往主节点。 ○ PolarDB支持创建仅包含一个节点的集群地址，但当读写模式为**只读**时，不允许创建仅包含一个主节点的单节点集群地址。
	新节点自动加入	新增的节点是否要自动添加到该地址中。
负载均衡设置	**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。
	主库是否接受读	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。关于主库保护的更多介绍，请参见读写分离。 **说明** 仅**可读可写（自动读写分离）**模式下支持该配置。
	事务拆分	开启或关闭事务拆分。关于事务拆分的更多介绍，请参见事务拆分。 **说明** 仅当读写模式为**可读可写（自动读写分离）**时，支持该配置。
一致性设置	**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（弱）**和**会话一致性（中）**，详情请参见一致性级别。 ○ 读写模式为**只读**时，默认一致性级别为**最终一致性（弱）**且不可更改。 **说明** 一致性级别修改后对所有连接立即生效。

6. 单击**确定**。

修改集群地址

您可以在修改集群地址时，选择开启读写分离、事务拆分和一致性级别。

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，单击目标集群地址右侧的**编辑配置**。
5. 在**编辑地址配置**对话框内，您可以设置相关参数，参数详情请参见集群配置表。
6. 单击**确定**。

释放自定义集群地址

> 说明
> - 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。

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，单击目标集群地址右侧的**设置 > 释放**。
5. 在弹出的对话框中，单击**确定**。

相关API

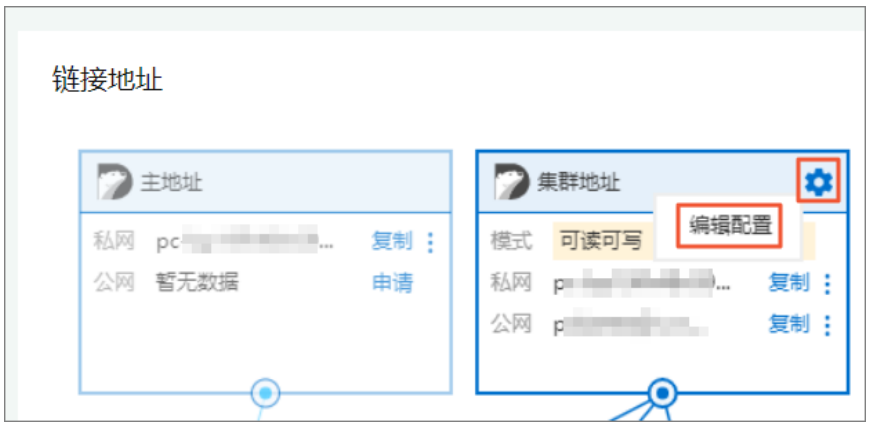
API	描述
CreateDBClusterEndpoint	创建自定义集群地址。
DescribeDBClusterEndpoints	查询集群地址。
DeleteDBClusterEndpoint	释放自定义集群地址。

6.4.7. 修改集群地址

您可以在PolarDB PostgreSQL引擎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。
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（弱）**和**会话一致性（中）**，详情请参见一致性级别。 ○ 读写模式为**只读**时，默认一致性级别为**最终一致性（弱）**且不可更改。
主库是否接受读	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
DescribeDBClusterEndpoints	查询集群地址。
ModifyDBClusterEndpoint	修改集群地址。

6.4.8. 释放自定义集群地址

本文介绍如何释放PolarDB PostgreSQL引擎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，找到目标集群地址，单击目标集群地址右侧的**设置 > 释放**。
5. 在弹出的对话框中，单击**确定**。

相关API

API	描述
DescribeDBClusterEndpoints	查询集群地址。
DeleteDBClusterEndpoint	释放自定义集群地址。

6.5. 私有域名

如果您之前是通过域名连接到数据库，当数据库迁移上云后，想要保留原来的数据库域名，可以通过私有域名功能进行绑定。

应用场景

PolarDB的每一个内网地址，均可以绑定一个私有域名，私有域名仅在当前地域内指定的VPC中生效。私有域名优先级比全球生效的域名优先级高，会优先解析。

例如，原来的数据库的域名是developer.aliyundoc.com，把数据库迁移至PolarDB集群，PolarDB集群的连接地址是image.developer.aliyundoc.com，为了保留原来的域名不变，可以创建一个私有域名将developer.aliyundoc.com绑定（CNAME）到image.developer.aliyundoc.com。绑定成功后在指定VPC中访问developer.aliyundoc.com就可以直接访问到PolarDB集群，如下图所示。

费用说明

PolarDB私有域名功能是通过PrivateZone管理的私有域名映射到PolarDB的私网地址来实现的，PrivateZone会收取少量费用，具体费用，请参见产品定价。

绑定私有域名

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**链接地址**区域，单击右上角图标切换视图。
5. 单击目标私网地址右侧的**绑定私有域名**。

6. 在**绑定私有域名**对话框中，输入私有域名的前缀和后缀。

私有域名格式为 `<前缀>.<后缀>` ，详细信息请参见下表。

配置	说明
私有域名前缀	私有域名前缀由小写字母、数字、中划线（-）中的至少一种组成；以字母开头，以数字或字母结尾；6~30个字符。
私有域名后缀（Zone）	您可以下拉选择已有Zone或手动输入新的Zone，更多关于Zone的信息，请参见PrivateZone。 **说明** ◦ 若PolarDB所在的VPC不在配置的Zone中，系统将会自动绑定VPC和Zone。 ◦ 您可以在PrivateZone控制台查看和管理Zone。

说明 绑定私有域名时，系统将会自动创建一个服务关联角色（AliyunServiceRoleForPolarDB），具体信息，请参见PolarDB服务关联角色。

7. 单击**确定**。
8. 在**绑定私有域名**对话框中，再次确认域名信息无误后，单击**确定**即可。

相关API

API	描述
ModifyDBEndpointAddress	该接口用于修改PolarDB集群的连接地址，包括主地址、默认集群地址、自定义集群地址和私有域名。

7.集群

7.1. 创建数据库集群

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

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十五个只读节点（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 PostgreSQL引擎仅支持专有网络VPC（Virtual Private Cloud）。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，详情请参见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**创建新集群**。
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**：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。

参数	说明
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。

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集群。 ○ **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。 **说明** 其他选项用于创建其它引擎的数据库。
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。
网络类型	固定为**VPC**专有网络，无需选择。
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，详情请参见创建和管理专有网络。
兼容性	○ **MySQL 8.0**（与MySQL 8.0完全兼容），原生支持并行查询，特定场景下性能提升十倍，详情请参见并行查询（Parallel Query）。 ○ **MySQL 5.7**（与MySQL 5.7完全兼容）。 ○ **MySQL 5.6**（与MySQL 5.6完全兼容）。 ○ **PostgreSQL 11**（与PostgreSQL 11 完全兼容）。 ○ **兼容Oracle语法**（高度兼容Oracle语法）。 **说明** 当前华北1（青岛）、美国（弗吉尼亚）、英国（伦敦）和澳大利亚（悉尼）四个地域暂不支持PostgreSQL 11和兼容Oracle语法。
系列	默认为**集群版（2-16个节点）【推荐】**。
节点规格	按需选择。所有PolarDB节点均为独享型，性能稳定可靠。 更多关于计算节点规格的详情，请参见规格与定价。

参数	说明
节点个数	如果源集群系列为**集群版（2-16个节点）【推荐】**，系统将默认创建规格相同的两个节点（一主一只读），无需选择。 **说明** 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，请参见产品架构。
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，详情请参见规格与定价。 **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** TDE功能开启后不可关闭。
集群名称	输入集群名称，集群名称需满足如下要求： ○ 不能以 `http://` 或 `https://` 开头。 ○ 长度为2~256个字符。 如果留空，系统将为自动生成一个集群名称，创建集群后还可以修改。
资源组	从已创建资源组中选择一个目标资源组。 **说明** 资源组是在单个云账号下将一组相关资源进行统一管理的容器，一个资源只能归属于一个资源组，详情请参见RAM资源分组与授权。

5. 设置**购买数量**后，单击**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，阅读并选中服务协议，单击**立即开通**即可。

   开通成功后，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**创建中**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**运行中**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，详情请参见。

下一步

设置集群白名单

相关API

API	描述
CreateDBCluster	创建数据库集群。
DescribeDBClusters	查看集群列表。

API	描述
DescribeDBClusterAttribute	查看指定PolarDB集群的详细属性。
DescribeAutoRenewAttribute	查询PolarDB包年包月集群自动续费状态。
ModifyAutoRenewAttribute	设置PolarDB包年包月集群自动续费状态。

7.2. 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，如需降级请参见手动变配。

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手动变配。
- 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。

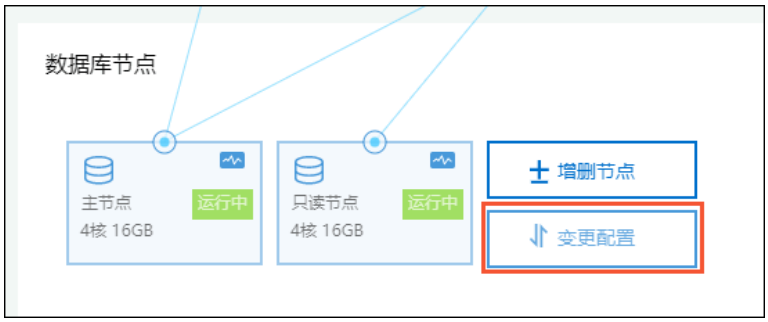
操作步骤

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**变更配置（包年包月）**页面：
   - 单击目标集群**操作**栏中的**变更配置**。

   - 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**变更配置**。



5. 在**变更配置（包年包月）**页面，勾选**临时升配**，单击**确定**。

> **说明** 仅包年包月集群支持**临时升配**。

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**。

7.3. 变更配置

您可以根据业务需求变更集群的配置。新规格会立即开始生效（每个节点需要5到10分钟）。

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：增加或删除只读节点。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。

变更配置的费用说明

详情请参见变更配置费用说明。

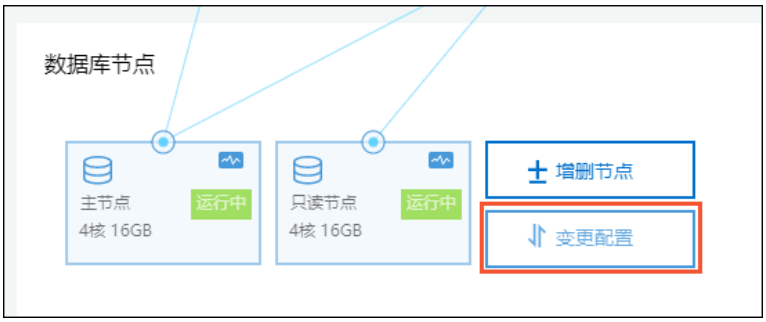
注意事项

- 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，每个连接地址都会有不超过30秒的连接闪断，建议您在业务低谷期执行变更，并确保应用具备自动重连机制。

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. 您可以按照如下两种方式进入**变更配置（包年包月）**页面。
   - 找到目标集群，在**操作**列单击**变更配置**。

   - 找到目标集群，单击集群ID，在**基本信息**页面下方，单击**变更配置**。



4. 勾选**升配**或**降配**，单击**确定**。

> **说明** 仅包年包月集群支持**临时升配**，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**立即购买**。
7. 在**支付**页面中，确认订单信息，单击**购买**。

> **说明** 规格变更预计需要10分钟生效。

相关API

API	描述
ModifyDBNodeClass	变更PolarDB集群节点规格。

7.4. 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。

背景信息

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。

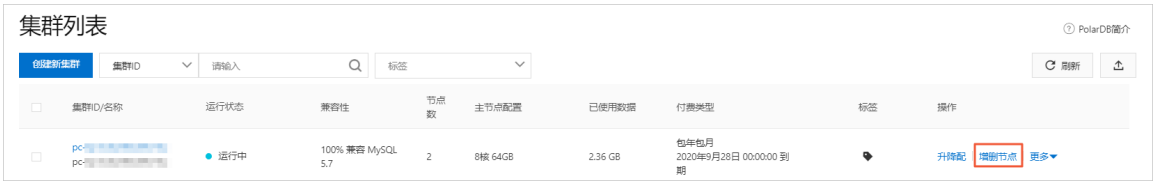
> 说明
>
> - 包年包月和按小时付费的只读节点都可以随时释放，释放后会退款或停止计费。
> - 增加节点仅收取节点规格的费用（详情请参见计费项概览），存储费用仍然按实际使用量收费，与节点数量无关。

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。
- 增加一个只读节点预计耗时5分钟。增加节点的过程中，对数据库无任何影响。
- 删除只读节点时，该节点上的连接会发生闪断，其他节点不受影响。建议您在业务低谷期执行删除节点操作，并确保应用具备自动重连机制。

增加只读节点

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 您可以按照如下两种方式进入**增删节点向导**页面。
   - 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，进入**基本信息**页面，在**数据库节点**区域单击**增删节点**。

4. 选中**增加节点**并单击**确定**。

5. 单击**+增加一个节点**，勾选服务协议，单击**立即购买**。
6. 在**支付**页面确认订单信息，单击**支付**。

删除只读节点

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**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. 选中**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**立即购买**。

相关API

API	描述
CreateDBNodes	增加PolarDB集群节点。
ModifyDBNodesClass	独立变更PolarDB集群单个节点的规格。
ModifyDBNodeClass	变更PolarDB集群节点规格。
RestartDBNode	重启PolarDB集群节点。
DeleteDBNodes	删除PolarDB集群节点。

7.5. 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，为保障云原生关系型数据库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，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. 找到目标集群，单击集群ID。
4. 在**基本信息**中的**可维护窗口**后，单击**编辑**。

5. 在弹出的对话框中，选中集群的可维护窗口时间，单击**确定**。

> **说明**
> - 为保障云原生关系型数据库PolarDB的稳定性，后端系统会不定期对集群进行维护操作。建议将可维护窗口设置在业务低峰期，以免维护过程中可能对业务造成影响。
> - 在可维护窗口内，集群可能会发生1到2次连接闪断，请确保应用程序具有自动重连机制。

相关API

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

7.6. 重启节点

PolarDB提供了重启节点功能，您可以在控制台上手动重启节点解决数据库连接和性能问题。

注意事项

- 重启只读节点之后新建的读写分离连接会转发请求到该只读节点。重启只读节点之前建立的读写分离连接不会转发请求到重启后的只读节点，您可以重启应用断开该读写分离连接并重新建立连接，请求将转发到重启后的只读节点。
- 重启节点的过程中，可能会出现1分钟以内的连接闪断。建议您在业务低谷期执行重启节点操作并确保应用具备重连机制。
- 重启节点的时间长短跟您业务的数据量有关，可能需要几个小时，请谨慎操作。

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. 找到目标集群，单击集群ID。
4. 在**基本信息**页的**数据库节点**区域，单击右上角图标切换视图。
5. 找到目标节点，单击右侧**操作**栏中的**重启**。

6. 在弹出的对话框中，单击**确定**。

相关API

API	描述
RestartDBNode	重启数据库节点。

7.7. 释放集群

您可以根据业务需求手动释放后付费（按小时付费）的集群，本文介绍如何手动释放PolarDB PostgreSQL引擎集群。

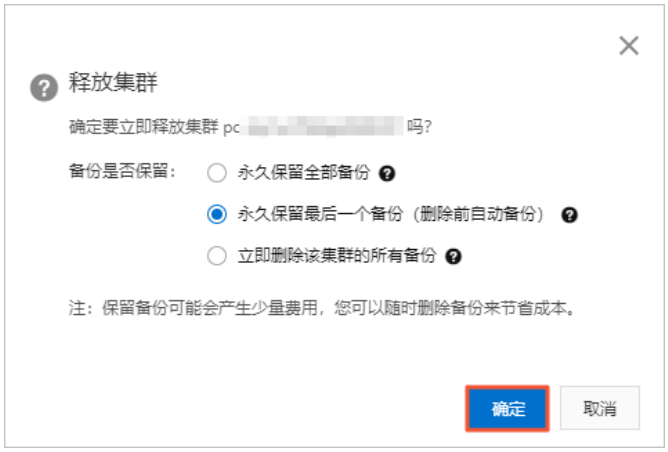
注意事项

- **包年包月**（也称预付费）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**为**运行中**的集群才能被手动释放。
- 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，请参见增加或删除节点。

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，单击右侧**操作**栏的**更多 > 释放**。

4. 在弹出的**释放集群**对话框中，选择备份保留策略后，单击**确定**。



备份是否保留	说明
删除集群时会自动备份，并永久保留该集群的所有备份集。	删除集群时保留所有备份。
删除集群时会自动备份，永久保留该备份集。	删除集群时保留最后一个备份。

备份是否保留	说明
删除集群时，立即删除该集群的所有备份集。	删除集群时不保留任何备份。 警告 选择该策略会导致集群删除后将无法恢复。

> 说明
> - 如果您选择了**删除集群时会自动备份，并永久保留该集群的所有备份集。**或**删除集群时会自动备份，永久保留该备份集。**策略，删除PolarDB集群时，系统会主动发起一次备份，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，更多内容请参见恢复已释放的集群。

相关API

API	描述
DescribeDBClusters	查看PolarDB集群列表。
DeleteDBCluster	删除PolarDB集群。

7.8. 集群保护锁

如果您的集群承载了关键业务，建议为按量付费集群开启集群保护锁，防止手动释放按量付费集群，可以有效避免因操作疏忽、团队成员沟通不及时等原因造成不可挽回的后果。本文为您介绍如何开启或关闭集群保护锁。

前提条件

集群的付费类型为按量付费。

注意事项

- 开启集群保护锁的集群不允许转为包年包月的集群。
- 集群保护锁不能阻止因合理原因自动执行的释放行为，包括但不限于：
  - 账号欠费导致集群停机超过8天，集群被自动释放的行为。
  - 集群存在安全合规风险，被停止或释放的行为。

开启集群保护锁

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 您可以按照以下两种方式中的任意一种开启集群保护锁。
   - 方式一：
     在**集群列表**页找到目标集群，单击右侧**操作**栏的**更多 > 添加集群保护锁**。

   - 方式二：
     a. 在**集群列表**页面，单击目标集群。

b. 在**基本信息**页面，单击**集群保护锁**右侧的**开启**。

4. 在弹出的对话框中，单击**确定**。

关闭集群保护锁

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 您可以按照以下两种方式中的任意一种关闭集群保护锁。
   - 方式一：
     在**集群列表**页找到目标集群，单击右侧**操作**栏的**更多 > 释放集群保护锁**。

   - 方式二：
     a. 在**集群列表**页面，单击目标集群。
     b. 在**基本信息**页面，单击**集群保护锁**右侧的**关闭**。

4. 在弹出的对话框中，单击**确定**。

查看集群保护锁状态

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 在**集群列表**页面，单击目标集群。
4. 在**基本信息**页面，查看**集群保护锁**的状态。

相关API

API	描述
ModifyDBClusterDeletion	开启或关闭集群保护锁。

7.9. 自动/手动主备切换

当系统发生故障时，PolarDB集群会自动进行主备切换。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。

注意事项

不论是自动切换还是手动切换，切换过程中，都可能会出现30秒左右的闪断，因此切换前请务必确保应用具备重连机制。

自动主备切换

PolarDB采用双活（Active-Active）的高可用集群架构。当系统发生故障时，可读写的主节点和只读节点之间会自动进行故障切换（Failover），系统自动选举新的主节点。

集群中每个节点都有一个故障切换（Failover）优先级，该优先级决定了故障切换时每个节点被选举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举为主节点。

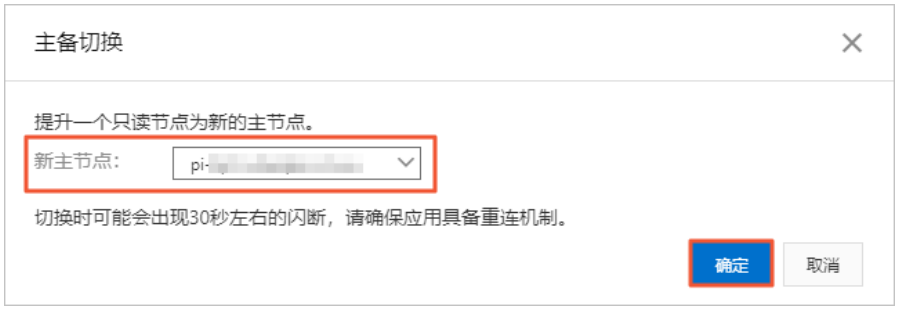
自动选取主节点按以下步骤进行：

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。

手动主备切换

1. 登录PolarDB管理控制台。
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**页的**数据库节点**区域，单击右上角图标切换视图。
5. 单击**主备切换**。

6. 在弹出的对话框中，从**新主节点**列表中选择目标节点，单击**确定**。



相关API

API	描述
FailoverDBCluster	为PolarDB集群进行手动主备切换，您可以指定一个只读节点为新的主节点。

7.10. 多可用区部署和更换主可用区

PolarDB PostgreSQL引擎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。

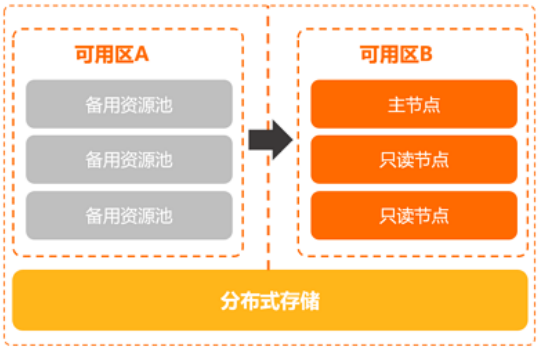
查看集群所属可用区

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. 找到目标集群，单击集群ID。
4. 在**基本信息**页面，单击**更换主可用区**。

5. 在弹出的对话框中，选择**目标可用区**和**目标交换机**，并根据业务需要选择**生效时间**。

> **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。

8.账号

8.1. 账号概述

本文介绍阿里云控制台账号和PolarDB数据库集群账号的相关内容。

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。更多信息请参见创建数据库账号。

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。

相关API

API	描述
CreateAccount	创建账号。
DescribeAccounts	查看账号列表。
ModifyAccountDescription	修改账号备注。
ModifyAccountPassword	修改账号密码。
DeleteAccount	删除账号。

8.2. 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**立即注册**。
- 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。

- 子账号的登录入口。

8.3. 创建和管理RAM用户

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则需要创建RAM用户。本文介绍如何创建RAM用户并为RAM用户授权以及RAM用户如何登录控制台。

前提条件

登录阿里云账号或有RAM权限的子账号。

- 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` 。

操作步骤

- 创建RAM用户，请参见创建RAM用户。
- 在用户页面下为RAM用户授权，请参见在用户页面为RAM用户授权。
- 在授权页面下为RAM用户授权，请参见在授权页面为RAM用户授权。
- 使用RAM用户登录控制台，请参见RAM用户登录阿里云控制台。

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。

8.4. 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。

参数	说明
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。
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

查看或申请连接地址

相关API

API	描述
CreateAccount	创建账号。
DescribeAccounts	查看账号列表。
ModifyAccountDescription	修改账号备注。
ModifyAccountPassword	修改账号密码。
DeleteAccount	删除账号。

8.5. 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
CreateAccount	创建账号。
DescribeAccounts	查看账号列表。
ModifyAccountDescription	修改账号备注。
ModifyAccountPassword	修改账号密码。
DeleteAccount	删除账号。

9.数据库

本文为您介绍如何创建和删除PolarDB PostgreSQL引擎数据库。

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。 ◦ 数据库名称在集群内必须是唯一的。 **说明** 请勿使用保留关键字作为数据库名称，如 `test` 等。
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。

参数	说明
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
CreateDatabase	创建数据库
DescribeDatabases	查看数据库列表信息
ModifyDBDescription	修改数据库描述
DeleteDatabase	删除数据库

10.备份与恢复

10.1. 概述

可靠的备份功能可以有效防止数据丢失，PolarDB PostgreSQL引擎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引擎集群时，您还可以选择保留备份数据。

数据备份

数据备份按照存储位置可分为一级备份和二级备份。

备份存储位置	是否默认开启	保留时长	特点	查看备份大小
一级备份（数据备份）	是	3~14天。	• 一级备份采用ROW（Redirect-on-Write）快照的方式，直接保存在PolarDB分布式存储系统上。每次保存时，一级备份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。 • PolarDB集群备份和恢复功能均采用多线程并行处理，并通过其它技术创新，10分钟内即可完成从备份集（快照）恢复到一个新的集群。 **说明** 一级备份默认开启，无法关闭。	一级备份（快照）总大小如下图所示。  **说明** PolarDB集群**一级备份（快照）总大小**是所有一级备份独占的物理空间之和（即上图中①所示），而非逻辑数据大小之和（即上图中②所示），PolarDB集群的数据与多个一级备份（快照）会复用相同的物理数据块，在计费时只会计算一次。更多关于备份的问题，请参见常见问题。
二级备份（数据备份）	否	• 30~7300天。 • 开启**删除集群前永久保留**来永久保存。	• 二级备份是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。保存成本较低，但使用二级备份恢复数据的速度较慢。 • 开启二级备份后，若一级备份超出您设置的保留时间，将会被自动转存为二级备份，转存速度约为150 MB/秒。 **说明** 若一级备份未能在下一个一级备份开始转存前完成，则下一个一级备份将会被直接删除而不会被转存为二级备份。例如将PolarDB集群的一级备份的备份时间设置为每日凌晨1点，保留时间为24小时，PolarDB集群在1月1号凌晨1点生成一级备份A，2号凌晨生成一级备份B，备份A在2号凌晨1点超过保留时间并开始转存为二级备份，由于该备份文件较大转存时间较长，到3号凌晨1点时该转存任务仍未完成，则此时备份B在3号凌晨1点到期后将会被直接删除而不会转存为二级备份。	二级备份大小如下图所示，二级备份总大小即为每个二级备份文件大小之和。 

物理日志备份

- 特点

物理日志备份通过实时并行上传数据库Redo日志文件到OSS来达到备份的目的。日志备份默认开启，最短保留时间为3天，最长保留时间为7300天。您也可以通过开启**删除集群前永久保留**功能永久保存。

> **说明** 日志备份默认开启，无法关闭。

借助日志备份可以实现任意时间点的一致性备份：通过一个完整的数据全量备份（快照）以及后续一段时间的日志备份，就可以将PolarDB集群恢复到任意时间点（Point-In-Time Recovery，简称PITR），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。恢复到任意时间点时，应用物理日志的恢复速度大概是20~70秒/GB，整个恢复时间是备份集（快照）恢复时间以及应用物理日志恢复时间之和。

- 查看备份大小
  日志备份大小如下图所示，日志备份总大小即为每个日志备份文件大小之和。

10.2. 费用说明

本文介绍备份与恢复功能的费用说明。
备份与恢复功能均免费使用，但备份文件会占用一定的存储空间，PolarDB根据备份文件（数据和日志）的存储容量和保存时长会收取一定的费用。

产品定价

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

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。
		• 正常按量付费：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。
		  ◦ 每小时单价请参见价格表。
		  ◦ 您可以在如下位置查看**一级备份（快照）总大小**：
		 
		
		说明 PolarDB集群**一级备份（快照）总大小**是所有一级备份独占的物理空间之和（即上图中①所示），而非逻辑数据大小之和（即上图中②所示），PolarDB集群的数据与多个一级备份（快照）会复用相同的物理数据块，在计费时只会计算一次。更多关于备份的问题，请参见常见问题。
		示例：一级备份（快照）总大小为700 GB，数据库存储用量为1000 GB，那么每小时费用为0.6元。
		计算公式：[700 GB-（1000 GBx50%）]x0.003元/GB/小时=0.6元/小时
		• 使用存储包。
		一级备份中超过免费额度的空间用量，除支持正常按量付费，现新增支持通过存储包进行抵扣。
		购买存储包后，若抵扣完账号下所有PolarDB集群的存储空间用量后存储包容量仍有剩余，将自动按照1：1的比例（即1 GB存储包容量可抵扣1 GB一级备份用量）抵扣一级备份中超过免费额度的部分，直至抵扣完存储包中的所有容量。
		若存储包中剩余容量不足以抵扣一级备份的空间用量，超出部分将正常按量付费。更多详情，请参见使用存储包抵扣。

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1000 GB，那么每小时费用为0.21元。 计算公式：1000 GB x 0.00021元/GB/小时=0.21元/小时
日志备份	100 GB	每小时费用 =（日志备份总大小 - 100 GB） x 每小时价格 示例：日志备份总大小为1000 GB，那么每小时费用为0.189元。 计算公式：（1000 GB - 100 GB）x 0.00021元/GB/小时=0.189元/小时

10.3. 备份操作说明

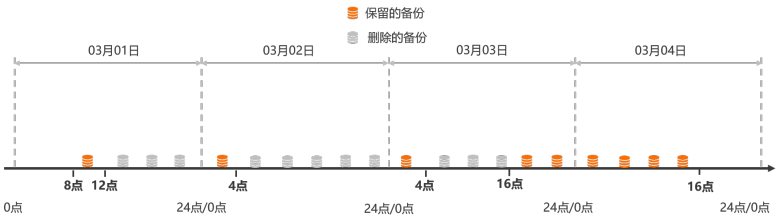
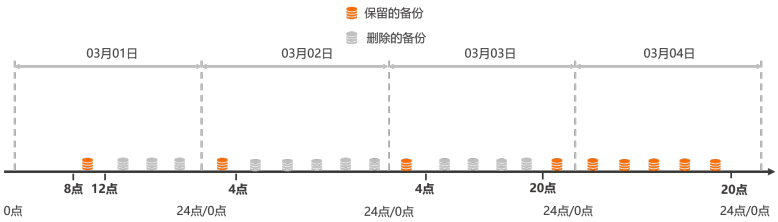
10.3.1. 备份策略设置

PolarDB支持数据备份和Redo日志备份。数据备份即将某个时间点上集群的全量数据生成一个备份集（快照），Redo日志备份即记录生成备份集后的增量数据。您可设置数据备份和Redo日志备份的备份策略，如数据自动备份的频率、数据备份文件保存时长、存储位置以及日志备份文件保存时长等。

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. 在左侧导航栏中，选择**配置与管理 > 备份恢复**。
5. 在**备份策略设置**页签，单击**编辑**。
6. 在**备份策略设置**对话框中，设置**数据备份策略**、**日志备份策略**和**通用备份策略**相关参数。

- **数据备份策略**
  数据备份策略包括自动备份的频率，以及自动备份和手动备份产生的备份文件的存储位置和保留时长。

参数名称	参数说明
数据备份频率	即自动备份的频率，可选择**常规备份（按备份周期）**或**高频备份**。 ■ **常规备份（按备份周期）**：自动备份默认每日1次，你可以设置数据自动备份的周期和自动备份开始时间。 **说明** ■ 出于安全考虑，常规备份的频率为每周至少2次。 ■ 备份文件不可删除。 ■ **高频备份**：PolarDB支持最近24小时增强保护功能，能够缩短备份周期，增加备份密度，从而提升恢复速度。设置备份频率，目前支持**最近24小时，每2小时备份一次**、**最近24小时，每3小时备份一次**或**最近24小时，每4小时备份一次**。 最近24小时增强保护功能开启后，备份完成时间在24小时内的备份会全部保留。超过24小时的，系统将仅保留每日00:00点后完成的第一个备份，其他均被删除。 假设您在3月1日8:00设置了**每4小时创建一个备份**，那么系统将自动在2小时内（即3月1日8:00~12:00）完成一个备份，并以每4小时一个备份的频率持续创建。 假设当前时间为3月4日16:00，系统将保留如下备份： ■ 最近24小时内（3月3日16:00~3月4日16:00）完成的备份。 ■ 3月3日0:00~4:00间完成的备份。 ■ 3月2日0:00~4:00间完成的备份。 ■ 3月1日8:00~12:00间完成的备份  那么4小时后，即3月4日20:00，系统将保留如下备份： ■ 最近24小时内（3月3日20:00~3月4日20:00）完成的备份。 ■ 3月3日0:00~4:00间完成的备份。 ■ 3月2日0:00~4:00间完成的备份。 ■ 3月1日8:00~12:00间完成的备份 

参数名称	参数说明
数据备份保留	自动备份和手动备份产生的数据备份文件的存储位置和保留时长。 数据备份文件的存储位置可选择**一级备份**和**二级备份**两种方式。关于两种方式的详情，可参见数据备份。 ■ **一级备份**：设置一级备份保留时间。 **说明** ■ 一级备份默认开启且默认保留时间为7天。 ■ 最短保留时间为3天，最长保留时间为14天。 ■ **二级备份**：开启或关闭二级备份。 **说明** ■ 二级备份默认关闭。开启二级备份会产生额外的费用，删除备份文件可节省成本。二级备份费用详情，请参见备份存储（超出免费额度）计费规则。 ■ 最短保留时间为30天，最长保留时间为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。

- **日志备份策略**
  Redo日志备份策略包括Redo日志的保留时长。

参数名称	参数说明
日志备份保留	设置日志备份的保留时长。 **说明** ■ 日志备份默认开启，无法关闭，默认保留时间为7天。 ■ 最短保留时间为3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。

- **通用备份策略**
  此外，您还可以设置删除集群时的备份文件保留策略。

参数名称	参数说明

参数名称	参数说明
集群备份保留	设置删除集群时的备份保留策略。 ■ **删除集群时会自动备份，并永久保留该集群的所有备份集。**：删除集群时保留所有备份。 ■ **删除集群时会自动备份，永久保留该备份集。**：删除集群时保留最后一个备份。 ■ **删除集群时，立即删除该集群的所有备份集。**：删除集群时不保留任何备份。 说明 ■ 如果您选择了**删除集群时会自动备份，并永久保留该集群的所有备份集。**或**删除集群时会自动备份，永久保留该备份集。**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，更多内容请参见恢复已释放的集群。

相关API

API	描述
DescribeBackupPolicy	查询PolarDB集群备份策略。
ModifyBackupPolicy	修改PolarDB集群备份策略。

10.3.2. 备份方式一：自动备份

自动备份默认开启，创建新集群后，PolarDB会默认按照每日备份一次的频率，自动进行数据备份。您可以根据业务需求，在控制台配置自动备份的频率、备份保存时长等参数。

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. 在左侧导航栏中，选择**配置与管理 > 备份恢复**。
5. 单击**备份策略设置**。
6. 在**备份策略设置**页，单击**编辑**。在弹出的窗口中，设置以下参数。

参数	说明
数据备份频率	选择**常规备份**或**高频备份**。 ○ **常规备份**：设置数据自动备份的周期和自动备份开始时间。 说明 出于安全考虑，自动备份的频率为每周至少2次。 ○ **高频备份**：设置备份频率，目前支持**最近24小时，每2小时备份一次**、**最近24小时，每3小时备份一次**或**最近24小时，每4小时备份一次**。

参数	说明
数据备份保留	设置**一级备份**和**二级备份**。 ○ **一级备份**：设置一级备份保留时间。 说明 ■ 一级备份默认开启且默认保留时间为7天。 ■ 一级备份最短保留时间为3天，最长保留时间为14天。 ○ **二级备份**：开启或关闭二级备份。 说明 ■ 二级备份默认关闭。开启二级备份会产生额外的费用，删除备份文件可节省成本。二级备份费用详情，请参见备份存储（超出免费额度）计费规则。 ■ 最短保留时间为30天，最长保留时间为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。

7. 完成备份策略设置后，单击**确定**。

相关API

API	描述
CreateBackup	创建PolarDB集群全量快照备份。
DescribeBackups	查询PolarDB集群备份信息。
DeleteBackup	删除PolarDB集群备份。

10.3.3. 备份方式二：手动备份

手动备份即主动备份，您可以根据业务需求随时进行手动备份，达到即时备份的效果，确保数据可靠性。本文介绍手动备份的相关操作步骤。

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。
> - 手动备份的备份文件可以删除，且删除后的备份文件不能恢复，请谨慎操作。

相关API

API	描述
CreateBackup	创建PolarDB集群全量快照备份。
DescribeBackups	查询PolarDB集群备份信息。
DeleteBackup	删除PolarDB集群备份。

10.4. 恢复操作说明

10.4.1. 恢复方式一：按时间点恢复

PolarDB PostgreSQL引擎支持按时间点恢复数据和按备份集（快照）恢复数据两种恢复方式，将历史数据恢复到新集群中。本文介绍如何按时间点恢复数据。

注意事项

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。

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. 在左侧导航栏中，选择**配置与管理 > 备份恢复**。
5. 在**备份恢复**页面，单击**按时间点恢复**。
6. 在**克隆实例**页面，选择新集群的**付费模式**：
   - **包年包月**：在创建集群时支付计算节点的费用，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。
   - **按量付费**：无需预先支付费用，计算集群和存储空间（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。

参数名称	参数说明
克隆类型	选择**恢复到过去时间点**。

参数名称	参数说明
时间点	选择需要恢复的时间点。  克隆实例 当前配置 付费模式 包年包月 按量付费 克隆类型 从备份集恢复数据 恢复到过去的时间点 克隆一个独立集群 基于备份集和Redo日志，恢复到过去的某个时间点 时间点 2022-04-12 17:00:00 **说明** 仅支持恢复到7天内的任意时间点。
地域	集群所在**地域**。 **说明** 无需修改，默认与原集群相同。
主可用区	选择集群所在的**主可用区**。 **说明** 在有两个及以上可用区的地域，PolarDB会自动复制数据到备可用区，用于灾难恢复。
网络类型	固定为**专有网络**，无需选择。
VPC网络	选择集群所在的**VPC网络**和**VPC交换机**，建议与原集群一致。 **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。
VPC交换机	
兼容性	默认与原集群**兼容性**一致，无需选择。 例如原集群**兼容性**为**PostgreSQL 11**（与PostgreSQL 11完全兼容），此处**兼容性**也固定为**PostgreSQL 11**。
系列	默认与原集群**系列**一致，无需选择。
子系列	默认与原集群**子系列**一致，无需选择。
节点规格	选择**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。 **说明** 为了保障恢复后的集群运行正常，建议选择高于原集群的**节点规格**。
节点个数	默认为2。 **说明** 新集群默认为1主1备节点，创建后最多可添加至1主15备节点，添加节点请参见增加或删除节点。

参数名称	参数说明
存储费用	您购买时无需选择容量，PolarDB会根据实际使用量按小时计费，您也可以预购存储包，如何购买存储包请参见购买存储包。
集群名称	新PolarDB的**集群名称**，名称要求如下： ○ 长度为2~128个字符。 ○ 以大小写字母或中文开头。 ○ 可包含数字，'.'，'_'或'-'。 如果留空，系统将为您自动生成一个集群名称，创建集群后依旧可以修改。
购买时长	选择PolarDB集群的**购买时长**。 **说明** 该参数只会在**付费模式**为**包年包月**时设置。
购买数量	选择PolarDB集群的**购买数量**。

8. 阅读并选中服务协议，根据选择的**付费模式**完成支付即可。
   - **按量付费**
     单击**立即购买**即可。
   - **包年包月**
     a. 单击**立即购买**。
     b. 在**支付**页面，确认未支付订单信息和支付方式，单击**支付**。

说明 购买成功后，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。

相关API

API	描述
CreateDBCluster	PolarDB的数据恢复需要通过CreateDBCluster来实现。 **说明** 参数CreationOption取值需要为CloneFromPolarDB。

10.4.2. 恢复方式二：按备份集（快照）恢复

PolarDB PostgreSQL引擎支持按时间点恢复数据和按备份集（快照）恢复数据两种恢复方式，将历史数据恢复到新集群中。本文介绍如何按备份集（快照）恢复数据。

注意事项

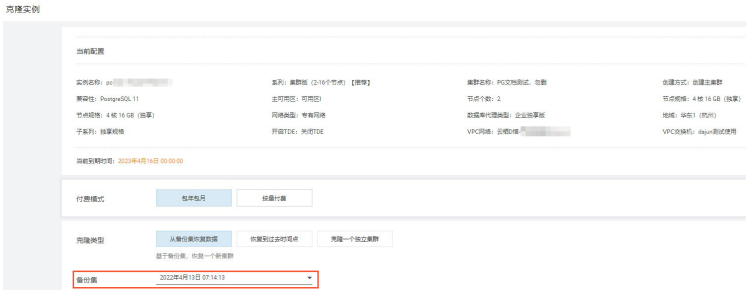
恢复后的集群包含原集群的数据和账号信息，不包含原集群的参数设置。

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**恢复数据到新集群**。
6. 在**克隆实例**页面，选择新集群的**付费模式**：
   - **包年包月**：在创建集群时支付计算节点的费用，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。

- **按量付费**：无需预先支付费用，计算集群和存储空间（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。

参数名称	参数说明
操作类型	选择**从备份集恢复数据**。
备份集	选择需要恢复的**备份集**。  **说明** 此处展示的为各备份集的**备份开始时间**，您可以根据该时间确定是否为需要恢复的备份集。
地域	集群所在**地域**。 **说明** 无需修改，默认与原集群相同。
主可用区	选择集群所在的**主可用区**。 **说明** 在有两个及以上可用区的地域，PolarDB会自动复制数据到备可用区，用于灾难恢复。
网络类型	默认为**专有网络**。
VPC网络	选择集群所在的**VPC网络**和**VPC交换机**，建议与原集群一致。
VPC交换机	
兼容性	默认与原集群**兼容性**一致，无需选择。 例如原集群**兼容性**为**PostgreSQL 11**（与PostgreSQL 11完全兼容），此处**兼容性**也固定为**PostgreSQL 11**。
系列	默认与原集群**系列**一致，无需选择。
子系列	默认与原集群**子系列**一致，无需选择。
节点规格	选择**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。 **说明** 为了保障恢复后的集群运行正常，建议选择高于原集群的**节点规格**。
节点个数	默认为2。 **说明** 新集群默认为1主1备节点，创建后最多可添加至1主15备节点，添加节点请参见增加或删除节点。

参数名称	参数说明
存储费用	您购买时无需选择容量，PolarDB会根据实际使用量按小时计费，您也可以预购存储包，如何购买存储包请参见购买存储包。
集群名称	新建PolarDB的**集群名称**，名称要求如下： ○ 长度为2~128个字符。 ○ 以大小写字母或中文开头。 ○ 可包含数字，'.'，'_'或'-'。 如果留空，系统将为您自动生成一个集群名称，创建集群后依旧可以修改。
购买时长	选择PolarDB集群的**购买时长**。 说明 该参数只会在**付费模式**为**包年包月**时设置。
购买数量	选择PolarDB集群的**购买数量**。

8. 阅读并选中服务协议，根据选择的**付费模式**完成支付即可。
   - **按量付费**
     单击**立即购买**即可。
   - **包年包月**
     a. 单击**立即购买**。
     b. 在**支付**页面，确认未支付订单信息和支付方式，单击**支付**。

说明 购买成功后，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。

相关API

API	描述
CreateDBCluster	PolarDB的数据恢复需要通过CreateDBCluster来实现。 说明 参数CreationOption取值需要为CloneFromPolarDB。

10.5. 常见问题

本文介绍PolarDB PostgreSQL引擎备份与恢复功能的常见问题。

备份FAQ

- Q：一级备份（快照）总大小是否为所有一级备份（快照）之和？
  A：一级备份（快照）总大小不是所有备份（快照）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x50%）]x每小时价格。以中国内地价格为例，PolarDB数据库的一级备份总大小为700 GB，数据库存储用量为1000 GB，那么每小时费用为(700 GB-500 GB)x0.003元/GB=0.6元。
- Q：存储包是否支持抵扣备份空间的费用？
  A：支持。购买存储包后，若抵扣完账号下所有PolarDB集群的存储空间用量后存储包容量仍有剩余，将自动按照1:1.6的比例抵扣一级备份中超过免费额度的空间用量，直至抵扣完存储包中的所有容量。若存储包中剩余容量不足以抵扣一级备份所用量，超出部分将正常按量付费。更多关于存储包抵扣的问题，请参见存储包。
- Q：手动备份仅支持一级备份吗？
  A：是的。
- Q：手动备份保留时长是多久？
  A：手动备份的备份文件的保留时长，由备份策略设置中的数据备份保留中一级备份设置的具体时长决定。

- Q：如何查看二级备份的大小？
  A：您可以在控制台上的**备份列表**页签下查看二级备份的大小。

恢复FAQ

- Q：数据误操作时，例如误删除（表、数据库、集群等）、误修改（整体覆盖或误修改表中的列、行、数据等），如何进行恢复？
  A：您可以根据业务场景，选择不同的方式恢复数据。恢复数据详情请参见恢复方式一：按时间点恢复和恢复方式二：按备份集（快照）恢复。
- Q：支持自定义恢复后的新表的名称吗？
  A：支持。
- Q：没有数据备份可以按时间点恢复吗？
  A：不可以。因为按时间点恢复是先将所选时间点前的一个全量数据备份恢复到集群，然后根据物理日志增量恢复数据到所选时间点。

11.集群回收站

11.1. 费用说明

集群回收站用于保存已释放的PolarDB集群，您可以将回收站中的集群恢复到新集群或者删除某个备份集。本文以PolarDB PostgreSQL引擎集群为例为您介绍集群回收站的费用说明。
集群回收站中一级备份不收取任何费用，转存为二级备份后才会收费。

地域	费用（元/GB/小时）
中国内地	0.00021元
中国香港及海外	0.000294元

11.2. 恢复已释放的集群

本文以PolarDB PostgreSQL引擎集群为例，为您介绍如何使用集群回收站恢复已释放的集群。

注意事项

- 集群回收站中每个集群至少需要存在一个备份集，如果删除了某个集群下所有的备份集，该集群将无法恢复。
- 释放集群后，被释放的集群在集群回收站中都会依次异步转存为二级备份，速度为150 MB/s左右。更多关于备份的介绍请参见概述。

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. 在左侧导航栏中单击**集群回收站**。
4. 找到目标集群，单击右侧**操作**栏中的**恢复数据到新集群**。

5. 选择**商品类型**为**包年包月**或**按量付费**。
   - **包年包月**：在创建集群时支付计算节点的费用，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。
   - **按量付费**：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。
6. 设置以下参数。

参数	说明
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。
创建方式	选择**从回收站恢复**。 即从回收站恢复已经删除的数据库
原版本	选择已删除集群的版本。

参数	说明
已删除集群	选择已删除的集群名称。
历史备份	选择需要恢复的备份。 **说明** **历史备份**显示的备份时间为UTC时间，备份列表中显示的备份时间为当前系统时间，请确保所选择的历史备份无误。例如，当前备份列表显示的备份时间为 `2020年05月08日 10:00:00（北京时间）`，对应的**历史备份**为 `2020-05-08T02:00:00Z`。
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。
网络类型	固定为**VPC**专有网络，无需选择。
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，详情请参见创建和管理专有网络。
兼容性	PolarDB集群的数据库引擎版本，默认与删除集群的版本保持一致，不可变更。
系列	固定为**集群版（2-16个节点）【推荐】**，无需选择。
子系列	PolarDB集群版支持**通用规格**和**独享规格**两种子系列，其中： ○ **独享规格**：每个集群会独占所分配到的计算资源（如CPU），而不会与同一服务器上的其他集群共享资源，性能更加稳定可靠。 ○ **通用规格**：同一服务器上的不同集群，会互相充分利用彼此空闲的计算资源（如CPU），通过复用计算资源享受规模红利，性价比更高。 关于两种类型的详细对比，请参见如何选择通用规格和独享规格。
节点规格	按需选择，建议不低于已删除集群的规格。关于PolarDB节点规格，详情请参见规格与定价。
节点个数	无需选择。系统将默认创建规格相同的两个节点（一主一只读）。 **说明** 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，请参见产品架构。

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，详情请参见规格与定价。 **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。开启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** TDE功能开启后不可关闭。
集群名称	输入集群名称，集群名称需满足如下要求： ○ 不能以 `http://` 或 `https://` 开头。 ○ 长度为2~256个字符。 如果留空，系统将为自动生成一个集群名称，创建集群后还可以修改。
资源组	从已创建资源组中选择一个目标资源组。 **说明** 资源组是在单个云账号下将一组相关资源进行统一管理的容器，一个资源只能归属于一个资源组，详情请参见RAM资源分组与授权。
购买时长	选择集群的**购买时长**。 **说明** 仅当**商品类型**为**包年包月**时支持设置该参数。
购买数量	选择集群的**购买数量**。

7. 根据集群的**付费模式**完成后续购买操作。
   - **按量付费**
     a. 单击**立即购买**。
     b. 在**确认订单**页面确认订单信息，阅读并选中服务协议，单击**立即开通**即可。
   - **包年包月**
     a. 单击**立即购买**。
     b. 在**确认订单**页面确认订单信息，阅读并选中服务协议，单击**去支付**。
     c. 在**支付**页面，确认未支付订单信息和支付方式，单击**支付**。

开通成功后，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 恢复到新集群所需的时间和您的备份大小有关，备份越大恢复到新集群所需的时间越久。集群创建成功后，您可以返回PolarDB控制台并在**集群列表**查看新创建的集群。

相关API

API	描述
CreateDBCluster	创建PolarDB集群。

11.3. 彻底删除已释放的集群

本文以PolarDB PostgreSQL引擎集群为例，介绍如何删除已释放集群的备份集。

注意事项

- 集群回收站中每个集群至少需要存在一个备份集，如果删除了某个集群下所有的备份集，该集群将无法恢复。
- 释放集群后，被释放的集群在集群回收站中都会依次异步转存为二级备份，速度为150 MB/s左右。更多关于备份的介绍请参见概述。

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. 在左侧导航栏中单击**集群回收站**。
4. 找到目标集群，单击目标集群名称左侧的 + 图标，展开备份集列表。
5. 找到需要删除的备份集，单击**操作**栏的**删除备份**。

6. 在弹出的对话框中，单击**确定**。

> **警告** 如果删除了**集群回收站**中某个集群下所有的备份集，该集群将无法恢复，请谨慎处理。

12.数据安全/加密

12.1. 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。

> **说明** PolarDB PostgreSQL引擎的集群仅支持为主地址配置SSL。

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。

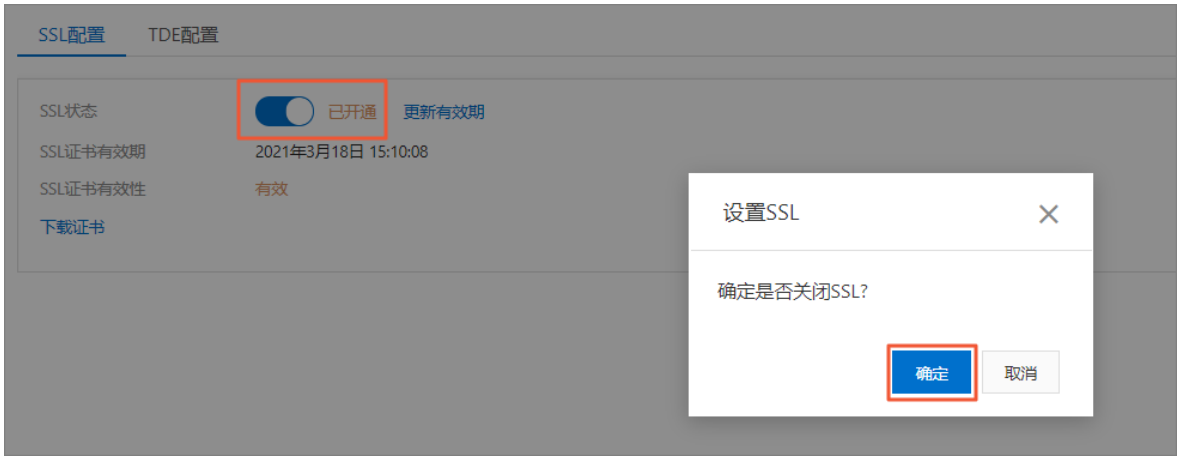
> **说明** 下载证书请参见开启SSL加密和下载证书步骤七。

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

12.2. 设置透明数据加密TDE

PolarDB PostgreSQL引擎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 PostgreSQL引擎。
- 已开通KMS。具体操作请参见开通密钥管理服务。

- 已授权RDS访问KMS。具体操作请参见授权RDS访问KMS。

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 PostgreSQL引擎 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 PostgreSQL引擎数据库集群。

5. 单击页面右下方的**立即购买**。
6. 在**确认订单**页面确认订单信息，阅读并选中服务协议，单击**去支付**。
7. 在**支付**页面，确认未支付订单信息和支付方式，单击**订购**。

   > **说明** 完成支付后，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。

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**。

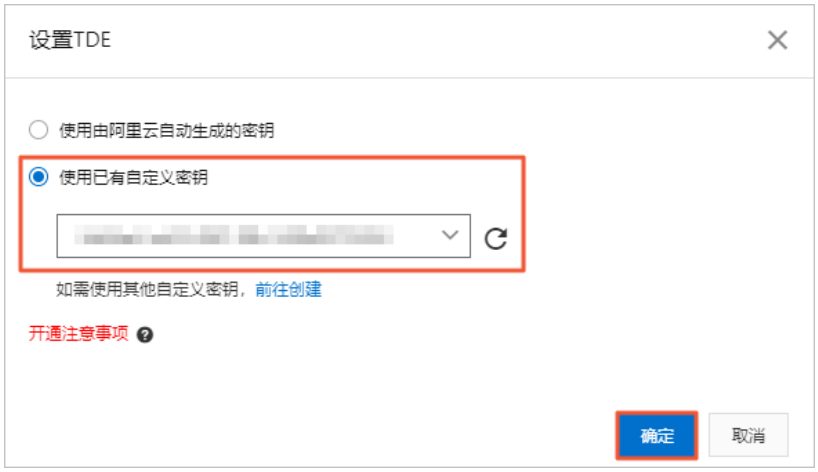
切换为自定义密钥

1. 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，单击**TDE状态**右侧的**切换为自定义密钥**。

6. 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。详情请参见创建密钥。



7. 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
CreateDBCluster	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。

13.诊断与优化

13.1. SQL洞察

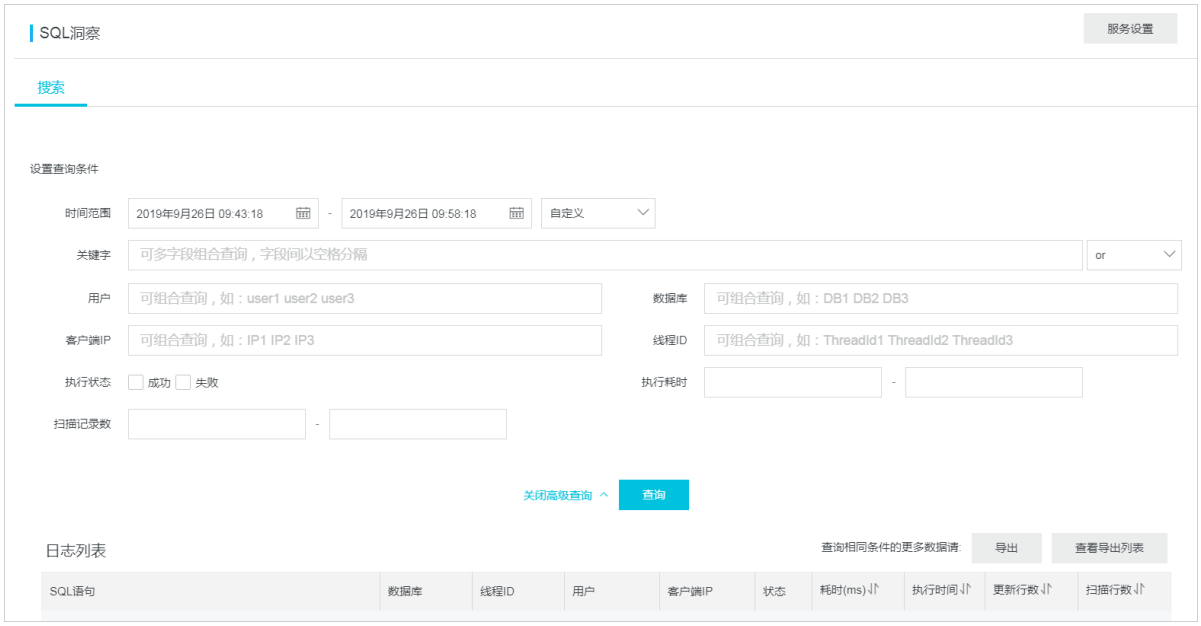
SQL洞察功能为您的数据库提供安全审计、性能诊断等增值服务。

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SQL洞察计费规则（可选）。

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**。

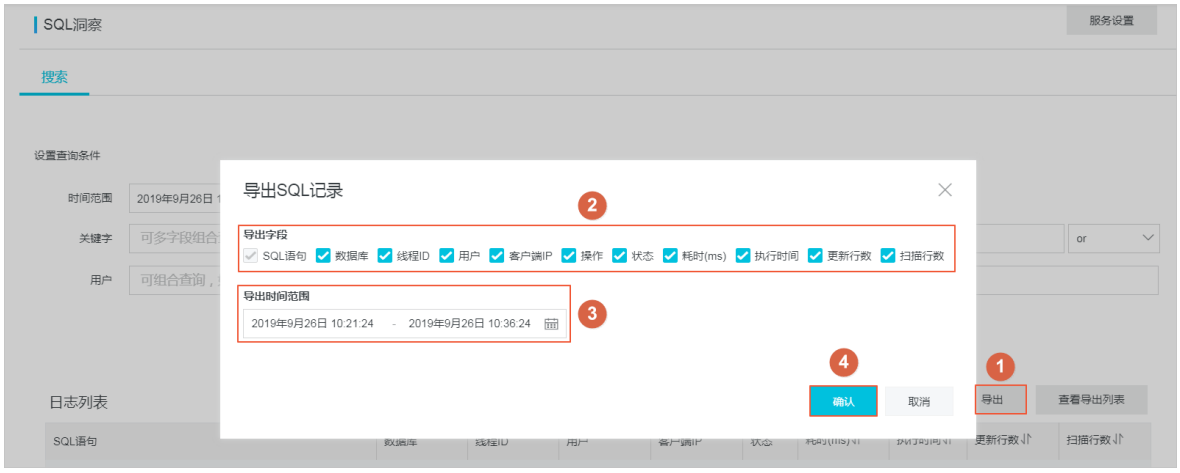
6. 选择SQL审计日志的保存时长，单击**开通服务**。

修改SQL日志的存储时长

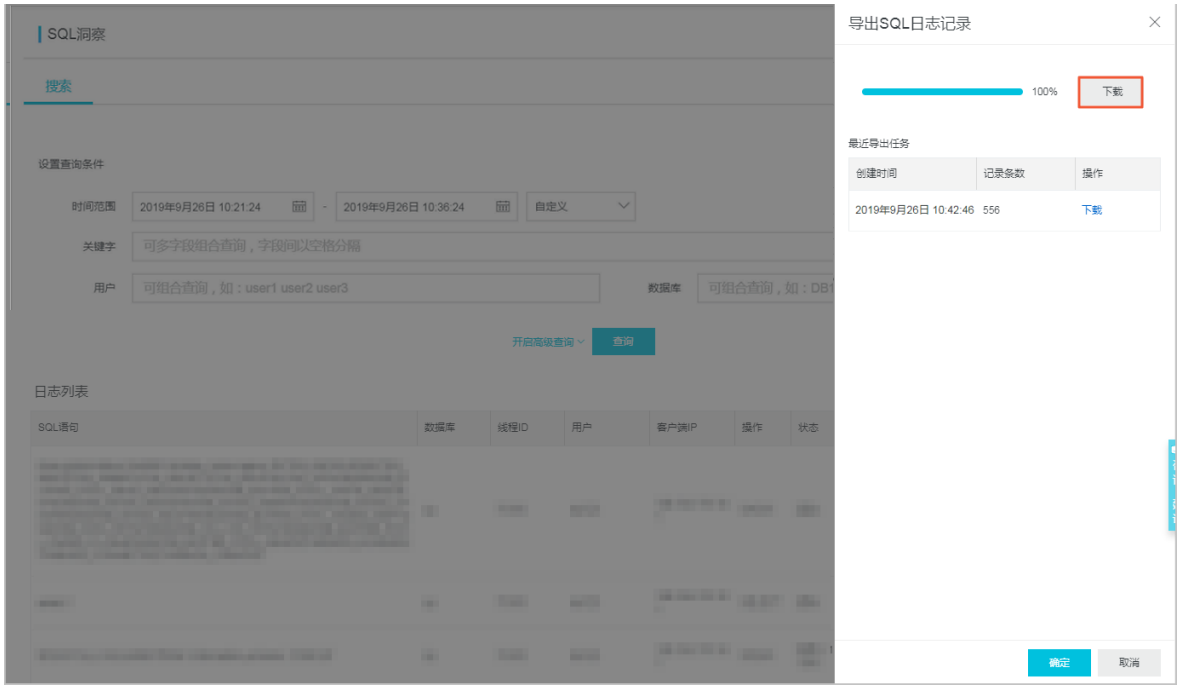
1. 登录PolarDB控制台。
2. 在控制台左上角，选择地域。
3. 单击目标集群ID。
4. 在左侧导航栏中，选择**日志与审计 > SQL洞察**。
5. 单击右上角**服务设置**。
6. 修改存储时长，单击**确认**。

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**账单管理 > 账单详情** 。
4. 在**账单详情**页签，单击**明细账单**页签，设置搜索**实例ID**并输入目标集群ID进行搜索。

> **说明** 若要查询超过12个月前的记录，请提交工单。

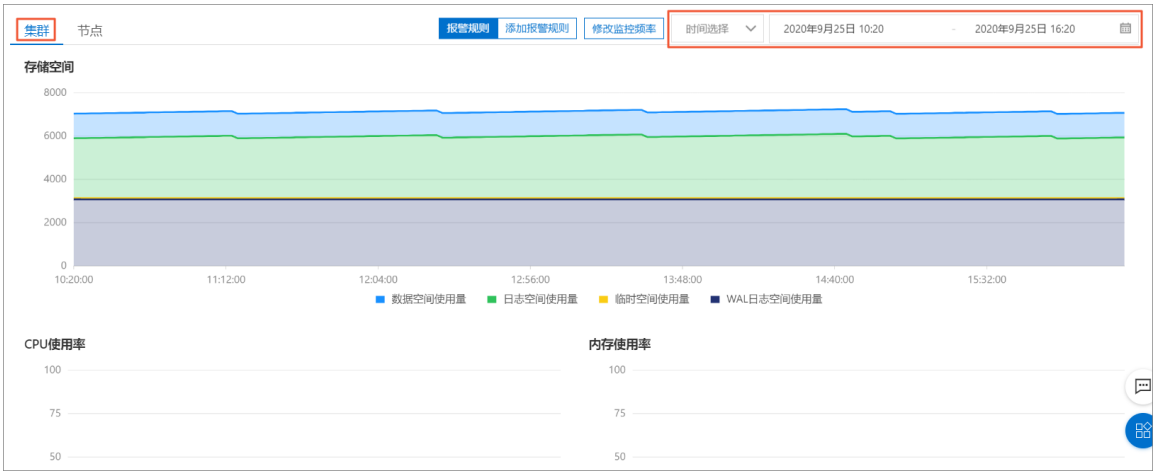
5. 查看**计费项**列为**sql_explorer**的费用明细。

13.2. 性能监控

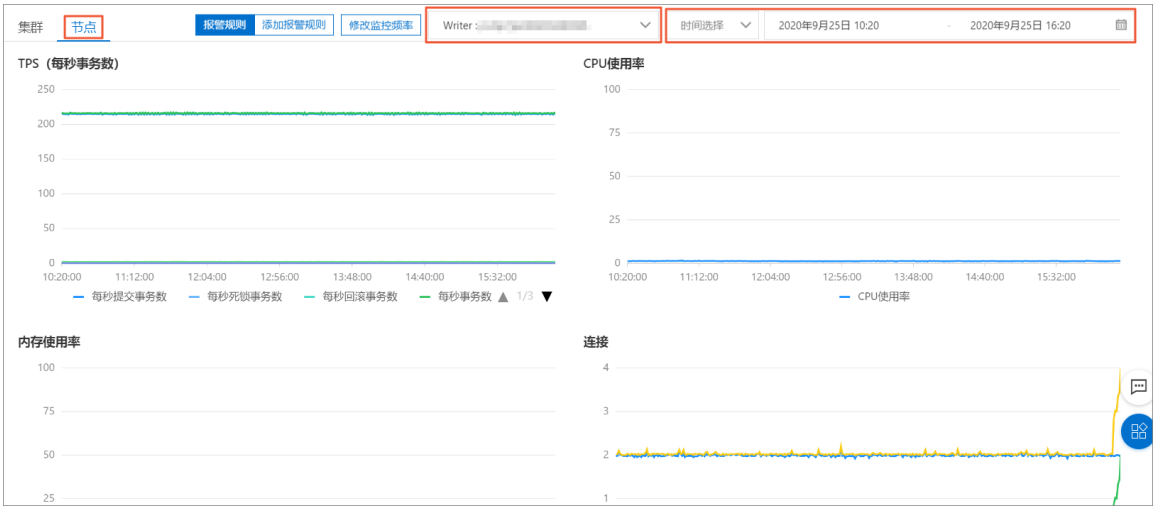
PolarDB控制台提供了丰富的性能监控项和秒级监控频率，方便您掌握集群的运行状态并通过细粒度的监控数据快速定位运维问题。

性能监控

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**诊断与优化 > 性能监控**。
5. 您可以根据业务需求选择查看**集群**或**计算节点**的监控信息。详细信息，请参见监控项说明。
   - 集群性能监控：单击**集群**页签，在右侧设置时间段后单击**确定**。



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



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。

类别	监控项	说明
	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。
	临时文件	展示所选择节点的临时文件数量和总大小。

相关API

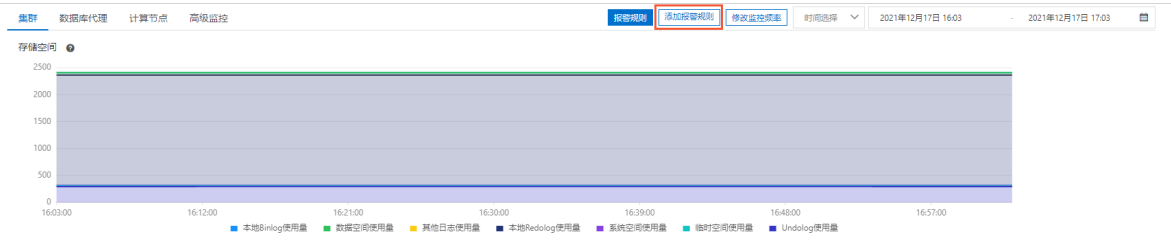
API	描述
DescribeDBClusterPerformance	查询PolarDB集群的性能数据。
DescribeDBNodePerformance	查询PolarDB集群节点的性能数据。
DescribeDBClusterMonitor	查询PolarDB集群监控数据的采集频率。
ModifyDBClusterMonitor	修改PolarDB集群监控数据的采集频率。

13.3. 创建报警规则

您可以通过PolarDB控制台创建阈值报警规则，帮助您及时了解PolarDB集群或节点的监控数据异常并快速进行处理。

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. 在左侧导航栏中，选择**诊断与优化 > 性能监控**。
5. 单击**添加报警规则**。



6. 在**创建报警规则**页面， 设置以下参数。

步骤	参数	说明
关联资源	**产品**	保持默认值**云数据库POLARDB-PostgreSQL**即可。
	资源范围	报警规则的作用范围，取值范围为**全部资源**或**集群** **说明** ○ 若选择**资源范围**为**全部资源**，则当产品下任何集群满足报警规则描述时，都会发送报警通知。 ○ 若选择**资源范围**为**集群**，则仅当目标集群满足报警规则描述时，才会发送报警通知。
	规则名称	输入报警规则的名称。

步骤	参数	说明
设置报警规则	规则描述	报警规则的主体，定义在监控数据满足指定条件时，触发报警规则。 说明 更多关于报警规则的举例说明，请参见创建阈值报警规则。
	通道沉默期	报警发生后如果关联资源未恢复正常，间隔多久重复发送一次报警通知，最短为5分钟，最长为24小时。
	生效时间	报警规则的生效时间。 说明 报警规则只在生效时间内发送报警通知，非生效时间内产生的报警只记录报警历史。
通知方式	关于通知方式步骤的参数详情，请参见创建阈值报警规则。	

7. 单击**确认**。

13.4. 管理报警规则

您可以通过PolarDB控制台管理阈值报警规则，帮助您及时了解PolarDB集群或节点的监控数据异常并快速进行处理。

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. 在左侧导航栏中，选择**诊断与优化 > 性能监控**。
5. 单击**报警规则**，进入**报警规则列表**页面。

6. 在**阈值报警**页签下，您可以对已有的报警规则执行以下操作。
   - 单击**操作**栏中的**查看**，可查看报警规则的基本信息。
   - 单击**操作**栏中的**报警历史**，可查看报警历史。
   - 单击**操作**栏中的**修改**，可修改报警规则。
   - 单击**操作**栏中的**禁用**，可禁用报警规则。
   - 单击**操作**栏中的**删除**，可删除报警规则。
   - 单击**通知对象**栏中的**查看**，可查看报警联系组、报警联系人及报警通知方式。

13.5. 性能洞察

PolarDB PostgreSQL引擎的一键诊断融合了DAS的部分功能，您可以通过其中的性能洞察，快速评估数据库负载，找到性能问题的源头，提升数据库的稳定性。

背景信息

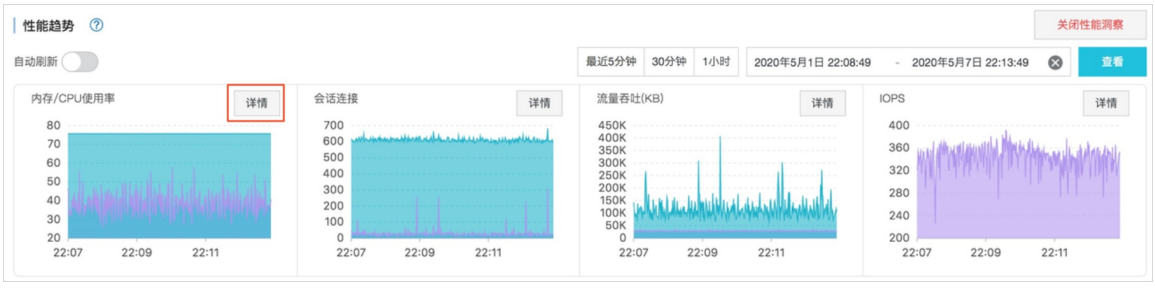
用于性能洞察的数据主要来源如下：

- 如果目标实例已经开启了performance_schema，直接采集和分析performance_schema中的数据。
- 如果目标实例未开启performance_schema，则采集和分析活跃会话数据。

操作步骤

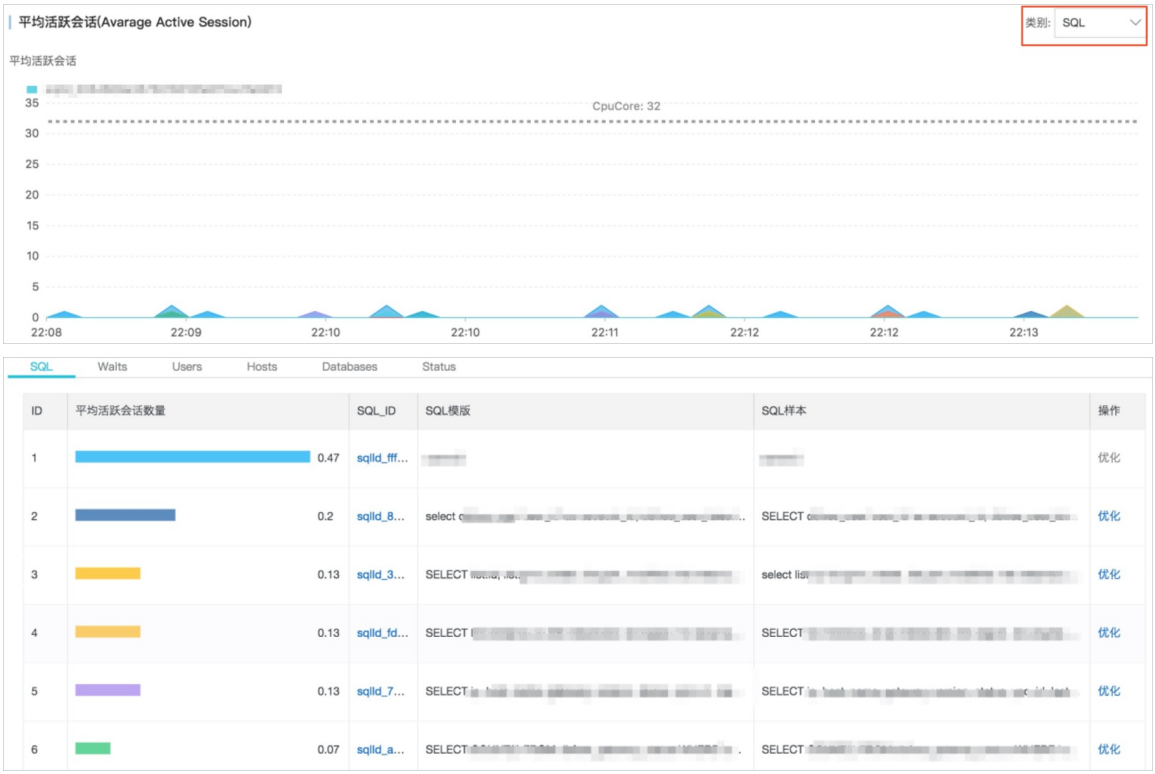
1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 在**集群列表**页，单击目标集群ID。
4. 在左侧导航栏中，选择**诊断与优化 > 一键诊断**。
5. 单击**性能洞察**页签。
6. 单击**开启性能洞察**。

7. 在弹出的对话框内，单击**确定**。
8. 您可以在性能洞察页面查看和管理如下信息：
   - 在**性能趋势**区域，您可以查看特定时段的数据库性能情况。若您需要查看某个具体性能（如CPU使用率），可以单击该性能名称右侧的**详情**按钮进行查看。



> **说明** 可供查看的时间范围不能超过7天。

   - 在**平均活跃会话**区域，您可以查看不同类别（如SQL）会话的变化趋势图和相关多维负载信息列表，确定性能问题源头。


平均活跃会话(Avarage Active Session)
类别: SQL
平均活跃会话
CpuCore: 32
35
30
25
20
15
10
5
0
22:08
22:09
22:10
22:10
22:11
22:12
22:12
22:13
SQL
Waits
Users
Hosts
Databases
Status
ID
平均活跃会话数量
SQL_ID
SQL模版
SQL样本
操作
1
0.47
sqlId_fff...
优化
2
0.2
sqlId_8...
优化
3
0.13
sqlId_3...
优化
4
0.13
sqlId_fd...
优化
5
0.13
sqlId_7...
优化
6
0.07
sqlId_a...
优化

14.配置参数

14.1. 设置集群参数

本文将介绍如何通过PolarDB PostgreSQL引擎控制台修改集群参数。

> **说明** 支持修改的参数以控制台为准。

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. 在左侧导航栏中，选择**配置与管理 > 参数配置**。
5. 找到目标参数，单击**当前值**栏的图标，在弹出的对话框中，输入新的参数值，单击**确定**。

> **说明**
> - 请输入目标参数右侧**修改范围**栏内规定的参数值，否则当您**提交修改**时会出现错误提示。
> - 您可以单击目标参数名称后的图标查看参数的详细说明。

6. 单击左上角**提交修改**，在弹出的**保存改动**对话框中，单击**确定**。

> **警告** 对于**重启**栏显示为**是**的参数，单击**确定**后集群将会重启，请在修改参数前做好业务安排，谨慎操作。

相关API

API	描述
DescribeDBClusterParameters	查看集群的参数。

API	描述
ModifyDBClusterParameters	修改集群的参数。

14.2. polar_create_table_with_full_replica_identity

当您通过逻辑复制的方式将PolarDB PostgreSQL引擎中的无主键表同步到其它数据库时，可能导致该表上的操作报错，您可以通过设置 `polar_create_table_with_full_replica_identity` 参数解决该问题。

PolarDB PostgreSQL引擎的逻辑复制采用“发布-订阅”模式，可以将发布端的操作以类似SQL的形式在订阅端执行，达到数据同步的目的。为了在订阅端标识待更新或删除的数据，发布端需要配置表的复制标识。

复制标识支持以下几种类型：

- 主键
- 唯一索引
- FULL（整行数据）

复制标识默认为主键，如果一个没有主键的表进行逻辑复制，会出现变更操作报错的情况，从而影响业务的正常运行。报错信息如下所示：

```
ERROR:  cannot delete from table "polardb_test" because it does not have a replica identity and publishes deletes
HINT:  To enable deleting from the table, set REPLICA IDENTITY using ALTER TABLE.
```

> **注意** 当您使用逻辑复制（例如使用DTS进行数据同步）时，需要确保所有待同步的无主键表的复制标识均已设置为 `FULL` 。

PolarDB PostgreSQL引擎提供了如下两种方式将表的复制标识修改为 `FULL` ：

- 将现有表的复制标识修改为 `FULL` ，修改命令如下：

  ```
  ALTER TABLE <table_name> REPLICA IDENTITY FULL;
  ```

- 将 `polar_create_table_with_full_replica_identity` 参数设置为*on*即可将新建表的默认复制标识设置为 `FULL` 。

  > **说明** `polar_create_table_with_full_replica_identity` 参数默认为*off*，该参数目前无法在控制台进行修改，如有需要您可以提交工单联系技术支持进行修改。

15.标签

15.1. 绑定标签

在集群数量较多的情况下，您可以创建多个标签，为集群绑定不同的标签对其进行分类，之后通过标签进行集群筛选。

注意事项

- 标签由一对键值组成，键在同账号同地域下唯一，值无此限制。
- 每个集群最多可绑定 20个标签，相同的标签键会被覆盖。
- 不同地域的标签信息是独立的。

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，将鼠标移动到目标集群**标签**栏的图标上。
4. 单击**编辑标签**。

5. 在**编辑标签**对话框，单击**新建标签**或**已有标签**。
   - **新建标签**：
     设置标签的**键**和**值**后，单击**确定**。

> **说明** 完成标签的新建后，您可以在其它实例中绑定已创建的标签。

   - **已有标签**：

单击需要绑定的**标签键**即可。

6. 重复以上步骤，完成所有标签的绑定，之后单击对话框右下角的**确定**。

相关API

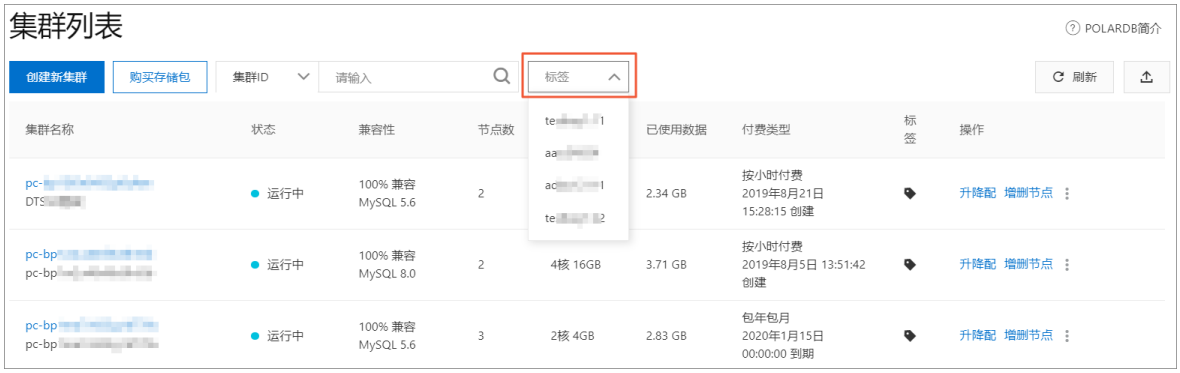
API	说明
TagResources	该接口用于为PolarDB集群绑定标签。

15.2. 根据标签筛选集群

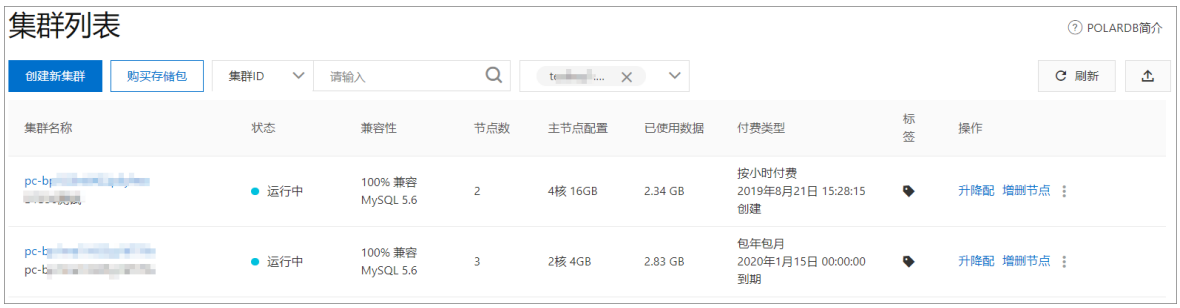
为PolarDB集群绑定标签后，您可以在集群列表中通过标签进行筛选，找出绑定指定标签的集群。

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**标签**，选择目标**标签**。



4. 选择指定标签后，**集群列表**将显示所有绑定该标签的集群。



相关API

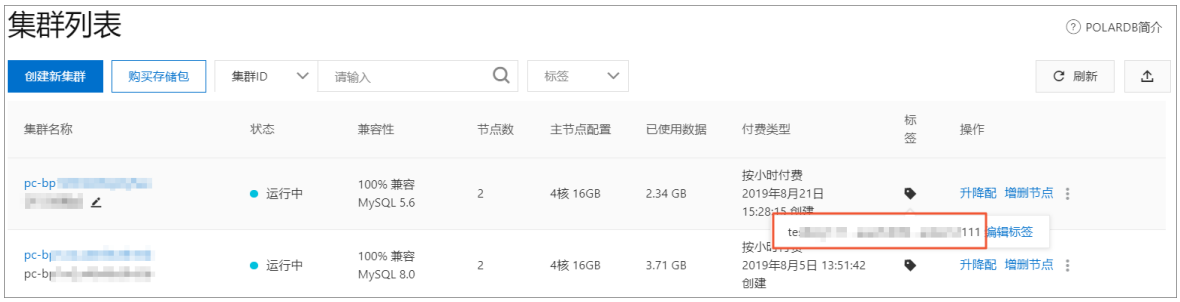
API	说明
ListTagResources	该接口用于查询一个或多个PolarDB集群已绑定的标签列表，或者查询一个或多个标签绑定的PolarDB集群列表。

15.3. 查看集群绑定的标签

您可以在集群列表中查看PolarDB集群绑定了哪些标签。

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，将鼠标移动到目标集群**标签**栏的图标上。
4. 查看目标集群绑定的标签，如下图所示。



相关API

API	说明
ListTagResources	该接口用于查询一个或多个PolarDB集群已绑定的标签列表，或者查询一个或多个标签绑定的PolarDB集群列表。

15.4. 解绑标签

当集群不再需要某个标签时，您可以将该标签从PolarDB集群上解绑。

注意事项

标签从一个集群解绑后，如果没有绑定到其它集群，则该标签自动被删除。

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，将鼠标移动到目标集群**标签**栏的图标上，单击**编辑标签**。

4. 在**编辑标签**对话框，单击目标标签右侧的图标。

5. 单击**确定**。

> **说明** 在一个集群中解绑标签不影响其它绑定了该标签的集群。

相关API

API	说明
UntagResources	该接口用于将标签从PolarDB集群上解绑。

16.插件

16.1. 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 PostgreSQL引擎数据库中，也支持将PolarDB PostgreSQL引擎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 PostgreSQL引擎集群的资源（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 PostgreSQL引擎与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。

16.2. 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引擎引入了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引擎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GitHub。

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 n
ull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')::inter
val 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
```

```
 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')::inter
val 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)
```

```
(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 zo
ne 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 zon
e 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 zon
e 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 zon
e 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 zon
e 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 ins
ert 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 zon
e 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 par
ent 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 zon
e 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 zon
e 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)
```

16.3. 使用中文分词

本文为您介绍PolarDB PostgreSQL引擎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。

16.4. 使用plprofiler插件

您可以使用plprofiler插件对PolarDB PostgreSQL引擎集群进行性能分析。

背景信息

您在使用Postgres服务端进行编程的时候，您会发现PostgreSQL的PL/PGSQL是黑盒环境，内部的任何问题都有可能造成性能瓶颈。常见异常情况如下：

- 问题语句一开始执行速度很快，调用多次后速度变慢。
- 随机出现的性能瓶颈问题。
- 生产系统上出现了性能问题。

通常情况下，上述性能问题只能采取人工分析（分析Schema、统计信息、SQL语句）或断点（pldebugger）的形式进行排查，排查时间长且不直观、性能问题时隐时现（对于上述的第一个问题，甚至无法排查出原因），因此您需要有一个更好的排查方式，来帮助您找到性能的瓶颈。

plprofiler提供了一个简洁的PL/PGSQL的性能采集方式，用于发现PL/PGSQL的性能瓶颈，让您可以针对性地对函数、存储过程和Schema等性能进行优化。更多关于插件的信息，请参见plprofiler。

准备工作

> **说明** 以下操作示例仅适用于Linux系统和Mac OS系统。

1. 从GitHub获取plprofiler源码。
2. 导出变量环境。

```
export PGHOST=<polardb_host>
export PGPORT=<polardb_port>
export PGUSER=<polardb_user>
export PGPASSWORD=<polardb_password>
export PGDATABASE=pgbench_plprofiler
export PLPROFILER_PATH=<path-to-plprofiler>
export USE_PGXS=1
export PATH=<path-to-plprofiler>/bin:$PATH
```

> **说明** 请将示例中的连接串信息替换成真实连接串信息，如何查看连接地址请参见查看或申请连接地址。

3. 进入到源码目录，安装客户端软件，操作示例如下。

```
cd $PLPROFILER_PATH/python-plprofiler
python setup.py install #sudo python setup.py install, or using 'pip install plprofiler'
```

4. 登录PolarDB PostgreSQL引擎数据库，创建数据库和插件，操作示例如下。

```
> CREATE DATABASE pgbench_plprofiler;
> \c pgbench_plprofiler
> CREATE EXTENSION plprofiler;
```

5. 准备用于测试的表、数据和函数。

```
cd $PLPROFILER_PATH/examples
bash prepdb.sh
```

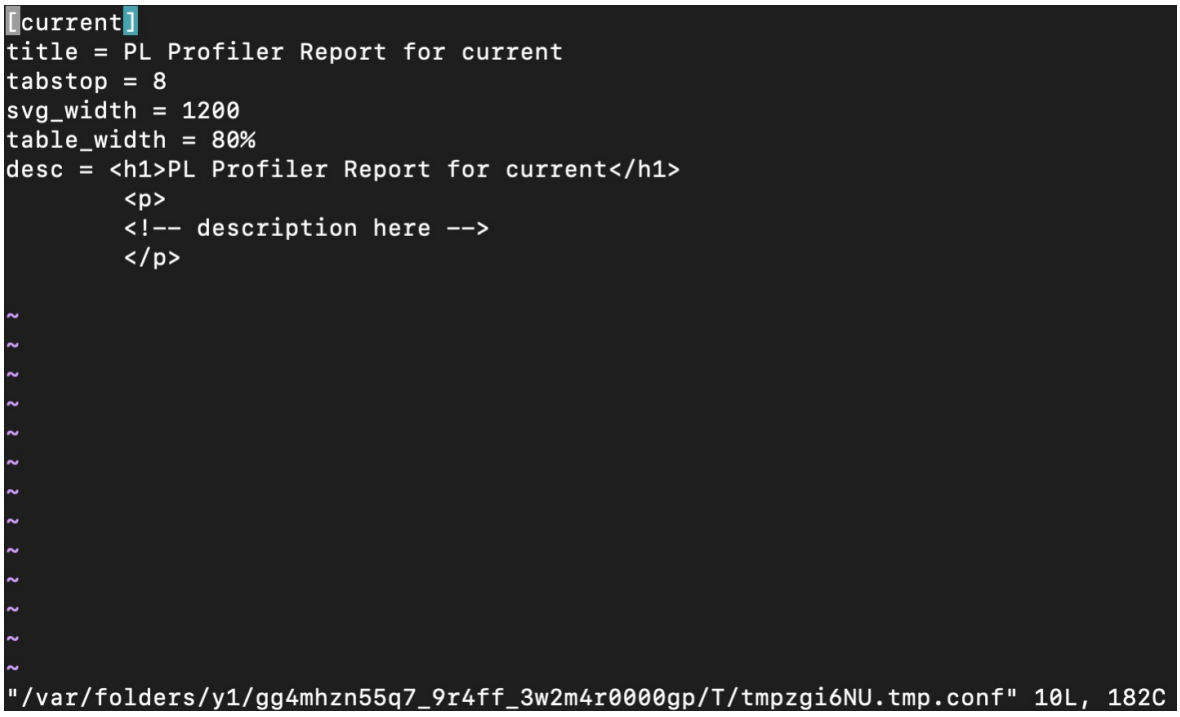
性能分析

1. 运行plprofiler命令进行性能分析，命令如下：

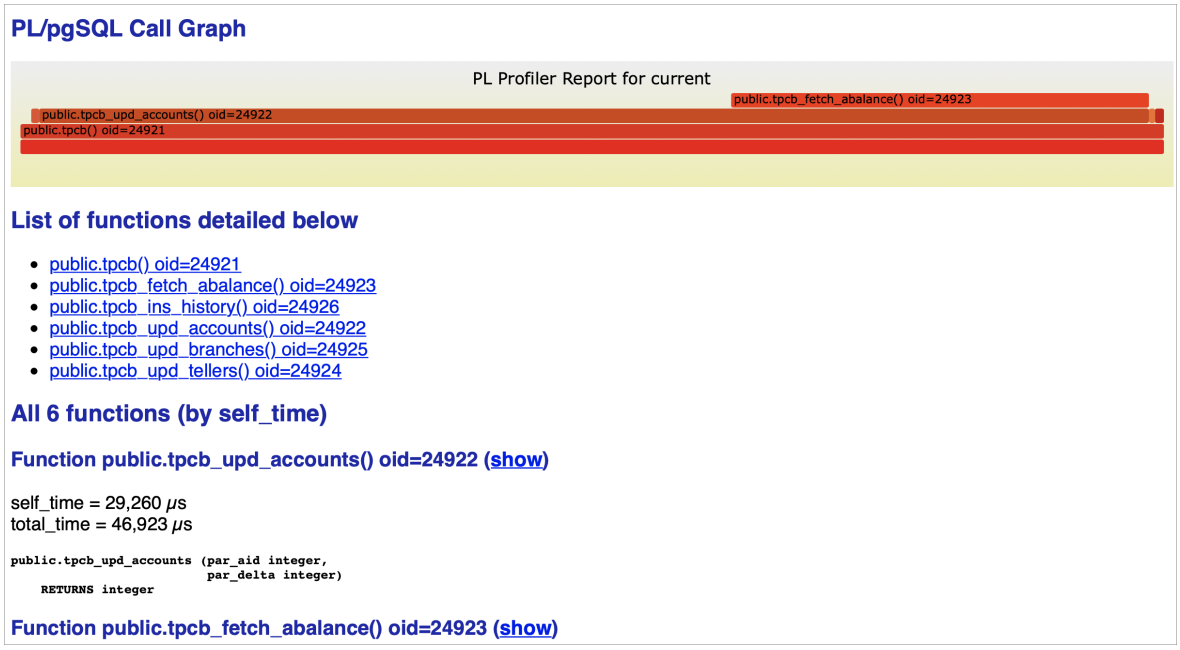
```
plprofiler run --command "SELECT tpcb(1, 2, 3, -42)" --output tpcb-test1.html
```

> **说明** 其中tpcb-test1.html为输出的性能分析页面地址。

2. 命令完成后会进入编辑界面，可以在当前界面编辑输出网页的标题、长宽、描述等信息，您无需修改直接退出即可。



3. 使用浏览器打开tpcb-test1.html，如下所示。

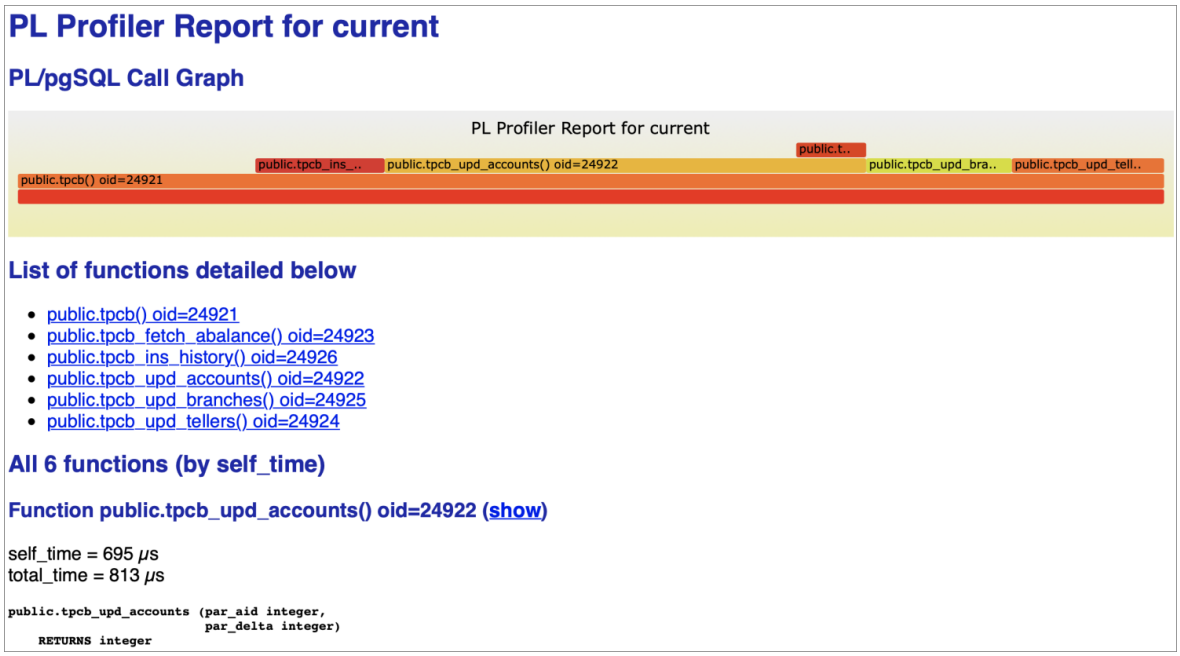


从上方火焰图中可以看出最影响性能的函数是 `tpcb_fetch_abalance` ，可以分析出是因为没有创建索引导致性能差。虽然 `tpcb_upd_accounts` 性能也很差，但是该函数可能是受到子函数的影响导致性能差，需要优化完子函数后再查看是否正常。

4. 创建索引完成本次优化，示例如下。

```
psql
> CREATE INDEX pgbench_accounts_aid_idx ON pgbench_accounts (aid);
```

5. 再次执行plprofiler命令进行性能分析，结果如下。



通过上图中的火焰图可以看出， `tpcb_fetch_abalance` 不再是性能瓶颈。由于优化了 `tpcb_fetch_abalance` ， `tpcb_upd_accounts` 的执行时间也缩短了。

如果符合期望，那么本次优化就到此为止；如果没有达到预期，您可以通过新的火焰图继续分析出性能瓶颈进行优化。

16.5. 使用pldebugger插件

PolarDB PostgreSQL引擎支持多种存储过程语言，例如PLpgSQL、PL/Python、PL/Perl、PL/Tcl、PL/Java等等，您可以使用这些存储过程语言创建对应的函数或存储过程。PolarDB提供了pldebugger插件，可用于调试存储过程。

前提条件

pgAdmin 4客户端版本需要为V4.19及以上版本，如何下载pgAdmin 4请参见pgAdmin下载页。

注意事项

PolarDB对pldebugger插件的连接数进行了限制，每台集群最多可以启动三个调试连接。如果因业务需求需要超过三个调试连接数，请提交工单联系售后服务。

例如，当前已有三个debugger调试连接，此时第四个调试连接无法正常运行，可以关闭一个正常运行的连接

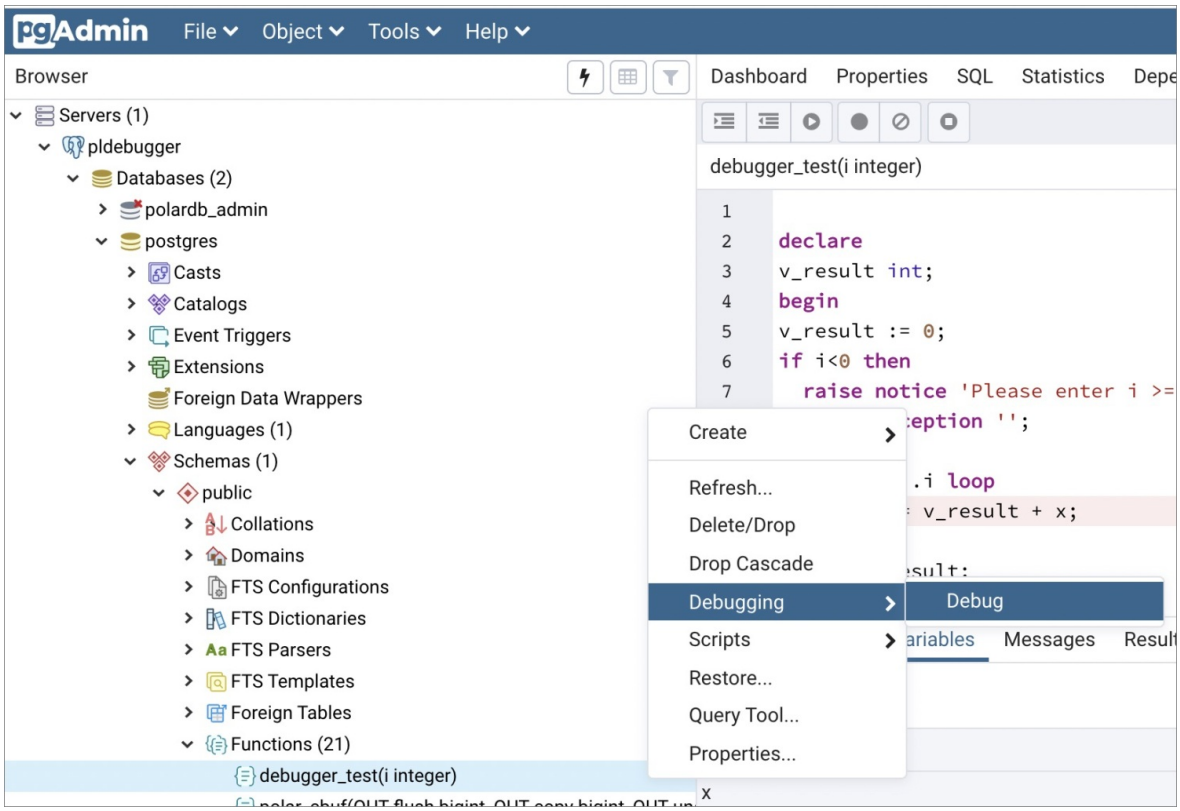
创建插件

请使用polar_superuser用户对插件进行创建与调试。

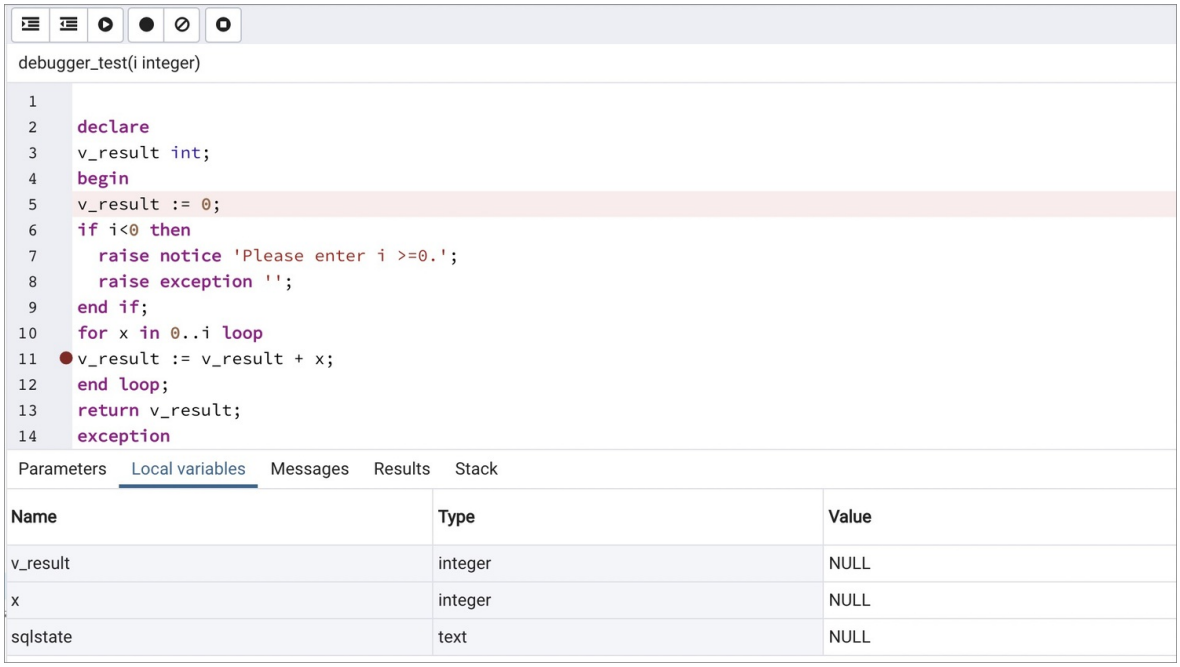
```
postgres=# CREATE EXTENSION if not exists pldbgapi;
CREATE EXTENSION
```

使用插件

1. 使用pgAdmin 4连接PolarDB集群，具体操作请参见连接数据库集群。
2. 找到待调试函数，右键该函数，单击**Debugging > Debug**。



3. 至此，即可在pgAdmin 4中使用pldebugger插件。



- 在pgAdmin 4页面右侧函数调试框中，您可以对目标函数进行单步调试，例如 `step into/over` 、 `continue` 、加设断点以及停止等操作。
- 在pgAdmin 4页面底部，您可以查看调试过程中显示的局部变量信息、调试结果以及函数堆栈。

调试连接数监控

PolarDB提供了pldebugger调试连接数监控函数，您可以使用该函数查看已有的调试连接和默认设置的最大连接。

```
postgres=# select * from polar_monitor_pldebugger_count();
 current_connection | max_connection
--------------------+----------------
                  6 |              6
(1 row)
```

- `current_connection` ：当前已使用的连接数。
- `max_connection` ：默认的最大调试连接数。

通过以上示例可以看出，当前有六个已使用的连接。由于一个pldebugger调试会使用两个连接数，可以看出以上示例中有三个处于调试中pldebugger进程；最多允许有三个pldebugger调试进程。

16.6. 使用pg_roaringbitmap插件

RoaringBitmap是一种压缩位图，其性能优于WAH、EWAH、Concise等传统压缩位图。在特定场景下，RoaringBitmap可以在提供良好压缩性能的条件下，仍然具备快于传统压缩位图近百倍的索引性能，甚至性能可以超过未采用压缩的位图。

创建插件

```
postgres=# CREATE EXTENSION if not exists roaringbitmap;
CREATE EXTENSION
```

查看插件版本

```
postgres=# \dx
                    List of installed extensions
     Name      | Version |   Schema   |         Description
---------------+---------+------------+------------------------------
 plpgsql       | 1.0     | pg_catalog | PL/pgSQL procedural language
 roaringbitmap | 0.5     | public     | support for Roaring Bitmaps
(2 rows)
```

输入和输出格式

PolarDB目前仅支持 `array` 和 `bytea` 两种输入输出格式。

- 输入格式
  - array

```
postgres=# select roaringbitmap('{1,100,10}');
                roaringbitmap
-----------------------------------------------
 \x3a300000010000000000002001000000001000a006400
(1 row)
```

  - bytea

```
postgres=# select '\x3a300000010000000000002001000000001000a006400'::roaringbitmap;
                roaringbitmap
-----------------------------------------------
 \x3a300000010000000000002001000000001000a006400
(1 row)
```

- 输出格式

> **说明** 输出格式默认为 `bytea` ，您可以通过 `roaringbitmap.output_format` 修改输出格式。

```
postgres=# set roaringbitmap.output_format='bytea';
SET
postgres=# select '{1}'::roaringbitmap;
             roaringbitmap
---------------------------------------
 \x3a300000010000000000000100000000100
(1 row)
postgres=# set roaringbitmap.output_format='array';
SET
postgres=# select '{1}'::roaringbitmap;
 roaringbitmap
---------------
 {1}
(1 row)
```

创建表

```
CREATE TABLE t1 (id integer, bitmap roaringbitmap);
```

整数集合创建位图

```
INSERT INTO t1 SELECT 1,rb_build(ARRAY[1,2,3,4,5,6,7,8,9,200]);
INSERT INTO t1 SELECT 2,rb_build_agg(e) FROM generate_series(1,100) e;
```

位图计算函数

位图计算函数包含OR、AND、XOR、ANDNOT。

```
SELECT roaringbitmap('{1,2,3}') | roaringbitmap('{3,4,5}');
SELECT roaringbitmap('{1,2,3}') & roaringbitmap('{3,4,5}');
SELECT roaringbitmap('{1,2,3}') # roaringbitmap('{3,4,5}');
SELECT roaringbitmap('{1,2,3}') - roaringbitmap('{3,4,5}');
```

位图聚合函数

位图聚合函数包含OR、AND、XOR、BUILD。

```
SELECT rb_or_agg(bitmap) FROM t1;
SELECT rb_and_agg(bitmap) FROM t1;
SELECT rb_xor_agg(bitmap) FROM t1;
SELECT rb_build_agg(e) FROM generate_series(1,100) e;
```

位图基数计算函数

```
SELECT rb_cardinality('{1,2,3}');
```

位图转换整数集合

```
SELECT rb_to_array(bitmap) FROM t1 WHERE id = 1;
```

位图转换整数

```
SELECT unnest(rb_to_array('{1,2,3}'::roaringbitmap));
```

或

```
SELECT rb_iterate('{1,2,3}'::roaringbitmap);
```

操作列表

操作符	输入	输出	描述	示例	结果
&	roaringbitmap,roaringbitmap	roaringbitmap	位元且运算。	roaringbitmap('{1,2,3}') & roaringbitmap('{3,4,5}')	{3}
|	roaringbitmap,roaringbitmap	roaringbitmap	位元或运算。	roaringbitmap('{1,2,3}') | roaringbitmap('{3,4,5}')	{1,2,3,4,5}
|	roaringbitmap,integer	roaringbitmap	将元素添加到RoaringBitmap中。	roaringbitmap('{1,2,3}') | 6	{1,2,3,6}
|	integer,roaringbitmap	roaringbitmap	将元素添加到RoaringBitmap中。	6 | roaringbitmap('{1,2,3}')	{1,2,3,6}
#	roaringbitmap,roaringbitmap	roaringbitmap	位元互斥或运算。	roaringbitmap('{1,2,3}') # roaringbitmap('{3,4,5}')	{1,2,4,5}
<<	roaringbitmap,bigint	roaringbitmap	按位左移。	roaringbitmap('{1,2,3}') << 2	{0,1}
>>	roaringbitmap,bigint	roaringbitmap	按位右移。	roaringbitmap('{1,2,3}') >> 3	{4,5,6}
-	roaringbitmap,roaringbitmap	roaringbitmap	差异。	roaringbitmap('{1,2,3}') - roaringbitmap('{3,4,5}')	{1,2}
-	roaringbitmap,integer	roaringbitmap	从RoaringBitmap中删除元素。	roaringbitmap('{1,2,3}') - 3	{1,2}
@>	roaringbitmap,roaringbitmap	bool	包含。	roaringbitmap('{1,2,3}') @> roaringbitmap('{3,4,5}')	f
@>	roaringbitmap,integer	bool	包含。	roaringbitmap('{1,2,3,4,5}') @> 3	t
	roaringbitmap,roaringbitmap	bool	包含。	roaringbitmap('{1,2,3}')	f

操作符	输入	输出	描述	示例	结果
	integer,roaringbitmap	bool	包含。	3	t
&&	roaringbitmap,roaringbitmap	bool	重叠。	roaringbitmap('{1,2,3}') && roaringbitmap('{3,4,5}')	t
=	roaringbitmap,roaringbitmap	bool	等于。	roaringbitmap('{1,2,3}') = roaringbitmap('{3,4,5}')	f
<>	roaringbitmap,roaringbitmap	bool	不等于。	roaringbitmap('{1,2,3}') <> roaringbitmap('{3,4,5}')	t

功能函数列表

函数	输入	输出	描述	示例	结果
rb_build	integer[]	roaringbitmap	Create roaringbitmap from integer array	rb_build('{1,2,3,4,5}')	{1,2,3,4,5}
rb_index	roaringbitmap,integer	bigint	Return the 0-based index of element in this roaringbitmap, or -1 if do not exsits	rb_index('{1,2,3}',3)	2
rb_cardinality	roaringbitmap	bigint	Return cardinality of the roaringbitmap	rb_cardinality('{1,2,3,4,5}')	5
rb_and_cardinality	roaringbitmap,roaringbitmap	bigint	Return cardinality of the AND of two roaringbitmaps	rb_and_cardinality('{1,2,3}',rb_build('{3,4,5}'))	1
rb_or_cardinality	roaringbitmap,roaringbitmap	bigint	Return cardinality of the OR of two roaringbitmaps	rb_or_cardinality('{1,2,3}','{3,4,5}')	1
rb_xor_cardinality	roaringbitmap,roaringbitmap	bigint	Return cardinality of the XOR of two roaringbitmaps	rb_xor_cardinality('{1,2,3}','{3,4,5}')	4
rb_andnot_cardinality	roaringbitmap,roaringbitmap	bigint	Return cardinality of the ANDNOT of two roaringbitmaps	rb_andnot_cardinality('{1,2,3}','{3,4,5}')	2
rb_is_empty	roaringbitmap	boolean	Check if roaringbitmap is empty.	rb_is_empty('{1,2,3,4,5}')	t
rb_fill	roaringbitmap,range_start bigint,range_end bigint	roaringbitmap	Fill the specified range (not include the range_end)	rb_fill('{1,2,3}',5,7)	{1,2,3,5,6}
rb_clear	roaringbitmap,range_start bigint,range_end bigint	roaringbitmap	Clear the specified range (not include the range_end)	rb_clear('{1,2,3}',2,3)	{1,3}

函数	输入	输出	描述	示例	结果
rb_flip	roaringbitmap,range_start bigint,range_end bigint	roaringbitmap	Negative the specified range (not include the range_end)	rb_flip('{1,2,3}',2,10)	{1,4,5,6,7,8,9}
rb_range	roaringbitmap,range_start bigint,range_end bigint	roaringbitmap	Return new set with specified range (not include the range_end)	rb_range('{1,2,3}',2,3)	{2}
rb_range_cardinality	roaringbitmap,range_start bigint,range_end bigint	bigint	Return the cardinality of specified range (not include the range_end)	rb_range_cardinality('{1,2,3}',2,3)	1
rb_min	roaringbitmap	integer	Return the smallest offset in roaringbitmap. Return NULL if the bitmap is empty	rb_min('{1,2,3}')	1
rb_max	roaringbitmap	integer	Return the greatest offset in roaringbitmap. Return NULL if the bitmap is empty	rb_max('{1,2,3}')	3
rb_rank	roaringbitmap,integer	bigint	Return the number of elements that are smaller or equal to the specified offset	rb_rank('{1,2,3}',3)	3
rb_jaccard_dist	roaringbitmap,roaringbitmap	double precision	Return the jaccard distance(or the Jaccard similarity coefficient) of two bitmaps	rb_jaccard_dist('{1,2,3}','{3,4}')	0.25
rb_select	roaringbitmap,bitset_limit bigint,bitset_offset bigint=0,reverse boolean=false,range_start bigint=0,range_end bigint=4294967296	roaringbitmap	Return subset [bitset_offset,bitset_offset+bitset_limit) of bitmap between range [range_start,range_end)	rb_select('{1,2,3,4,5,6,7,8,9}',5,2)	{3,4,5,6,7}
rb_to_array	roaringbitmap	integer[]	Convert roaringbitmap to integer array	rb_to_array(roaringbitmap('{1,2,3}'))	{1,2,3}
rb_iterate	roaringbitmap	SET of integer	Return set of integer from a roaringbitmap data.	SELECT rb_iterate (rb_build('{1,2,3}'))	123

聚合函数列表

函数	输入	输出	描述	示例	结果
rb_build_agg	integer	roaringbitmap	Build a roaringbitmap from a integer set	select rb_build_agg(id) from (values (1),(2),(3)) t(id)	{1,2,3}
rb_or_agg	roaringbitmap	roaringbitmap	AND Aggregate calculations from a roaringbitmap set	select rb_or_agg(bitmap) from (values (roaringbitmap('{1,2,3}')), (roaringbitmap('{2,3,4}'))) t(bitmap)	{1,2,3,4}
rb_and_agg	roaringbitmap	roaringbitmap	AND Aggregate calculations from a roaringbitmap set	select rb_and_agg(bitmap) from (values (roaringbitmap('{1,2,3}')), (roaringbitmap('{2,3,4}'))) t(bitmap)	{2,3}
rb_xor_agg	roaringbitmap	roaringbitmap	XOR Aggregate calculations from a roaringbitmap set	select rb_xor_agg(bitmap) from (values (roaringbitmap('{1,2,3}')), (roaringbitmap('{2,3,4}'))) t(bitmap)	{1,4}
rb_or_cardinality_agg	roaringbitmap	bigint	OR Aggregate calculations from a roaringbitmap set, return cardinality.	select rb_or_cardinality_agg(bitmap) from (values (roaringbitmap('{1,2,3}')), (roaringbitmap('{2,3,4}'))) t(bitmap)	4
rb_and_cardinality_agg	roaringbitmap	bigint	AND Aggregate calculations from a roaringbitmap set, return cardinality	select rb_and_cardinality_agg(bitmap) from (values (roaringbitmap('{1,2,3}')), (roaringbitmap('{2,3,4}'))) t(bitmap)	2
rb_xor_cardinality_agg	roaringbitmap	bigint	XOR Aggregate calculations from a roaringbitmap set, return cardinality	select rb_xor_cardinality_agg(bitmap) from (values (roaringbitmap('{1,2,3}')), (roaringbitmap('{2,3,4}'))) t(bitmap)	2

17.跨机并行查询

17.1. 概述

PolarDB PostgreSQL引擎提供了跨机并行查询（Parallel Execution）的功能，支持多个计算节点分布式地执行SQL查询，加速PolarDB PostgreSQL引擎的分析型查询性能，充分发挥存储层PolarFileSystem的高I/O吞吐能力，以及提高所有计算节点的CPU和内存资源的使用率。

前提条件

- 您在使用跨机并行查询功能前，请先提交工单，申请为您的集群开启跨机并行查询功能。
- PolarDB PostgreSQL引擎的内核小版本需在V1.1.11以上。关于升级内核小版本，请参见版本管理。

功能优势

跨机并行查询功能具有如下优势：

- 具备一定的HTAP能力：
  - 能在TP数据上实时执行分析型查询。
  - 执行分析查询的只读节点和执行TP型查询的只读节点可以物理隔离，避免影响TP业务。
- 结合PolarDB PostgreSQL引擎存储计算分离的架构，可以做到弹性扩展：
  - 当算力不够时，可以弹性地增加只读节点，新增的只读节点加入分布式的并行计算中，而不需数据重新分片（Reshard）。
  - 不会出现数据倾斜问题。

使用场景

日常业务中的轻分析类业务，例如：对账业务。

功能简介

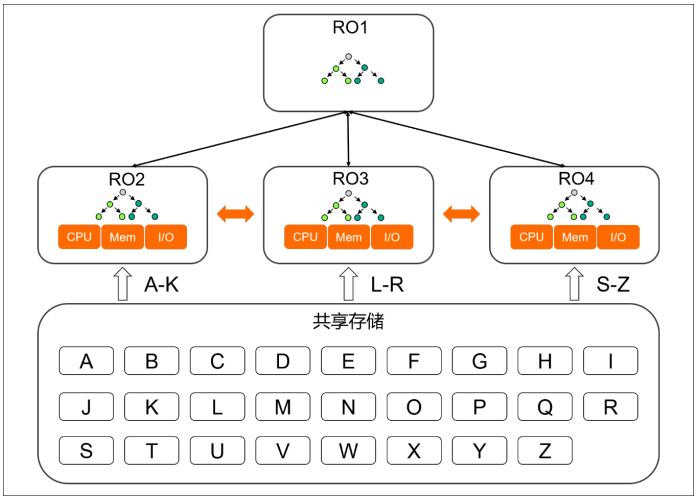
- 跨机并行查询最主要的功能即进行分析型查询，实现一定的HTAP能力。具体可参见使用跨机并行查询进行分析型查询。
- 跨机并行查询功能还可以用于加速构建索引。具体可参见使用跨机并行查询加速索引创建。
- 跨机并行查询功能可用于BRIN索引扫描。在设置polar_enable_px=on开启跨机并行查询功能后，可使用BRIN索引，进一步提升查询性能。

17.2. 使用跨机并行查询进行分析型查询

PolarDB PostgreSQL引擎支持使用跨机并行查询功能进行分析型查询，实现一定的HTAP能力。本文介绍如何使用跨机并行查询，提升分析型查询的性能。

原理介绍

当一条查询请求在查询协调节点上被执行跨机并行查询时，该查询产生的执行计划会被分片路由至各个执行节点，每个执行节点将会执行各自的分片计划，并将分片的查询结果汇总至查询协调节点。可以称查询协调节点为QC（Query Coordinator）节点，称分片计划的执行节点为PX（Parallel Execution）节点。



如上图所示，RO1为QC节点，它接收了一条查询输入请求，并将查询计划分片路由至RO2、RO3、RO4三个PX节点。每个PX节点将会针对各自接收到的分片计划，最终从PolarFS共享存储上读取到各自所需的数据块，由执行节点执行完成相应的分片计划，并将执行结果返回为QC节点，QC节点汇总后返回查询结果。

注意事项

由于跨机并行查询功能需使用多个只读节点资源，因此只适用于低频次的分析型查询。

参数说明

默认情况下，PolarDB PostgreSQL引擎不开启跨机并行查询功能。若您需要使用此功能，请使用如下参数：

参数	说明
polar_cluster_map	用于查询当前PolarDB PostgreSQL引擎所有只读节点的名称。该参数不可配置。当您新增一个只读节点时，该参数会进行更新。

polar_px_nodes	指定参与跨机并行查询的只读节点。默认为空，表示所有只读节点都参与。可配置为指定节点参与跨机并行查询，以逗号分隔。例如：
	`SHOW polar_px_nodes ;`
	`polar_px_nodes`
	`----------`
	`(1 row)`
	
	`SET polar_px_nodes='node1,node2';`
	
	`SHOW polar_px_nodes ;`
	`polar_px_nodes`
	`----------`
	`node1,node2`
	`(1 row)`

参数	说明
polar_px_enable_replay_wait	PolarDB PostgreSQL引擎的主节点与只读节点存在一定程度的延迟，当主节点执行DDL语句（例如CREATE TABLE），只读节点需要耗时回放该DDL日志后才可见新创建的表。当设置 `polar_px_enable_replay_wait` 为on后，跨机并行查询启用强一致性，当前发起的跨机并行查询请求路由到只读节点上执行时，需要只读节点回放到该查询请求前最近的一条日志后，才会执行查询请求。 该参数默认为off，即关闭强一致性，在数据库主备日志延迟较高时，不保证只读节点可以读到最近的DDL记录。您可配置 `polar_px_enable_replay_wait` 为on，表示启用强一致性，但是跨机并行查询会损失一定程度的性能。
polar_px_max_workers_number	设置单个节点上的最大跨机并行查询workers进程数，默认为30。该参数限制了单个节点上的最大并发度，节点上所有会话的跨机并行查询workers进程数不能超过该参数大小。
polar_enable_px	指定是否开启跨机并行查询功能。默认为off，即不开启。
polar_px_dop_per_node	设置当前会话并行查询的并行度，默认为1，推荐值为当前CPU总核数。若设置该参数为N，则一个会话在每个节点上将会启用N个px workers进程，用于处理当前的跨机并行查询逻辑。

px_workers	指定跨机并行查询是否对特定表生效。默认不生效。跨机并行查询功能比较消耗计算节点集群资源，因此只有对设置了px_workers的表才使用该功能。例如：
	`--表示t1表允许跨机并行查询`
	`ALTER TABLE t1 SET(px_workers=1);`
	`--表示t1表禁止跨机并行查询`
	`ALTER TABLE t1 SET(px_workers=-1);`
	`--表示t1表忽略跨机并行查询，默认状态`
	`ALTER TABLE t1 SET(px_workers=0);`

示例

本示例以简单的单表查询操作，来描述跨机并行查询的功能是否有效。

示例背景：

执行如下命令，创建test表并插入基础数据。

```
CREATE TABLE test(id int);
INSERT INTO test SELECT generate_series(1,1000000);
EXPLAIN SELECT * FROM test;
```

默认情况下跨机并行查询功能是不开启的，单表查询执行计划为原生的Seq Scan，结果如下所示。

```
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on test  (cost=0.00..35.50 rows=2550 width=4)
(1 row)
```

通过以下步骤，开启并使用跨机并行查询功能：

1. 对test表启用跨机并行查询功能。

   ```
   ALTER TABLE test SET (px_workers=1);
   SET polar_enable_px=on;
   EXPLAIN SELECT * FROM test;
   ```

   查询结果如下：

```
                                QUERY PLAN
-------------------------------------------------------------------------------
 PX Coordinator 2:1  (slice1; segments: 2)  (cost=0.00..431.00 rows=1 width=4)
   ->  Seq Scan on test (scan partial)  (cost=0.00..431.00 rows=1 width=4)
 Optimizer: PolarDB PX Optimizer
(3 rows)
```

2. 查询当前所有只读节点的名称。

   查询命令如下：

   ```
   SHOW polar_cluster_map;
   ```

   查询结果如下：

   ```
    polar_cluster_map
   -------------------
    node1,node2,node3
   (1 row)
   ```

   可得出当前集群有3个只读节点，名称分别为：node1，node2和node3。

3. 指定node1和node2只读节点参与跨机并行查询。

   命令如下：

   ```
   SET polar_px_nodes='node1,node2';
   ```

   查询参与并行查询的节点：

   ```
   SHOW polar_px_nodes ;
   ```

   查询结果如下：

   ```
    polar_px_nodes
   ----------------
    node1,node2
   (1 row)
   ```

性能数据

在5个只读节点的情况下，测试的性能数据如下：

- 在 `SELECT COUNT(*)` 扫表的场景下，跨机并行查询比单机并行查询加速60倍。
- 在TPC-H场景下，跨机并行查询比单机并行查询加速30倍。

> **说明** 此处的TPC-H场景是基于TPC-H的基准测试，并不能与已发布的TPC-H基准测试结果相比较，此处提及的测试结果并不符合TPC-H基准测试的所有要求。

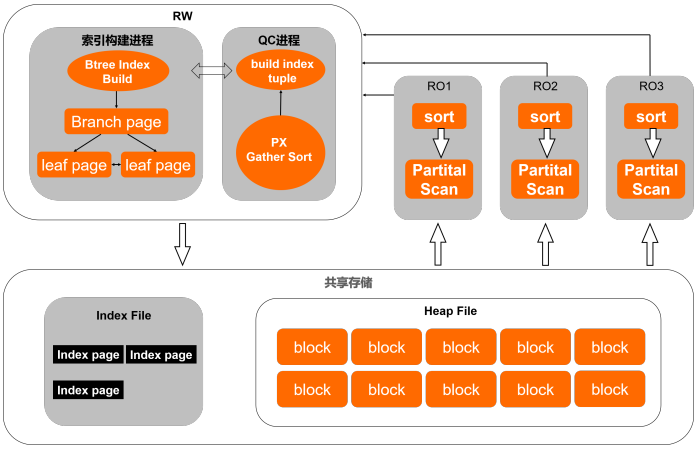
17.3. 使用跨机并行查询加速索引创建

跨机并行查询功能还可以用于加速构建B-tree索引，同时支持加速创建B-Tree索引的GLOBAL索引。本文介绍工作原理以及如何使用该功能加速索引构建。

原理介绍

PolarDB PostgreSQL引擎在执行索引构建时，会首先扫描待构建索引的基表构造出索引项，然后再进一步的根据索引项完成整棵索引树的构建过程。

当使用跨机并行查询功能加速Btree索引构建时，系统会自动构建出一个QC进程完成对基表项的并行扫描，并由索引构建进程接收QC进程扫描结果完成后续的索引创建逻辑。



注意事项

- 当前仅支持简单场景，对普通列类型的索引构建过程，暂不支持CONCURRENTLY，INCLUDE等索引构建语法。
- 暂不支持表达式等索引列类型。

参数说明

如果需要使用跨机并行查询功能加速创建索引，请使用如下参数：

参数	说明
polar_px_enable_btbuild	是否开启使用跨机并行查询加速创建索引。取值如下： • off：不开启（默认） • on：开启
polar_px_dop_per_node	指定通过跨机并行查询加速构建索引的并行度。默认为1，推荐值8或者16。 该参数同时也指定了跨机并行查询的并行度。详细信息，请参见使用跨机并行查询进行分析型查询。
polar_px_enable_replay_wait	当使用跨机并行查询加速索引构建时，当前会话内无需再手动开启polar_px_enable_replay_wait，该参数将自动生效，以便保证最近更新的数据表项可以被创建到索引中，保证索引表的完整性。索引创建完成后，该参数将会被重置为数据库默认设置。
polar_bt_write_page_buffer_size	指定索引构建过程中的写IO策略。该参数默认值为0，不开启，单位为块，最大值可设置为8192。推荐设置为4096。 • 当该参数设置为不开启时，在索引创建的过程中，对于索引页满写盘的方式采用的是block-by-block进行单个块写盘。 • 当该参数设置为开启时，则会在内核中缓存一个polar_bt_write_page_buffer_size大小的buffer，对于需要写盘的索引页，会通过该buffer进行IO合并再统一写盘，避免了频繁IO调度的性能开销。该参数会额外再带来20%的索引创建性能提升。

示例

示例背景：

执行如下命令，创建test表。

```
CREATE TABLE test(id int,id2 int);
```

查询表结构：

```
\d test
              Table "public.test"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           |          |
 id2    | integer |           |          |
```

执行如下步骤，对test表通过跨机并行查询构建索引。

1. 开启使用跨机并行查询加速创建索引功能。

   命令如下：

   ```
   SET polar_px_enable_btbuild=on;
   ```

   查看设置状态：

   ```
   SHOW polar_px_enable_btbuild;
   ```

   返回结果如下：

   ```
     polar_px_enable_btbuild
   -------------------------
    on
   (1 row)
   ```

2. 使用如下语法创建索引。

   ```
   CREATE INDEX t ON test(id) WITH(px_build=on);
   ```

   查询表结构：

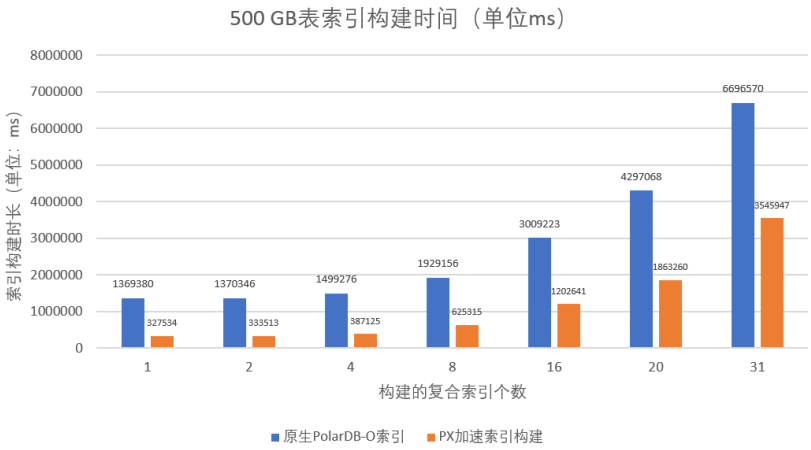
   ```
   \d test
                 Table "public.test"
    Column |  Type   | Collation | Nullable | Default
   --------+---------+-----------+----------+---------
    id     | integer |           |          |
    id2    | integer |           |          |
   Indexes:
       "t" btree (id) WITH (px_build=finish)
   ```

> **说明** 若要使用跨机并行查询加速索引创建，创建索引的语法需要添加px_build选项。
> 构建完成后，该表的索引类型会带有(px_build=finish)字段，说明该索引项是通过跨机并行查询的方式构建的。
> 如果开启polar_px_enable_btbuild，但创建索引的语法上未添加px_build选项，则会使用PolarDB PostgreSQL引擎原生的索引创建方式构建索引。示例如下：
>
> ```
> CREATE INDEX t ON test(id);
> ```
>
> 查询表结构：
>
> ```
> \d test
>               Table "public.test"
>  Column |  Type   | Collation | Nullable | Default
> --------+---------+-----------+----------+---------
>  id     | integer |           |          |
> Indexes:
>     "t" btree (id)
> ```

性能数据

使用跨机并行查询加速索引构建功能，对于大表创建索引时间，相较于原生PolarDB PostgreSQL引擎的索引创建，可缩短近5倍。

如下是500GB数据表索引构建性能对比，横坐标为构建的复合索引个数，纵坐标为索引构建时长（单位：ms）。



17.4. 对分区表使用跨机并行查询

PolarDB PostgreSQL引擎支持对分区表使用跨机并行查询的功能。通过对分区表使用跨机并行查询，提升数据库的性能。

功能介绍

当前对分区表使用跨机并行查询支持的功能如下所示：

- 支持range分区的并行查询。
- 支持list分区的并行查询。
- 支持hash分区的并行查询。
- 支持分区裁剪。
- 支持带有索引的分区表并行查询。
- 支持分区表join查询。
- 支持多级分区的并行查询

使用限制

不支持多列的hash分区并行查询。

使用指南

开启分区表跨机并行查询功能。

1. 分区表跨机并行查询功能默认关闭，需要先开启跨机并行查询功能，执行以下语句，开启该功能：

   ```
   SET polar_enable_px=on;
   ```

2. 执行以下语句，开启分区表跨机并行查询功能。

   ```
   SET polar_px_enable_partition = true;
   ```

3. 执行以下语句，开启多级分区表跨机并行查询功能。

   ```
   set  polar_px_optimizer_multilevel_partitioning = true;
   ```

18.闪回

18.1. 闪回删除

PolarDB PostgreSQL引擎中没有undo日志，开发或运维人员如果误操作（例如DROP TABLE）可能会导致数据丢失。PolarDB PostgreSQL引擎支持闪回删除功能，用于快速恢复已经删除的表，以及查看和清理回收站等，方便您找回数据。

注意事项

- 建议在删除重要表时打开闪回删除功能，日常业务操作请谨慎使用。
- 闪回删除是将已删除的表转存到回收站模式空间下，并非真正意义上的删除表，所以表所占用的空间并没有被释放。因此，数据长期积累可能会导致磁盘空间占用较大，建议定期清理回收站。
- 闪回删除功能开启状态下，表被删除后，在执行表的相关依赖项删除操作时可能会失败（因为表并未被真正删除，表依旧依赖这些依赖项）。
- 由于删除的表转存在**recyclebin**（回收站）模式空间下，因此避免创建名称为**recyclebin**的模式，因为**recyclebin**模式下的表会被 `purge recyclebin` 语法清理掉。
- 打开闪回删除功能后，不支持以下删除操作：
  - 分区表和临时表不支持闪回删除，会被真正删除，无法恢复。
  - 不支持 `sql_drop` 事件触发器。

参数说明

参数名称	数据类型	参数说明
polar_enable_flashback_drop	BOOL	打开或者关闭闪回删除功能。取值： • ON：打开。 • OFF：关闭（默认）。

语法

闪回删除功能支持以下SQL语法：

- **删除策略**

```
drop table table_name;          #放入回收站。
drop table table_name purge;    #完全删除，无法恢复。
```

- **恢复删除的表**

```
flashback table table_name to before drop;                      #恢复删除的表（重名的表则恢复最新的表）。
flashback table table_name to before drop rename to table_name_1; #恢复删除的表并重命名。
```

- **彻底删除回收站中的表**

```
purge table table_name;
```

> **说明** 重名的表则删除最旧的表。

- **清空回收站**

```
purge recyclebin;
```

> **说明** 清空回收站需要polar_super_user权限。

- **查看回收站信息**

```
show recyclebin;
```

示例

- 准备测试数据：
  创建表 `test1` 和表 `test2` ，并插入数据。

```
CREATE TABLE test1(id int primary key, name text);
CREATE TABLE test2(id int primary key, name text);
INSERT INTO test1 select t,repeat('test1',1024) from generate_series(1, 10000) as t;
INSERT INTO test2 select t,repeat('test2',1024) from generate_series(1, 10000) as t;
\dt
```

显示结果如下：

```
         List of relations
 Schema | Name  | Type  |  Owner
--------+-------+-------+----------
 public | test1 | table | postgres
 public | test2 | table | postgres
(2 rows)
```

- 删除表并查看回收站。
  - 删除表 `test1` 和表 `test2` 。

```
DROP TABLE test1;
DROP TABLE test2;
\dt
```

显示结果如下：

```
Did not find any relations.
```

  - 查看回收站。

```
show recyclebin;
```

显示结果如下：

```
 table_catalog |         table_name          | table_type
---------------+-----------------------------+------------
 postgres      | public$test1$69461331094004 | BASE TABLE
 postgres      | public$test2$69461332967609 | BASE TABLE
(2 rows)
```

- 闪回删除的表：
  - 恢复删除的表。

```
FLASHBACK TABLE test1 TO BEFORE DROP;
FLASHBACK TABLE test2 TO BEFORE DROP RENAME TO test3;
\dt
```

显示结果如下：

```
   List of relations
 Schema | Name  | Type  |  Owner
--------+-------+-------+----------
 public | test1 | table | postgres
 public | test3 | table | postgres
(2 rows)
```

> **说明** 其中表 `test2` 被重命名为 `test3` 。

- 查看恢复后的数据。
  查看表 `test1` 的数据。

  ```
  SELECT count(*) FROM test1;
  ```

  显示结果如下：

  ```
   count
  -------
   10000
  (1 row)
  ```

  查看表 `test3` 的数据。

  ```
  SELECT count(*) FROM test3;
  ```

  显示结果如下：

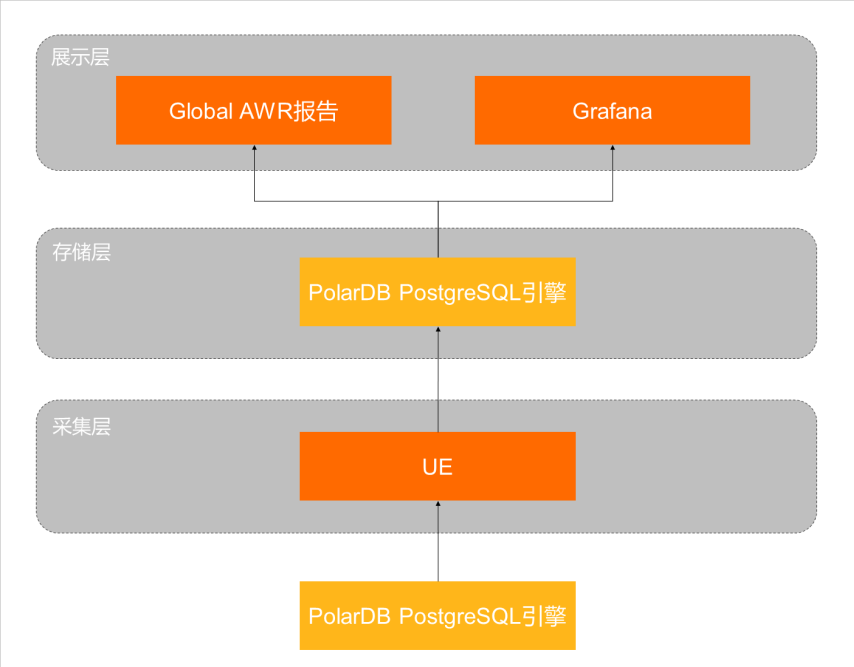
  ```
   count
  -------
   10000
  (1 row)
  ```

19.AWR使用说明

19.1. 概述

自动负载信息库AWR（Automatic Workload Repository）是一种性能收集和分析的工具。通过AWR工具，您可以从数据库的动态视图等统计信息中生成某个时间段的统计分析报告以及数据库性能报告。
PolarDB PostgreSQL引擎的Global AWR性能报告功能，在提供的数据库资源、Top SQL、Top表及索引的分析基础上，进一步拓展了传统AWR报告的边界，数据范围涵盖数据库集群所有RW/RO节点，具备集群全局视角。数据粒度将关键指标的采集细化到秒级，并提供趋势图，以方便定位性能抖动根因。

AWR架构图



AWR架构图说明

- 采集层：UE作为代理进程部署于物理机，负责采集PolarDB PostgreSQL引擎数据库实例的数据。
- 存储层：PolarDB PostgreSQL引擎数据库。UE完成采集后，您可以配置是否将采集的数据回写到对应的数据库实例。
- 展示层：输出Global AWR报告和通过Grafana查看实时数据两种形式。
  - Global AWR报告：通过SQL函数生成的HTML形式的离线报告 。
  - Grafana：在线形式的实时监控，您可以安装Grafana，并配置对应的数据源，导入dashboard配置文件即可查看实时数据。

查看方式

- 报告：通过PolarDB PostgreSQL引擎数据库可生成类似Oracle AWR的报告文件。
- Grafana：支持通过Grafana查看PolarDB PostgreSQL引擎的实时监控信息和性能趋势信息。

19.2. 使用前准备

本文介绍在使用Global AWR前需要做的准备工作。

内核版本

内核版本建议升级至20211231或以上的版本。

执行以下命令，查看当前内核版本：

```
show polar_release_date;
```

显示结果如下所示，表示当前使用的内核版本为20211231。

```
polar_release_date
--------------------
20211231
(1 row)
```

权限说明

目前所有Global AWR提供的控制权限及数据权限均为PUBLIC。

时区

PolarDB PostgreSQL引擎数据库实例默认的时区为UTC，在查看性能数据时要注意所在时区的时间差异。

启用和禁用Global AWR数据的本地存储功能

在使用AWR之前，需要启用Global AWR数据的本地存储功能。Global AWR功能启用后，无需定期触发快照也可生成细粒度的性能报告。可以在postgres库的polar_gawr_collection模式下查看数据，详情请参见数据解析。

> **说明** 对于关键指标，PolarDB PostgreSQL引擎会进行细粒度采集及写入，Global AWR功能启用后会带来两部分额外开销，请您谨慎评估是否需要启用此功能：
>
> - 性能开销：整体性能损失在10%以内。
> - 存储开销：目前默认保留3天的数据，3天最细粒度的数据量在10 GB以内。

操作说明

- 启用Global AWR数据的本地存储功能。

  ```
  SELECT polar_gawr_collection.enable_store_in_localdb();
  ```

  显示结果如下：

  ```
   enable_store_in_localdb
  -------------------------
  (1 row)
  ```

- 禁用Global AWR数据的本地存储功能。

  ```
  SELECT polar_gawr_collection.disable_store_in_localdb();
  ```

  显示结果如下：

  ```
   disable_store_in_localdb
  --------------------------
  (1 row)
  ```

- 查看Global AWR数据的本地存储启停状态。

  ```
  SELECT polar_gawr_collection.show_store_in_localdb();
  ```

  显示结果如下：

  ```
   show_store_in_localdb
  -----------------------
   t
  (1 row)
  ```

19.3. 使用说明

本文介绍AWR详细使用说明。包括输出和查看Global AWR报告，连接Grafana查看实时数据等。

生成和查看Global AWR报告

- **打印快照**

与Oracle AWR类似，PolarDB PostgreSQL引擎支持显式触发性能数据的快照。不同的是PolarDB PostgreSQL引擎Global AWR本身支持例行采集，并且为不同类型的数据设置了不同的采集间隔，无需定期触发快照也可生成性能报告。各项采集及其间隔请参见数据解析。

- **打印快照。**

  ```
  SELECT polar_gawr_collection.snapshot();
  ```

  显示结果如下：

  ```
   snapshot
  ----------
          2
  (1 row)
  ```

  > **说明** 打印快照为异步触发，触发后最多等待10秒即可生成完整快照。

- **查看快照列表。**

  ```
  SELECT polar_gawr_collection.list_snapshots();
  ```

  显示结果如下：

  ```
           list_snapshots
  ----------------------------------
   (1,"2021-08-11 07:06:53.678668")
   (2,"2021-08-11 11:18:12.747412")
  (2 rows)
  ```

- **输出报告**
  - **语法**

    ```
    psql -Aqtc "SELECT polar_gawr_report.make_report()" > index.html
    ```

  - **输出方式**

    推荐使用psql客户端进行操作，并在执行命令时添加以下参数。
    - -A，--no-align：输出结果紧凑，不做对齐。
    - -q，--quiet：不输出查询结果以外的内容。
    - -t，--tuples-only：只显示结果行，不显示列名。
    - -c，--command=COMMAND：只执行单条SQL并退出。

- **输出函数**
  所有输出报告的相关函数均定义在postgres库中的 `polar_gawr_report` 模式下，以下介绍3个重载的make_report函数。
  - 默认打印当前实例在begin_time到end_time之间的性能报告。

    ```
    polar_gawr_report.make_report(
          begin_time TIMESTAMP WITH TIME ZONE,
          end_time TIMESTAMP WITH TIME ZONE,
          ins_id TEXT DEFAULT '',
          comment TEXT DEFAULT '')
    RETURNS TEXT
    ```

  - 打印当前实例在最近time_range时间范围内的性能报告，time_range默认为1小时。

    ```
    polar_gawr_report.make_report(
          ins_id TEXT DEFAULT '',
          time_range INTERVAL DEFAULT '1 hour'::INTERVAL,
          comment TEXT DEFAULT '')
    RETURNS TEXT
    ```

  - 打印两个指定snapshot id间的性能报告， `snapshot` 为异步触发，在生成报告时会将结束时间默认延后五分钟，以便获取到完整的数据。

    ```
    polar_gawr_report.make_report(
        begin_snapshot_no INTEGER, end_snapshot_no INTEGER,
        end_snapshot_delta_time INTERVAL DEFAULT '5 min',
        ins_id TEXT DEFAULT '', comment TEXT DEFAULT '')
    RETURNS TEXT
    ```

  AWR提供各种参数来控制报告显示的内容并跟踪报告输出过程，下表列出几个重要的参数，所有参数均通过 `SET` 命令设置其参数值。更多参数说明

参数	默认值	说明
polar_gawr_report.enable_role_ro	off	同时打印RO节点的报告信息。
polar_gawr_report.log_full_error	off	显示报告打印过程中完整的错误信息。
polar_gawr_report.log_error_sql	off	显示与错误相关的SQL。
polar_gawr_report.log_timing	off	显示输出报告过程中每一个查询SQL的耗时时间。

- **示例**
  目前仅介绍使用psql客户端打印性能报告的方法。
  - 基本模式：默认打印RW节点最近一小时的性能报告。

    ```
    SELECT polar_gawr_report.make_report()
    ```

  - 打印RW节点指定时间段报告。

    ```
    SELECT polar_gawr_report.make_report(begin_time => '2021-08-12 19:00:00', end_time => '2021-08-12 19
    :30:00')"
    ```

  - 打印RW节点和RO节点的报告信息：打印报告默认不带RO，需要通过参数显式打开。

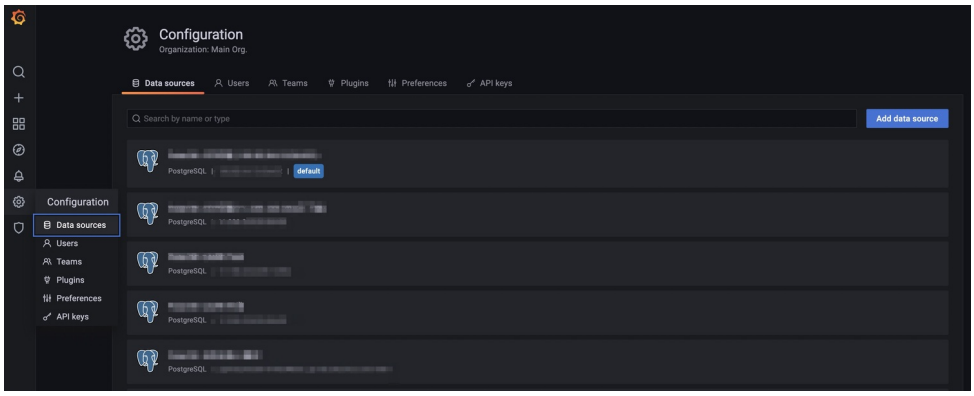
    ```
    SET polar_gawr_report.enable_role_ro='on'; SELECT polar_gawr_report.make_report()
    ```

- **查看Global AWR报告。**
  使用Global AWR生成的报告为HTML文件，可以使用浏览器（建议使用Google Chrome）打开文件，或者开启一个HTTP服务远程访问此文件。简单的HTTP服务远程访问HTML文件方式如下：

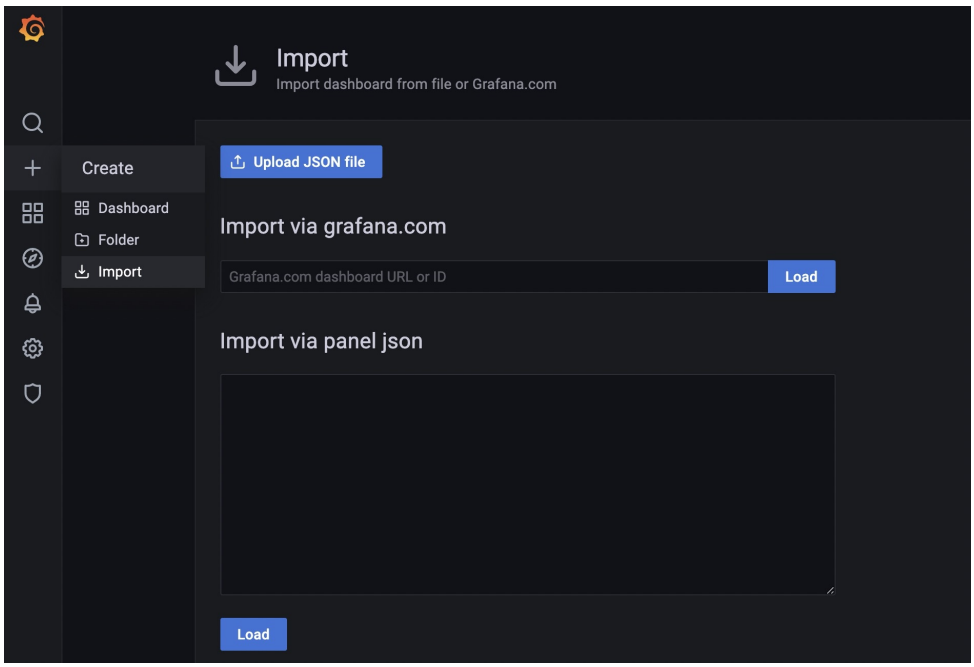
  ```
   # 在index.html所在目录下执行如下命令，提供访问端口为7070的HTTP服务
  nohup python -m SimpleHTTPServer 7070 &
  ```

连接Grafana查看实时数据

1. 下载并安装Grafana。
   详情请参见Download Grafana，建议使用8.1.2或以上版本。
2. 添加数据源。
   根据实际情况将PolarDB PostgreSQL引擎数据库实例添加为PostgreSQL数据源。
   i. 打开Grafana，在左侧导航栏中单击**Configuration > Data sources**。
   ii. 在打开的**Data sources**页面，单击右侧**Add data source**添加数据源。



3. 导入Dashboard配置文件。
   i. 在左侧导航栏中单击**+ > Import**。
   ii. 在打开的**Import**页面，单击**Upload JSON file**导入配置文件。



> **说明** dashboard配置文件压缩包请参见dashboard配置文件。配置文件下载后解压缩即可使用。

19.4. 附录

19.4.1. 报告解析

本文介绍AWR报告各部分的详细信息。

报告元信息

报告元信息包括头部表格和报告正文。

- 头部表格包含数据库集群唯一标识和报告时间段等信息。
- 报告正文分为两层标签页。
  - 组件标签：目前报告只有数据库。
  - 实例标签：以物理实例唯一标识为名称，每个RW/RO单独一个标签页。


PolarDB 数据库负载报告

实例	
时间段	2021-08-05 11:03:50 ~ 2021-08-05 11:18:50
备注	

数据库

1016 1020

基本信息

基本信息包含数据库的元信息、数据库的状态信息、操作系统地元信息以及数据库的监控汇总。

1 基本信息

1.1 数据库信息

时间段	主机	角色	内核版本
2021-08-05 11:03:52 ~ 2021-08-05 11:18:47	10.	RW	release_date: 20201230, polar_version: 2.1.8.3

CPU规格(核数)	普通内存规格(GB)	文件系统
3	6	PFS

1.2 系统信息

主机	操作系统
10.	Linux 3.10.0-327.ali2019.alios7.x86_64

CPU架构	CPU型号	逻辑CPU核数	NUMA nodes	总内存(GB)	大页内存页面(MB)	大页内存总量(GB)
x86_64	Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz	96	1	755	2	583

1.3 监控汇总

监控汇总分为负载相关、内存相关和存储相关等信息，可根据不同资源瓶颈具体查看，其中负载相关包含DB Time、DB CPU等与Oracle相同的关键指标。

1.3.1 负载相关

监控项	平均值	95分位值	峰值	区间累计值
CPU(%)	189.09	276.20	319	
数据文件读(MB/s)	271.09	695.96	883	24398
数据文件写(MB/s)	1004.86	1421.43	1501	90438
WAL写(MB/s)	595.52	1549.30	1714	53597
DB Time	160.39	202.00	203	
DB CPU	4.73	9.05	13	
IO Waits	0.76	3.00	4	
LWLock Waits	145.77	191.05	196	
Lock Waits	8.43	19.00	22	
QPS	38479.01	64520.70	71370	
TPS	4232.17	24805.15	25701	
回滚事务数	3.56	7.00	9	
逻辑读	333375.84	560778.65	684246	
物理读	3066.59	4667.95	91474	
缓存命中率(%)	99.22	99.74	100	
backend写	33.77	131.15	709	
bgwriter写	12924.18	18638.95	22542	
checkpoint写	376.61	917.55	1358	
返回行数	424060.44	979918.50	1084757	
扫描行数	674213.14	1208238.95	1298307	
插入行数	5240.15	9359.00	10644	
更新行数	10376.84	19056.30	22102	
删除行数	403.06	750.50	920	

总目录

监控汇总之后会生成一个目录，方便您查看信息。

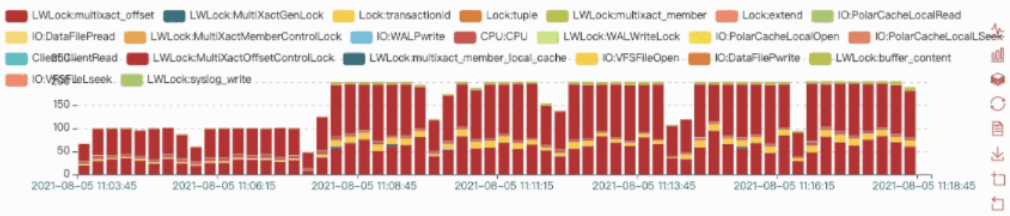
总目录

性能趋势

对于关键资源指标，给出细粒度的变化趋势会更有助于问题排查。建议的排查思路是以等待事件为中心查看高等待事件类型，再进行有针对性的进一步排查。

2.1 等待事件

等待事件类型	等待事件	平均每秒会话数(AAS)
LWLock	MultiXactOffsetControlLock	88.11
LWLock	multixact_offset	52.2
Lock	transactionid	8.02
CPU	CPU	4.93
LWLock	buffer_content	4.06
Lock	tuple	0.7
Client	ClientRead	0.5
IO	DataFilePread	0.31
LWLock	MultiXactGenLock	0.31
LWLock	WALWriteLock	0.24
IO	WALPwrite	0.2
IO	PolarCacheLocalRead	0.08
IO	VFSFileOpen	0.08
LWLock	syslog_write	0.04
LWLock	MultiXactMemberControlLock	0.04
IO	PolarCacheLocalOpen	0.02
Lock	extend	0.01
IO	VFSFileLseek	0.01
LWLock	multixact_member	0.01
IO	DataFilePwrite	0.01
LWLock	multixact_member_local_cache	0.0
IO	PolarCacheLocalLSeek	0.0

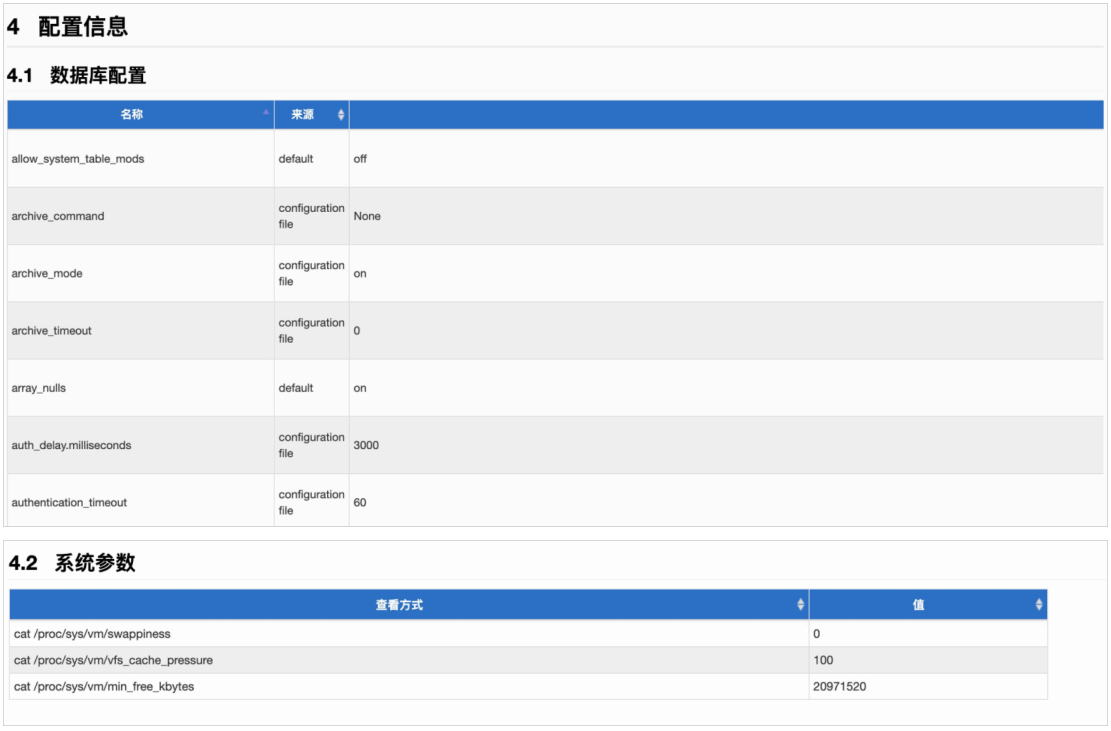


详细数据

趋势的表达方式仅能表示有限的数据，表达方式也比较单一。所以对于一些更详细的数据，会给出表格汇总形式，包括各个独立模块的监控信息、Top SQL、表和索引的监控信息等。

配置信息

提供数据库参数以及操作系统关键参数的快照。



4 配置信息

4.1 数据库配置

名称	来源	
allow_system_table_mods	default	off
archive_command	configuration file	None
archive_mode	configuration file	on
archive_timeout	configuration file	0
array_nulls	default	on
auth_delay.milliseconds	configuration file	3000
authentication_timeout	configuration file	60

4.2 系统参数

查看方式	值
cat /proc/sys/vm/swappiness	0
cat /proc/sys/vm/vfs_cache_pressure	100
cat /proc/sys/vm/min_free_kbytes	20971520

19.4.2. 数据解析

本文介绍数据模型、数据存储模式及其对应的数据详细说明。

数据模型及存储模式

为了让数据更灵活的在PolarDB PostgreSQL引擎数据库和时序数据库中进行存储，阿里云对监控数据模型进行了抽象，与数据仓库的基本概念一样，会为每一个主题的数据单独建立一张表，称之为数据模型。该表具有三种类型的列：维度列、指标列和时间列，其中指标列即为采集指标，为数值类型；维度列是对应指标列的描述，一般为字符串类型，在时序数据库中其表现形式为tag或label字段。

在实际存储中，所有监控数据均存储在postgres库中的 `polar_gawr_collection` 模式下：

- 以 `fact_` 开头的表即为一个数据模型，维度列、指标列和时间列分别有以下对应关系：
  - 以 `tag` 开头的列，为维度列，一般为TEXT类型，如果为INTEGER类型，则会和对应的维度表相关联。
  - 以 `value_` 开头的列，为指标列，为INT8类型。
  - time列为时间列，存储秒级时间戳，使用to_timestamp函数可以还原出可读时间。
- 以 `dim_` 开头的表为维度表，其可与对应fact表中的对应列相关联，其中 `dim_ins_info` 维度表较为特殊，它会与每一个fact表的 `tag_ins_info` 列相关联，标识采集元信息。
- 以 `agg_` 开头的表为fact表的聚合表，其末尾表示聚合粒度，单位为秒。
- 以 `view_` 开头的是视图，它会将 `fact_` 表和 `dim_` 表做好关联，如果需要查询，只需查视图即可。

示例

以 `polar_stat_process` 为例，在 `postgres` 库的 `polar_gawr_collection` 模式下存在如下表 `fact_polar_stat_process` 。

```
select * from polar_gawr_collection.fact_polar_stat_process limit 1;
```

显示结果如下：

```
-[ RECORD 1 ]-----------------+-----------
tag_backend_type              | 2
tag_ins_info                  | 3
time                          | 1645010019
value_backend_num             | 1
value_cpu_sys                 | 0.1
value_cpu_user                | 0.1
value_local_read_ps           | 0
value_local_read_throughput   | 0
value_local_write_ps          | 0
value_local_write_throughput  | 0
value_rss                     | 2
value_shared_read_ps          | 0
value_shared_read_throughput  | 0
value_shared_write_ps         | 0
value_shared_write_throughput | 0
```

`value_` 开头的列为指标列。 `tag_` 开头的列为维度列，INTEGER类型的维度列可以和 `dim_polar_stat_process_backend_type` 的ID列做关联：

```
select * from polar_gawr_collection_db_lq$pg$2.dim_polar_stat_process_backend_type where id = 2;
```

显示结果如下：

```
-[ RECORD 1 ]+--------------------------
id           | 2
backend_type | background logindex writer
```

可以在 `view_fact_polar_stat_process` 表查看已关联好的结果。

```
select * from polar_gawr_collection.view_fact_polar_stat_process limit 1;
```

显示结果如下：

```
-[ RECORD 1 ]-----------+---------------------------------------------
logical_ins_name        | xxxxxx
physical_ins_name       | xxxxxxxxxxx
host                    | xxxxxxxxx
port                    | xxxx
role                    | Standby
version                 | release date: 20211231, polar version: 1.1.20
backend_type            | background logindex writer
time                    | 1645010019
shared_write_ps         | 0
shared_write_throughput | 0
backend_num             | 1
cpu_sys                 | 0.1
local_write_ps          | 0
local_write_throughput  | 0
local_read_ps           | 0
shared_read_ps          | 0
local_read_throughput   | 0
rss                     | 2
shared_read_throughput  | 0
cpu_user                | 0.1
```

数据详细说明

采集到的监控信息保存在数据库的表内，包括等待事件统计、数据库IO及延迟分布、各后端进程的资源统计等信息。每一类监控信息对应一张数据表，每张数据表包含维度列和指标列。

dbmetrics

dbmetrics展示了采集到的数据库系统资源消耗情况及基本监控项，其指标和描述如下：

指标	周期	单位	说明
CPU			
cpu_total_usage	5s	%	CPU总使用率。
cpu_user_usage	5s	%	用户态CPU使用率。
cpu_sys_usage	5s	%	系统态CPU使用率。
内存			
mem_total_usage	5s	%	内存使用率。
mem_total_used	5s	MB	内存使用量。
mem_rss	5s	MB	RSS使用量。
mem_cache	5s	MB	Cache使用量。
mem_mapped_file	5s	MB	mapped_file使用量。
mem_inactiverss	5s	MB	inactive rss使用量。
mem_inactivecache	5s	MB	inactive cache使用量。
文件系统			
fs_size_total	5s	MB	空间总量。
fs_size_used	5s	MB	空间使用量。
fs_size_usage	5s	%	空间使用率。
fs_inodes_total	5s	个	inode总量。
fs_inodes_used	5s	个	inode使用量。
fs_inodes_usage	5s	%	inode使用率。
polar_base_dir_size	5s	MB	数据目录空间大小。
polar_wal_dir_size	5s	MB	WAL目录空间大小。
连接状态			
active_connections	5s	个	活跃连接数。
waiting_connections	5s	个	等待状态的连接数。
等待事件			
client_waits	5s	个	等待客户端进程数。
lwlock_waits	5s	个	lwlock等待进程数。
io_waits	5s	个	等待IO进程数。

指标	周期	单位	说明
lock_waits	5s	个	lock等待进程数。
extension_waits	5s	个	插件等待进程数。
ipc_waits	5s	个	处于进程间通信的进程数。
timeout_waits	5s	个	等待超时的进程数量。
bufferpin_waits	5s	个	等待bufferpin的进程数量。
cpu_waits	5s	个	使用CPU的进程数量。
activity_waits	5s	个	当前处于空闲状态，等待变成活跃状态的进程数量。
事务数			
commits_delta	5s	个	提交的事务数。
rollbacks_delta	5s	个	回滚的事务数。
事务状态			
active_transactions	5s	个	活跃事务数。
idle_transactions	5s	个	空闲事务数。
waiting_transactions	5s	个	等待状态的事务数。
idle_connections	5s	个	空闲连接数。
two_pc_transactions	5s	个	两阶段事务数。
swell_time	5s	秒	膨胀点：当前最长事务持续时间。
SQL			
deadlocks_delta	5s	个	死锁数量。
conflicts_delta	5s	个	由于恢复冲突导致取消的查询数量。
数据库处理行数			
tup_returned_delta	5s	行	扫描行数。
tup_fetched_delta	5s	行	返回行数。
tup_inserted_delta	5s	行	插入行数。
tup_updated_delta	5s	行	更新行数。
tup_deleted_delta	5s	行	删除行数。
临时文件			
temp_files_delta	5s	个	临时文件个数。
temp_bytes_delta	5s	字节	临时文件字节数。
数据库buffer			

指标	周期	单位	说明
blks_hit_delta	5s	个	命中缓存block数量。
blks_read_delta	5s	个	物理读次数。
buffers_backend_delta	5s	个	backend写buffer数量。
buffers_alloc_delta	5s	个	buffer分配数量。
buffers_backend_fsync_delta	5s	个	backend fsync buffer数。
buffers_checkpoint_delta	5s	个	checkpoint写buffer数量。
buffers_clean_delta	5s	个	bgwriter写buffer数量。
polar_dirtypage_size	5s	个	buffer脏页数量。
polar_copybuffer_used_size	5s	个	copy buffer使用数量。
polar_copybuffer_isfull	5s	N/A	copy buffer是否满。
checkpoint			
checkpoint_sync_time_delta	5s	秒	checkpoint sync时间。
checkpoints_timed_delta	5s	次	定时checkpoint次数。
checkpoint_write_time_delta	5s	秒	checkpoint write时间。
checkpoints_req_delta	5s	个	主动请求checkpoint次数。
logindex_mem_tbl_size	5s	个	logindex table个数。
数据库年龄			
db_age	5s	xid	数据库年龄。
复制			
replay_latency_in_mb	5s	MB	备库回放延迟。
send_latency_in_mb	5s	MB	主库发送延迟。
ap_cp_latency_mb	5s	MB	回放位点与一致性位点差距。
wp_ap_latency_mb	5s	MB	写入位点与回放位点差距。
wp_cp_latency_mb	5s	MB	写入位点与一致性位点差距。

polar_stat_process

polar_stat_process展示了采集的系统资源消耗统计信息，展示的维度包括：

维度	说明
process_type	进程类型。

资源统计信息的指标信息如下：

指标	采集周期	单位	说明
cpu_user	10s	%	用户态CPU。
cpu_sys	10s	%	系统态CPU。
rss	10s	MB	RSS（实际内存占用大小）。
share_read_ps	10s	次	共享存储读次数。
share_read_throughput	10s	MB	共享存储读吞吐。
share_write_ps	10s	次	共享存储写次数。
share_write_throughput	10s	MB	共享存储写吞吐。
local_read_ps	10s	次	本地读次数。
local_read_throughput	10s	MB	本地读吞数。
local_write_ps	10s	次	本地写次数。
local_write_throughput	10s	MB	本地写吞吐。

polar_stat_io_info

polar_stat_io_info展示了采集到的数据库IO调用（例如 `falloc` 、 `fsync` 、 `read` 、 `write` 、 `creat` 、 `seek` 、 `open` 和 `close` ）信息，展示的维度如下：

维度	说明
fileloc	文件位置： • local：本地。 • pfs：pfs。
filetype	文件类型。 • DATA：数据文件。 • WAL：日志文件。

数据库IO的指标信息如下：

指标	周期	单位	说明
falloc_latency_us	1s	us	1秒内falloc调用时间的累计值。
read_throughput	1s	MB	1秒内read调用的吞吐量。
read_latency_us	1s	us	1秒内read调用时间的累计值。
creat_latency_us	1s	us	1秒内creat调用时间的累计值。
open_latency_us	1s	us	1秒内open调用时间的累计值。
write_latency_us	1s	us	1秒内write调用时间的累计值。
seek_count	1s	次	1秒内seek调用的次数。
falloc_count	1s	次	1秒内falloc调用的次数。

指标	周期	单位	说明
fsync_count	1s	次	1秒内fsync调用的次数。
fsync_latency_us	1s	us	1秒内fsync调用时间的累计值。
close_count	1s	次	1秒内close调用的次数。
read_count	1s	次	1秒内read调用的次数。
write_count	1s	次	1秒内write调用的次数。
creat_count	1s	次	1秒内creat调用的次数。
open_count	1s	次	1秒内open调用的次数。
write_throughput	1s	MB	1秒内write调用的吞吐量。
seek_latency_us	1s	us	1秒内seek调用时间的累计值。

polar_stat_io_latency

polar_stat_io_latency展示了采集的数据库IO调用（例如 `fsync` 、 `read` 、 `write` 、 `seek` 、 `open` ）延迟时间的分布信息，展示的维度包括：

维度	说明
latency	延迟分布。

数据库IO延迟分布指标信息如下：

指标	采集周期	单位	说明
seek	10s	个	处于延迟区间内的seek调用个数。
fsync	10s	个	处于延迟区间内的fsync调用个数。
read	10s	个	处于延迟区间内的read调用个数。
write	10s	个	处于延迟区间内的write调用个数。
open	10s	个	处于延迟区间内的open调用个数。

polar_stat_network

polar_stat_network展示了采集数据的网络信息，展示的维度包括：

维度	说明
client	客户端IP地址。
backend_type	客户端类型。

采集数据的网络统计指标信息如下：

指标	周期	单位	说明
client_count	10s	个	同一维度的客户端个数。
send_mb	10s	MB	发送的数据量。

指标	周期	单位	说明
recv_mb	10s	MB	接收的数据量。
retrans	10s	次	重传次数。
max_sendq	10s	个	最大发送队列长度。
max_recvq	10s	个	最大接收队列长度。
max_rtt	10s	ms	网络延迟。
max_cwnd	10s	个	最大滑动窗口大小。

polar_aas_history

polar_aas_history展示了采集的等待事件统计信息，展示的维度包括：

维度	说明
wait_event_type	等待事件类型。
wait_event	等待事件的名称。
queryid	SQL唯一标识。

展示等待事件统计的指标信息如下：

指标	采集周期	单位	说明
wait_count	1s	个	同一queryid处于同一等待事件的会话数量。

polar_stat_slow_query

polar_stat_slow_query展示了采集数据的SQL统计信息，展示的维度包括：

维度	说明
wait_event_type	等待事件类型。
wait_event	等待事件的名称。
pid	进程ID。
dbname	数据库名称。
query	SQL语句。

SQL统计的指标信息如下：

指标	周期	单位	说明
executing_time	10s	ms	截止采集时SQL语句执行时长。

polar_stat_max_memory_sql

polar_stat_max_memory_sql展示了占用大量内存的SQL统计信息，展示的维度包括：

维度	说明
dbname	数据库名称。

维度	说明
pid	进程ID。
query	SQL语句。

占用大量内存的SQL语句指标信息如下：

指标	周期	单位	说明
rss_mb	10s	MB	当前占用最大内存的SQL语句及其占用内存大小。

polar_lock_contention_history

polar_lock_contention_history展示了锁冲突信息，展示的维度包括：

维度	说明
locktype	锁类型。
lockmode	锁级别。
datname	数据库名称。
relation	表名称。
query	SQL语句。

锁冲突的指标信息如下：

指标	周期	单位	说明
total_num	60s	个	阻塞会话个数。

polar_stat_top_sql

polar_stat_top_sql展示了采集的Top SQL统计信息，展示的维度包括：

维度	说明
queryid	SQL语句的唯一标识。
query	SQL语句。

Top SQL统计信息的指标信息如下：

指标	周期	单位	说明
logical_read	600s	次	逻辑读次数。
blks_read	600s	次	物理读次数。
logical_write	600s	次	逻辑写次数。
blks_written	600s	次	物理写次数。
blk_read_time	600s	ms	物理读IO时间。
blk_write_time	600s	ms	物理写IO时间。
calls	600s	次	执行次数。

指标	周期	单位	说明
total_time	600s	ms	执行时间
rows	600s	行	访问/变更行数。

polar_stat_user_tables

polar_stat_user_tables展示了Top表访问信息，展示的维度如下：

维度	说明
dbname	数据库名称。
schemaname	schema名称。
relname	表名称。

Top表指标信息如下：

指标	周期	单位	说明
seq_scan	3600s	次	顺序扫描的次数。
seq_tup_read	3600s	行	顺序扫描的行数。
idx_scan	3600s	次	索引扫描的次数。
idx_tup_fetch	3600s	行	索引扫描的行数。
n_tup_ins	3600s	行	插入的行数。
n_tup_upd	3600s	行	更新的行数。
n_tup_del	3600s	行	删除的行数。
n_tup_hot_upd	3600s	行	HOT更新的行数。
n_mod_since_analyze	3600s	行	上次analyze后更新的行数。
n_live_tup	3600s	个	活跃元组数量。
n_dead_tup	3600s	个	死亡元组数量。
dead_tup_ratio	3600s	%	死亡元组比例。
last_vacuum	3600s	s	距离上次vacuum的时间。
last_autovacuum	3600s	s	距离上次auto vacuum的时间。
last_analyze	3600s	s	距离上次analyze时间。
last_autoanalyze	3600s	s	距离上次auto analyze的时间。

polar_statio_user_tables

polar_statio_user_tables展示了Top表IO信息。展示的维度包括：

维度	说明
dbname	数据库名称。
schemaname	schema名称。

维度	说明
relname	表名称。

Top表IO指标信息如下：

指标	周期	单位	说明
heap_logical_read	3600s	次	表逻辑读次数。
heap_blks_read	3600s	次	表物理读次数。
idx_logical_read	3600s	次	表索引逻辑读次数。
idx_blks_read	3600s	次	表索引物理读次数。

polar_stat_user_indexes

polar_stat_user_indexes展示了Top索引访问信息，展示的维度包括：

维度	说明
dbname	数据库名称。
schemaname	schema名称。
relname	表名称。
indexrelname	索引名称。

Top索引访问指标信息如下：

指标	周期	单位	说明
idx_scan	3600s	次	索引扫描次数。
idx_tup_read	3600s	行	索引读取行数。
idx_tup_fetch	3600s	行	索引扫描行数。

polar_statio_user_indexes

polar_statio_user_indexes展示了Top索引IO信息，展示的维度包括：

维度	说明
dbname	数据库名称。
schemaname	schema名称。
relname	表名称。
indexrelname	索引名称。

Top索引IO指标信息如下：

指标	周期	单位	说明
idx_blks_read	3600s	次	索引物理读次数。

指标	周期	单位	说明
idx_logical_read	3600s	次	索引逻辑读次数。

polar_stat_relation_size

polar_stat_relation_size展示了Top relation占用空间大小信息，展示的维度包括：

维度	说明
dbname	数据库名称。
schemaname	schema名称。
relname	表名称。
relkind	表类型。

Top relation占用空间指标信息如下：

指标	周期	单位	说明
relsize_total	3600s	MB	占用空间大小。

polar_stat_table_age

polar_stat_table_age展示了Top relation年限大小信息，展示的维度包括：

维度	说明
dbname	数据库名称。
schemaname	schema名称。
relname	表名称。
relkind	表类型。

Top relation年限大小指标信息如下：

指标	周期	单位	说明
table_age	3600s	xid	年限。

polar_statio_user_sequences

polar_statio_user_sequences展示了Top sequence IO信息，展示的维度包括：

维度	说明
dbname	数据库名称。
schemaname	schema名称。
relname	表名称。
indexrelname	索引名称。

Top sequence IO的指标信息如下：

指标	周期	单位	说明
blks_read	3600s	次	物理读次数。
logical_read	3600s	次	逻辑读次数。

polar_stat_user_functions

polar_stat_user_functions展示了Top函数执行信息，展示的维度包括：

维度	说明
dbname	数据库名称。
schemaname	schema名称。
funcname	函数名称。

Top函数的指标信息如下：

指标	周期	单位	说明
calls	3600s	次	累计执行次数。
total_time	3600s	ms	累计执行时间。

polar_settings

polar_settings展示了数据库的配置信息，展示的维度包括：

维度	说明
name	配置项名称。
source	设置来源。
setting	配置项的值。

20.版本管理

PolarDB集群架构共三层：数据库代理Proxy、数据库内核引擎DB和数据库分布式存储Store。您可以根据实际情况单独升级Proxy或内核引擎，也可以绑定一起升级。

注意事项

- 版本升级一般不超过30分钟，升级过程中会重启数据库代理Proxy或内核引擎DB，可能会导致数据库连接闪断。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。
- **同时升级数据库代理（Proxy）和内核引擎**期间，主地址和集群地址均会有30～90秒的连接闪断，请确保应用具备重连机制。
- **仅升级数据库代理（Proxy）**期间，集群地址和自定义地址会有30秒的连接闪断，主地址的连接不受影响，请确保应用具备重连机制。
- **仅升级内核引擎**期间，数据库代理Proxy的版本高于2.4.7（包含）的PolarDB集群可以通过Connection Preserving技术保护95%的数据库连接不中断。
- 升级过程中无法使用PolarDB控制台的部分变更类功能（如升降配置、增删节点、修改参数、重启节点），但查询类功能（如性能监控）不受影响。
- 版本升级后无法降级。

查看版本信息

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，选择**配置与管理 > 版本管理**。
5. 在**版本信息**区域，查看数据库代理Proxy和内核引擎DB的版本信息。

升级版本

若集群当前数据库代理Proxy或内核引擎DB的版本不是最新版本，则可以根据实际需要进行升级操作。

1. 进入目标集群的**配置与管理 > 版本管理**菜单，在**升级版本**区域，您可以根据需要选择**同时升级数据库代理（Proxy）和内核引擎**、**仅升级内核引擎**或**仅升级数据库代理（Proxy）**。

> **说明**
>
> - 若集群当前数据库代理Proxy或内核引擎DB的版本已经是最新版本，则**同时升级数据库代理（Proxy）和内核引擎**、**仅升级内核引擎**或**仅升级数据库代理（Proxy）**选项将置灰不可选。
> - 若选择**仅升级数据库代理（Proxy）**，将只升级读写分离相关功能，例如一致性级别（全局一致性）、事务拆分、主库是否接受读等。

2. 单击**立即升级**或**可维护窗口升级**。

   若选择在**可维护窗口升级**，您还可以在**计划任务**页查看该任务的具体信息或取消该任务。

> 注意
>
> - **同时升级数据库代理（Proxy）和内核引擎**期间，主地址和集群地址均会有30～90秒的连接闪断，请确保应用具备重连机制。
> - **仅升级数据库代理（Proxy）**期间，集群地址和自定义地址会有30秒的连接闪断，主地址的连接不受影响，请确保应用具备重连机制。

3. 在弹出的对话框中，单击**确定**即可。

相关API

API	描述
DescribeDBClusterVersion	查看集群当前内核版本的详细信息。
UpgradeDBClusterVersion	将集群更新至最新版本。

21.错误码

本文将为您介绍PolarDB PostgreSQL引擎错误码。

错误码格式

PolarDB PostgreSQL引擎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。

Class 00 — Successful Completion

Error Code	Condition Name
00000	successful_completion

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
01P01	deprecated_feature

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

lass 03 — SQL Statement Not Yet Complete

Error Code	Condition Name
03000	sql_statement_not_yet_complete

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

Class 09 — Triggered Action Exception

Error Code	Condition Name
09000	triggered_action_exception

Class 0A — Feature Not Supported

Error Code	Condition Name
0A000	feature_not_supported

Class 0B — Invalid Transaction Initiation

Error Code	Condition Name
0B000	invalid_transaction_initiation

Class 0F — Locator Exception

Error Code	Condition Name
0F000	locator_exception
0F001	invalid_locator_specification

Class 0L — Invalid Grantor

Error Code	Condition Name
0L000	invalid_grantor
0LP01	invalid_grant_operation

Class 0P — Invalid Role Specification

Error Code	Condition Name
0P000	invalid_role_specification

Class 0Z — Diagnostics Exception

Error Code	Condition Name
0Z000	diagnostics_exception
0Z002	stacked_diagnostics_accessed_without_active_handler

Class 20 — Case Not Found

Error Code	Condition Name
20000	case_not_found

Class 21 — Cardinality Violation

Error Code	Condition Name
21000	cardinality_violation

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

Error Code	Condition Name
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
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

Error Code	Condition Name
2200T	invalid_xml_processing_instruction

Class 23 — Integrity Constraint Violation

Error Code	Condition Name
23000	integrity_constraint_violation
23001	restrict_violation
23502	not_null_violation
23503	foreign_key_violation
23505	unique_violation
23514	check_violation
23P01	exclusion_violation

Class 24 — Invalid Cursor State

Error Code	Condition Name
24000	invalid_cursor_state

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

Class 26 — Invalid SQL Statement Name

Error Code	Condition Name
26000	invalid_sql_statement_name

Class 27 — Triggered Data Change Violation

Error Code	Condition Name
27000	triggered_data_change_violation

Class 28 — Invalid Authorization Specification

Error Code	Condition Name
28000	invalid_authorization_specification
28P01	invalid_password

Class 2B — Dependent Privilege Descriptors Still Exist

Error Code	Condition Name
2B000	dependent_privilege_descriptors_still_exist
2BP01	dependent_objects_still_exist

Class 2D — Invalid Transaction Termination

Error Code	Condition Name
2D000	invalid_transaction_termination

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

Class 34 — Invalid Cursor Name

Error Code	Condition Name
34000	invalid_cursor_name

Class 38 — External Routine Exception

Error Code	Condition Name
38000	external_routine_exception
38001	containing_sql_not_permitted

Error Code	Condition Name
38002	modifying_sql_data_not_permitted
38003	prohibited_sql_statement_attempted
38004	reading_sql_data_not_permitted

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

Class 3B — Savepoint Exception

Error Code	Condition Name
3B000	savepoint_exception
3B001	invalid_savepoint_specification

Class 3D — Invalid Catalog Name

Error Code	Condition Name
3D000	invalid_catalog_name

Class 3F — Invalid Schema Name

Error Code	Condition Name
3F000	invalid_schema_name

Class 40 — Transaction Rollback

Error Code	Condition Name
40000	transaction_rollback
40002	transaction_integrity_constraint_violation
40001	serialization_failure
40003	statement_completion_unknown
40P01	deadlock_detected

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
42723	duplicate_function
42P05	duplicate_prepared_statement
42P06	duplicate_schema
42P07	duplicate_table
42712	duplicate_alias

Error Code	Condition Name
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

Class 44 — WITH CHECK OPTION Violation

Error Code	Condition Name
44000	with_check_option_violation

Class 53 — Insufficient Resources

Error Code	Condition Name
53000	insufficient_resources
53100	disk_full
53200	out_of_memory
53300	too_many_connections
53400	configuration_limit_exceeded

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

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

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

Class 58 — System Error (errors external to PostgreSQL itself)

Error Code	Condition Name
58000	system_error
58030	io_error
58P01	undefined_file
58P02	duplicate_file

Class 72 — Snapshot Failure

Error Code	Condition Name
72000	snapshot_too_old

Class F0 — Configuration File Error

Error Code	Condition Name
F0000	config_file_error
F0001	lock_file_exists

Class HV — Foreign Data Wrapper Error (SQL/MED)

Error Code	Condition Name
HV000	fdw_error

Error Code	Condition Name
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
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

Class P0 — PL/pgSQL Error

Error Code	Condition Name
P0000	plpgsql_error
P0001	raise_exception

Error Code	Condition Name
P0002	no_data_found
P0003	too_many_rows
P0004	assert_failure

Class XX — Internal Error

Error Code	Condition Name
XX000	internal_error
XX001	data_corrupted
XX002	index_corrupted

22.常见错误处理方法

本文为您介绍PolarDB PostgreSQL引擎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。

23.指定服务器编码格式

您可以指定数据库中服务器的编码格式，作为数据库数据的存储格式。本文以示例的形式介绍PolarDB兼容的GBK与GB18030两种编码格式的使用方法。

简介

- **GBK**
  GBK（Chinese Internal Code Specification）即汉字内码扩展规范 。GBK共收录21886个汉字和图形符号，包括：
  - GB2312中的全部汉字和非汉字符号。
  - BIG5中的全部汉字。
  - 国家标准GB13000中的其它CJK汉字。
  - 其它汉字、部首和符号。
- **GB18030**
  GB18030即国家标准GB18030-2005《信息技术中文编码字符集》，是GB18030-2000《信息技术信息交换用汉字编码字符集基本集的扩充》的修订版，也是最新的汉字编码字符集国家标准。向下兼容GBK和GB2312标准，共收录70244个汉字。且采用多字节编码，每个字可以由单字节、双字节或四字节组成。其中单字节、双字节和GBK完全兼容。四字节编码的码位为CJK扩展A的6582个汉字。编码空间庞大，最多可定义161万个字符。支持中国内地所有的文字。汉字收录范围包含繁体汉字以及日韩汉字。

示例

- 指定GBK编码格式。
  i. 创建数据库 `db_gbk` ，并指定服务器编码格式为GBK，客户端编码格式为UTF-8。

  ```
  CREATE DATABASE db_gbk template template0 encoding = gbk LC_COLLATE = 'zh_CN.gbk' LC_CTYPE = 'zh_CN.gbk';
  \c db_gbk;
  SET client_encoding = 'utf-8';
  ```

  ii. 创建 `test_table` 表。

  ```
  CREATE TABLE test_table(f1 varchar(3));
  ```

  iii. 向 `test_table` 表中插入数据。

  ```
  INSERT INTO test_table (f1) VALUES ('张三');
  INSERT INTO test_table (f1) VALUES ('李四');
  INSERT INTO test_table (f1) VALUES ('王先生');
  INSERT INTO test_table (f1) VALUES ('张女士');
  ```

  iv. 查询 `test_table` 表中的全量数据，并按照升序排列。

  ```
  SELECT * FROM test_table ORDER BY f1;
  ```

  查询结果如下：

  ```
  李四
  王先生
  张女士
  张三
  (4 rows)
  ```

- 指定GB18030编码格式。
  i. 创建数据库 `db_gb18030` ，并指定服务器编码格式为GB18030，客户端编码格式为UTF-8。

  ```
  CREATE DATABASE db_gb18030 template template0 encoding = gb18030 LC_COLLATE = 'zh_CN.gb18030' LC_CTYPE = 'zh_CN.gb18030';
  \c db_gb18030;
  SET client_encoding = 'utf-8';
  ```

ii. 创建 `polar_text` 表。

```
CREATE TABLE polar_text(i int, f1 text);
```

iii. 向 `polar_text` 表中插入数据。

```
INSERT INTO polar_text (f1) VALUES ('张三');
INSERT INTO polar_text (f1) VALUES ('李四');
INSERT INTO polar_text (f1) VALUES ('王先生');
INSERT INTO polar_text (f1) VALUES ('€张女士');
```

iv. 查询 `polar_text` 表中的全量数据，并升序排列。

```
SELECT * FROM polar_text ORDER BY f1;
```

查询结果如下：

```
 i |     f1
---+------------
   | 李四
   | 王先生
   | €张女士
   | 张三
(4 rows)
```

24.更多操作

24.1. 克隆集群

您可以根据源PolarDB集群的数据克隆一个新的PolarDB集群。

注意事项

- 支持被克隆的数据包括：
  - 源集群的账号信息。
  - 若源集群在克隆前已开启TDE，则相关TDE配置也支持被克隆到新集群。
- 不支持被克隆的数据包括：
  - 源集群的参数配置。
  - 源集群的白名单配置。
  - 源集群的SSL配置。
- 仅克隆集群开始创建前的数据支持被克隆，开始克隆后才写入源集群的数据将不会被克隆。

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. 找到目标集群，单击**操作**列的**更多 > 克隆数据到新集群**。
4. 在配置页面设置以下参数：

参数	说明
付费模式	您可以选择**包年包月**或**按量付费**。 ○ **包年包月**：在创建集群时支付计算节点的费用，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。 ○ **按量付费**：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。
克隆类型	默认选择**克隆一个独立集群**，无需修改。
地域	指克隆集群所在的地理位置，克隆集群和源集群的地域默认保持一致。例如源集群的地域为**华东1（杭州）**，则此处克隆集群的地域也为**华东1（杭州）**，无需选择。
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	仅支持**专有网络**VPC（Virtual Private Cloud），无需选择。
VPC网络 **VPC交换机**	从下拉菜单中选择VPC和交换机，若您没有任何VPC网络，请先创建新的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发挥最佳性能。
兼容性	克隆集群和源集群的兼容性默认保持一致。例如源集群的**兼容性**为**PostgreSQL 11**，则此处克隆集群的**兼容性**也为**PostgreSQL 11**，无需选择。
系列	克隆集群和源集群的系列默认保持一致。例如源集群的**系列**为**集群版（2-16个节点）（默认推荐）**，则此处克隆集群的**系列**也为**集群版（2-16个节点）（默认推荐）**，无需选择。
节点规格	若您的源集群**系列**为**集群版（2-16个节点）（默认推荐）**，您可以按需选择克隆集群的节点规格，不同规格有不同的最大存储容量和性能，具体请参见规格与定价。

参数	说明
节点个数	○ 若您的源集群**系列**为**集群版（2-16个节点）（默认推荐）**，系统将默认创建规格相同的两个节点（一主一只读），无需选择。 ○ 若您的源集群**系列**为**单节点（入门级）**，系统将默认创建一个节点（主节点），无需选择。
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，详情请参见规格与定价。 **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。
集群名称	输入集群名称，集群名称需满足如下要求： ○ 不能以 `http://` 或 `https://` 开头。 ○ 长度为2~256个字符。 如果留空，系统将为自动生成一个集群名称，创建集群后还可以修改。
购买时长	仅当源集群**商品类型**为**包年包月**时支持该参数。
购买数量	取值范围为1~50，默认值为1。

5. 阅读并选中服务协议，单击**立即购买**。

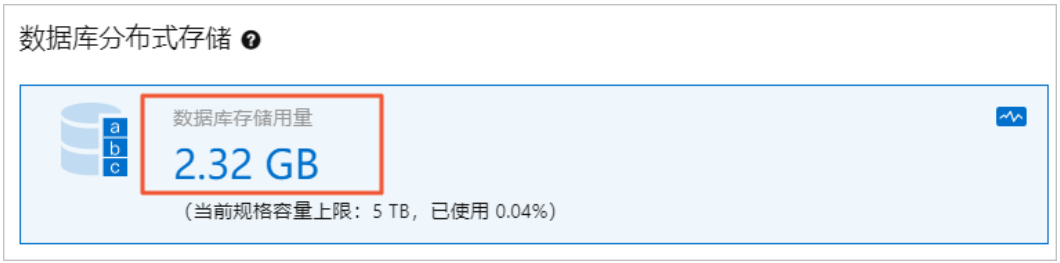
说明 订购成功后，需要1~5分钟开通服务，之后您就可以在**集群列表**中看到新克隆的集群。

24.2. 查看数据库存储用量

您可以在PolarDB控制台查看当前集群的数据库存储用量。本文将介绍如何查看数据库存储用量。

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**基本信息**页面的**数据库分布式存储**区域，查看**数据库存储用量**。



说明 每种集群规格都有对应的最大存储容量。当数据库存储用量达到当前规格容量上限的90%时，系统会每天给您发送短信和邮件通知。如需提高存储容量上限，请升级集群规格，详情请参见变更配置。

24.3. 查看或取消计划任务

您可以在执行部分运维操作（如集群升配、增加节点、升级版本和更换主可用区等）任务时，自定义任务的执行时间。任务创建成功后，您可以在控制台上查看或取消该任务。

注意事项

- 当前仅支持查看以下计划任务的详情：
  - 集群升配。操作步骤请参见操作步骤。

- 增加节点。操作步骤请参见增加只读节点。
- 升级版本。操作步骤请参见版本管理。
- 更换主可用区。操作步骤请参见多可用区部署和更换主可用区。

- 仅支持取消**任务状态**为**等待执行中**的任务。降配类（如删除节点、自动和手动降配）的计划任务不支持取消。

查看计划任务

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 在左侧导航栏中，单击**计划任务**。
4. 在**计划任务**页，您可以查看该地域下所有计划任务的详情，包括**任务ID**、**任务状态**、**任务动作**、**计划起始时间**、**计划结束时间**和**计划执行时间**等信息。

> **说明**
> - **任务动作**即该任务对应的API接口，当前仅支持以下**任务动作**：
>   - ModifyDBClusterPrimaryZone（更换主可用区）
>   - ModifyDBNodeClass（集群升配）
>   - CreateDBNodes（增加节点）
>   - UpgradeDBClusterVersion（升级版本）
> - 仅当**任务动作**为ModifyDBNodeClass或CreateDBNodes时，支持查看**订单ID**信息。

取消计划任务

1. 登录PolarDB控制台。
2. 在控制台左上角，选择集群所在地域。
3. 在左侧导航栏中，单击**计划任务**。
4. 在**计划任务**页找到目标计划任务，单击右侧**操作**栏中的**取消**。

> **说明** 仅支持取消**任务状态**为**等待执行中**的任务。降配类（如删除节点、自动和手动降配）的计划任务不支持取消。

5. 在弹出的对话框中，单击**确定**即可。

相关API

API	描述
DescribeScheduleTasks	查看当前账号下所有或指定的计划任务详情。
CancelScheduleTasks	取消目标计划任务。