文档数据库服务

最佳实践

文档版本 01

发布日期 2025-12-09

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 DDS 最佳实践汇总

本手册基于华为云文档数据库服务实践所编写，用于指导您完成相关设置，购买更符合业务的数据库实例。

章节名称	简介
安全最佳实践	介绍在DDS使用过程中的安全最佳实践，旨在为提高整体安全能力提供可操作的规范性指导。
连接DDS实例的常用方式	介绍DDS的常用连接方式。
其他云MongoDB迁移到DDS	介绍DDS数据迁移的操作。
本地MongoDB迁移到DDS	
ECS自建MongoDB迁移DDS	
如何实现副本集高可用和读写分离	介绍如何连接副本集实例以实现高可用性能。
通过设置数据分片提升性能	介绍如何设置集群分片来提高数据库性能。
如何通过SQL优化来提升DDS性能	介绍DDS的使用建议。
如何规避dds mongos路由缓存缺陷	介绍如何规避集群mongos路由缓存缺陷。
排查DDS实例CPU使用率高的问题	介绍如何排查CPU使用率高的问题。
排查DDS实例内存占用较高的问题	介绍如何排查内存占用高的问题。
排查DDS实例连接数耗尽的问题	介绍如何排查连接数耗尽的问题。
创建用户并授权使用DDS只读权限	介绍如何通过IAM授权DDS只读权限。

章节名称	简介
合理使用DDL(Data Definition Languages)语句	介绍数据库中存储实体的语言，例如创建、修改和删除数据库、集合的结构。
DDS节点脱节原理和说明	介绍节点脱节原理和规避方法。
避免hideIndex导致游标失效	介绍避免hideIndex导致游标失效的场景和方法。
使用DDS存储和分析日志数据	介绍如何使用DDS存储和分析应用日志数据。
DDS查询计划及查询重规划	介绍查询计划及查询重规划原理和场景。
DDS事务与读写关注	介绍事务和读写关注。
DDS指标告警配置建议	介绍指标告警配置建议。
使用索引	介绍使用索引相关建议。

2 安全最佳实践

安全性是华为云与您的共同责任。华为云负责云服务自身的安全，提供安全的云；作为租户，您需要合理使用云服务提供的安全能力对数据进行保护，安全地使用云。详情请参见**责任共担**。

本文提供了DDS使用过程中的安全最佳实践，旨在为提高整体安全能力提供可操作的规范性指导。根据该指导文档您可以持续评估DDS的安全状态，更好地组合使用DDS提供的多种安全能力，提高对DDS的整体安全防御能力，保护存储在DDS的数据不泄露、不被篡改，以及数据传输过程中不泄露、不被篡改。

本文从以下几个维度给出建议，您可以评估DDS使用情况，并根据业务需要在本指导的基础上进行安全配置。

- **建议避免绑定EIP直接通过互联网访问DDS**
- **避免使用常用密码**
- **禁止使用默认端口号**
- **限制DDS的最大连接数**
- **关闭IPv6功能**
- **关闭脚本运行功能**
- **建议开启审计功能**
- **配置实例访问日志功能**
- **建议开启加密通信**
- **开启磁盘加密**
- **开启备份功能**
- **设置秒级监控和告警规则**
- **版本升级**
- **检查角色的合理性**

建议避免绑定 EIP 直接通过互联网访问 DDS

避免DDS部署在互联网或者DMZ里，应该将DDS部署在公司内部网络，使用路由器或者防火墙技术把DDS保护起来，避免直接绑定EIP方式从互联网访问DDS。通过这种方式防止未授权的访问及DDos攻击等。不推荐**绑定弹性公网IP**，如果业务必需，请务必**设置安全组**。

避免使用常用密码

新建/修改账号密码，密码需要为强密码，在满足安全密码复杂度要求的前提下，同时避免使用常用密码。通过这种方式防止黑客爆破密码、彩虹表攻击等。常用密码可以通过**检查弱密码**接口进行查询。

禁止使用默认端口号

MongoDB的默认端口是27017，使用默认端口容易被监听，存在安全隐患，DDS推荐使用非默认端口。详情请参见**修改数据库端口**。

限制 DDS 的最大连接数

如果DDS的连接数过高，会消耗服务器过多的资源，导致OPS（query、insert、update、delete）等操作响应变慢，同时也要根据操作系统环境来设置最大连接数**net.maxIncomingConnections**，如果高于操作系统接收的最大线程数，设置无效。详情请参见**参数调优**。

关闭 IPv6 功能

目前DDS不支持选择IPv6网段的子网，建议您在使用时创建实例时选择IPv4网段的子网。

关闭脚本运行功能

启用javascriptEnabled选项**security.javascriptEnabled**，可以在mongod 服务端运行javascript脚本，存在安全风险。禁用javascriptEnabled选项，mapreduce、group命令等将无法使用。如果您的应用中没有mapreduce等操作的需求，为了安全起见，建议关闭javascriptEnabled选项。详情请参见**参数调优**。

建议开启审计功能

审计功能可以记录用户对数据库的所有相关操作。通过查看审计日志，您可以对数据库进行安全审计、故障根因分析等操作，提高系统运维效率。详情请参见**审计日志**。

建议配置实例访问日志功能

配置访问日志后，DDS实例新生成的审计日志、错误日志和慢日志记录会上传到LTS进行管理。您可以查看DDS实例审计日志、错误日志和慢日志的详细信息，包括搜索日志、日志可视化、下载日志和查看实时日志等功能，提高系统运维效率。详情请参见**日志配置管理**。

建议开启加密通信

如果未配置SSL加密通信，那么在Mongo客户端和服务器之间传输的数据，容易受到窃听、篡改和"中间人"攻击。为了提高数据传输的安全性，建议您开启SSL加密通信。详情请参见**设置SSL数据加密**。

开启磁盘加密

建议您开启磁盘加密可以提供数据的安全性。详情请参见自**定义购买**中的“磁盘加密”说明。

开启备份功能

DDS实例支持自动备份和手动备份，您可以定期对数据库进行备份，当数据库故障或数据损坏时，可以通过备份文件恢复数据库，从而保证数据可靠性。详情请参见**数据备份**。

设置秒级监控和告警规则

DDS默认支持对实例进行监控，当监控指标的值超出设置的阈值时就会触发告警，系统会通过SMN自动发送告警通知给云账号联系人，帮助您及时了解DDS实例的运行状况。请您结合实际的业务，设置合适的监控和告警规则。详情请参见**监控与告警**。

版本升级

DDS支持**补丁升级**或者**大版本升级**，版本升级涉及新功能添加、问题修复，同时可以提升安全能力、性能水平。建议及时进行版本升级。

检查角色的合理性

DDS支持“基于角色”的方法授予账号对数据和命令的访问权限。建议管理员结合业务需要，遵从最低授权原则，创建合适的**自定义角色**，对账号进行授权。如果发现账号权限过大，请结合业务需要，对账号权限进行**更新**或者**删除**。

表 2-1 DDS 高权限角色的描述

序号	检查项	描述
1	检查在用户表中存在userAdmin角色的用户	当在admin数据库定义了userAdmin角色的用户后，该用户被提供了任何用户的任何权限的能力，即拥有这个角色的用户可以在任何数据库上定义它们自己的权限。
2	检查在用户表中存在userAdminAnyDatabase角色的用户	当定义了userAdminAnyDatabase角色的用户后,该用户被提供了任何用户的任何权限的能力,即拥有这个角色的用户可以在任何数据库上定义它们自己的权限。
3	检查拥有anyAction动作权限的角色	当定义了anyAction动作权限的角色，该角色用户拥有对应数据库的任何操作，不利于权限的管理。
4	检查拥有anyResource动作权限的角色	当定义了anyResource动作权限的角色，该角色用户拥有对应数据库的所有操作资源，不利于权限的管理。
5	检查拥有changeCustomData动作权限的角色	当定义了changeCustomData动作权限的角色，该角色用户拥有对应数据库所有用户自定义信息的更改权限，不利于权限的管理。
6	检查拥有changePassword动作权限的角色	当定义了changePassword动作权限的角色，该角色用户即可拥有对应数据库所有用户密码的更改权限，不利于权限的管理。

序号	检查项	描述
7	检查拥有createRole动作权限的角色	当定义了createRole动作权限的角色，该角色用户即可拥有对应数据库中任意创建角色的权限，不利于权限的管理。
8	检查拥有createUser动作权限的角色	当定义了createUser动作权限的角色，该角色用户即可拥有对应数据库中任意创建用户的权限，不利于权限的管理。
9	检查拥有dropRole动作权限的角色	当定义了dropRole动作权限的角色，该角色用户即可拥有对应数据库中删除任意角色的权限，不利于权限的管理。
10	检查拥有dropUser动作权限的角色	当定义了dropUser动作权限的角色，该角色用户即可拥有对应数据库中删除任意用户的权限，不利于权限的管理。
11	检查拥有grantRole动作权限的角色	当定义了grantRole动作权限的角色，该角色用户即可拥有对应数据库中任何用户授予任意角色的权限，不利于权限的管理。
12	检查拥有revokeRole动作权限的角色	当定义了revokeRole动作权限的角色，该角色用户即可拥有对应数据库中任何用户取消任意角色的权限，不利于权限的管理。
13	检查拥有authSchemaUpgrade动作权限的角色	当定义了authSchemaUpgrade动作权限的角色，该角色用户即可拥有执行authschemaupgrade的权限，不利于权限的管理。authschemaupgrade命令可以修改用户认证转换格式。
14	检查拥有closeAllDatabases动作权限的角色	当定义了closeAllDatabases动作权限的角色，该角色用户即可拥有执行closeAllDatabases命令的权限，关闭所有数据库并释放MongoDB占用的内存，不利于权限的管理。
15	检查拥有dropDatabase动作权限的角色	当定义了dropDatabase动作权限的角色，该角色用户即可拥有执行dropDatabase命令的权限，删除任意的数据库，不利于权限的管理。
16	检查拥有getParameter动作权限的角色	当定义了getParameter动作权限的角色，该角色用户即可拥有执行getParameter命令的权限，任意查看命令行选项的值，不利于权限的管理。
17	检查拥有setParameter动作权限的角色	当定义了setParameter动作权限的角色，该角色用户即可拥有执行setParameter命令的权限，任意修改命令行选项的值，不利于权限的管理。
18	检查拥有shutdown动作权限的角色	当定义了shutdown动作权限的角色，该角色用户即可拥有执行shutdown命令的权限，清除所有数据库资源，然后终止进程，不利于权限的管理。

序号	检查项	描述
19	检查拥有getCmdLineOpts动作权限的角色	当定义了getCmdLineOpts动作权限的角色，该角色用户即可拥有执行getCmdLineOpts命令的权限，获得argv和parsed两个字段，argv包含用于调用mongod或mongos命令字符串，parsed包含所有运行时选项，不利于权限的管理。
20	检查拥有internal动作权限的角色	当定义了internal动作权限的角色，该角色用户即可拥有对应数据库进行任何操作的权限，不利于权限的管理。
21	检查拥有readWrite角色的用户	当定义了readWrite角色的用户，该角色用户即可拥有对应数据库进行读的权限加上更改数据的权限，不利于权限的管理。
22	检查拥有backup角色的用户	当定义了backup角色的用户，该角色用户即可拥有在admin数据库mms.bak文档中insert、update的权限，不利于权限的管理。
23	检查拥有clusterAdmin角色的用户	当定义了clusterAdmin角色的用户，该角色用户即可拥有最高集群管理的权限，这个角色包括了clusterManager,clusterMonitor,hostManager角色的权限，不利于权限的管理。
24	检查拥有clusterManager角色的用户	当定义了clusterManager角色的用户，在集群上管理和监视操作，该角色用户可以有权管理分别被用来共享、复制的设置本地数据库的权限，不利于权限的管理。
25	检查拥有clusterMonitor角色的用户	当定义了clusterMonitor角色的用户，该角色用户即可拥有为监视工具提供只读的权限，不利于权限的管理。
26	检查拥有dbAdmin角色的用户	当定义了dbAdmin角色的用户，该角色用户即可拥有对应数据库管理员的权限，不利于权限的管理。
27	检查拥有dbAdminAnyDatabase角色的用户	当定义了dbAdminAnyDatabase角色的用户，该角色用户即可拥有和dbAdmin角色一样的权限，除了适用于集群内所有数据库这个特性，这个角色也为整个集群提供listDatabases操作，不利于权限的管理。
28	检查拥有dbOwner角色的用户	当定义了dbOwner角色的用户，该角色用户即可拥有执行数据库所有管理操作的权限，这个角色合并了readWrite，dbAdmin，userAdmin角色的权限，不利于权限的管理。
29	检查拥有hostManager角色的用户	当定义了hostManager角色的用户，该角色用户即可拥有监视和管理服务器的权限，不利于权限的管理。

序号	检查项	描述
30	检查拥有readAnyDatabase角色的用户	当定义了readAnyDatabase角色的用户，该角色用户即可拥有读任何数据库的权限，这个角色也为整个集群提供listdatabases操作，不利于权限的管理。
31	检查拥有readWriteAnyDatabase角色的用户	当定义了readWriteAnyDatabase角色的用户，该角色用户即可拥有读写任何数据库的权限，这个角色也为整个集群提供listdatabases操作，不利于权限的管理。
32	检查拥有restore角色的用户	当定义了restore角色的用户，该角色用户即可拥有还原备份所需的权限，不利于权限的管理。
33	检查拥有root角色的用户	当定义了root角色的用户，该角色用户即可拥有所有资源的所有操作，包括readWriteAnyDatabase，dbAdminAnyDatabase，userAdminAnyDatabases，clusterAdmin，restore角色的权限，不利于权限的管理。
34	检查拥有userAdmin角色的用户	当定义了userAdmin角色的用户，该角色用户即可拥有授权任意用户的权限，包括它们自己的，不利于权限的管理。
35	检查拥有userAdminAnyDatabase角色的用户	当定义了userAdminAnyDatabase角色的用户，该角色用户即可拥有授权所有数据库任意用户的权限，包括它们自己的，不利于权限的管理。

3 连接 DDS 实例的常用方式

本节介绍了四种连接DDS实例的方式：

- mongo shell连接
- python mongo客户端
- java mongo客户端
- 基于Spring MongoTemplate操作MongoDB

mongo shell 连接

- **前提条件**

  a. 连接数据库的弹性云服务器必须和DDS实例之间网络互通，可以使用**curl**命令连接DDS实例服务端的IP和端口号，测试网络连通性。

  curl *ip:port*

  返回“It looks like you are trying to access MongoDB over HTTP on the native driver port.”，说明网络互通。

  b. 在**MongoDB官网**，下载mongo shell安装包。解压后获取其中的“mongosh”文件，并上传到弹性云服务器。

  c. 如果开启SSL，需要在界面上下载根证书，并上传到弹性云服务器。

- **连接命令**

  – SSL开启

  方式一：./mongosh *ip:port* **--authenticationDatabase admin -u** *username* **-p** *password* **--ssl --sslCAFile $***path to certificate authority file* **--sslAllowInvalidHostnames**

  方式二：./mongosh "mongodb://*<username* >:<password>@*ip:port*/test?authSource=admin" **--ssl --sslCAFile $***path to certificate authority file* **--sslAllowInvalidHostnames**

  – SSL关闭

  方式一：./mongosh *ip:port* **--authenticationDatabase admin -u** *username* **-p** *password*

  方式二：./mongosh "mongodb://*<username* >:<password>@*ip:port*/test?authSource=admin"

表 3-1 参数说明

参数	说明
ip	如果通过弹性云服务器连接，***“ip”***是主机IP，即“基本信息”页面该实例的“内网地址”。
	如果通过连接了公网的设备访问，***“ip”***为该实例已绑定的“弹性公网IP”。
port	端口，默认8635，当前端口，参考“基本信息”页面该实例的“数据库端口”。
username	当前用户名。
password	当前用户的密码。连接方式二中，需要分别将‘@’、‘%’和‘!’字符进行转义，替换为对应的十六进制的URL编码（ASCII码）“%40”、“%25”和“%21”
path to certificate authority file	SSL证书的路径。

- **注意事项**
  a. 如果开启SSL，连接命令中必须包含“--ssl”和“--sslCAFile”选项。
  b. 数据库认证“--authenticationDatabase”必须为“admin”，rwuser用户必须要在admin上认证。

更多信息，请参见《文档数据库服务快速入门》中各实例类型下“**连接实例**”的内容。

python mongo 客户端

- **前提条件**
  a. 连接数据库的弹性云服务器必须和DDS实例之间网络互通，可以使用curl命令连接DDS实例服务端的IP和端口号，测试网络连通性。

     curl *ip:port*

     返回“It looks like you are trying to access MongoDB over HTTP on the native driver port.”，说明网络互通。
  b. 在弹性云服务器上安装Python以及第三方安装包**pymongo**。推荐使用pymongo2.8版本。
  c. 如果开启SSL，需要在界面上下载根证书，并上传到弹性云服务器。
- **连接代码**
  – SSL开启

```
import ssl
import os
from pymongo import MongoClient
# 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放(密码应密文存放、使用时解密)，确保安全
# 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV
rwuser = os.getenv('EXAMPLE_USERNAME_ENV')
```



```
password = os.getenv('EXAMPLE_PASSWORD_ENV')
conn_urls="mongodb://%s:%s@ip:port/{mydb}?authSource=admin"
connection = MongoClient(conn_urls % (rwuser,
password),connectTimeoutMS=5000,ssl=True,
ssl_cert_reqs=ssl.CERT_REQUIRED,ssl_match_hostname=False,ssl_ca_certs=${path to
certificate authority file})
dbs = connection.database_names()
print "connect database success! database names is %s" % dbs
```

- SSL关闭

```
import ssl
import os
from pymongo import MongoClient
# 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放(密码应密文存放、使用时解密)，确保安全
# 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV
rwuser = os.getenv('EXAMPLE_USERNAME_ENV')
password = os.getenv('EXAMPLE_PASSWORD_ENV')
conn_urls="mongodb://%s:%s@ip:port/{mydb}?authSource=admin"
connection = MongoClient(conn_urls % (rwuser, password),connectTimeoutMS=5000)
dbs = connection.database_names()
print "connect database success! database names is %s" % dbs
```

- **注意事项**

  a. {mydb}代表要连接数据库的名称。

  b. URL中的认证数据库必须为“admin”，即“authSource=admin”。

java mongo 客户端

- **使用介绍**

  通过Java连接实例的方式有无需下载SSL证书连接和用户下载SSL证书连接两种，其中使用SSL证书连接通过了加密功能，具有更高的安全性。DDS新实例默认关闭SSL数据加密，开启SSL请参考**设置SSL数据加密**。SSL连接实现了数据加密功能，但同时也会增加网络连接响应时间和CPU消耗，不建议开启SSL数据加密。

- **前提条件**

  用户需要具备以下技能：

  - 熟悉计算机基础知识。
  - 了解Java编程语言。

- **驱动获取及使用**

  - Jar驱动下载地址：**https://repo1.maven.org/maven2/org/mongodb/mongo-java-driver/3.0.4/**
  - 使用指南请参考：**https://mongodb.github.io/mongo-java-driver/4.2/driver/getting-started/installation/**

- **使用SSL证书连接**

说明

- 该方式属于SSL连接模式，需要下载SSL证书，通过证书校验并连接数据库。
- 您可以在“实例管理”页面，单击实例名称进入“基本信息”页面，单击“数据库信息”模块“SSL”处的，下载根证书或捆绑包。
- SSL连接指南请参考：MongoDB Java Driver官方文档：**https://www.mongodb.com/zh-cn/docs/drivers/java/sync/v5.0/fundamentals/connection/tls/**。
- Java 8之前的Java运行时环境(JRE)仅在更新版本中启用TLS 1.2协议。如果您的JRE尚未启用TLS 1.2协议，请升级到更高版本以使用TLS 1.2进行连接。

通过Java连接集群实例时，代码中的Java链接格式如下：

```
mongodb://<username>:<password>@<instance_ip>:<instance_port>/<database_name>?authSource=admin&ssl=true
```

表 3-2 参数说明

参数	说明
<username>	当前用户名。
<password>	当前用户的密码。
<instance_ip>	如果通过弹性云服务器连接，“instance_ip”是主机IP，即“基本信息”页面该实例的“内网地址”。
	如果通过连接了公网的设备访问，“instance_ip”为该实例已绑定的“弹性公网IP”。
	如果需要配置多个主机地址，按照<instance_ip1>:<instance_port1>,<instance_ip2>:<instance_port2>.....列出所需主机地址即可。例如：mongodb://username:*****@127.***.***.1:8635,127.***.***.2:8635/?authSource=admin
<instance_port>	端口，默认8635，当前端口，参考“基本信息”页面该实例的“数据库端口”。
<database_name>	数据库名，即需要连接的数据库名。
authSource	鉴权用户数据库，取值为admin。
ssl	连接模式，值为true代表是使用ssl连接模式。

使用keytool工具配置CA证书，参数请参见**表3-3**：

```
keytool -importcert -trustcacerts -file <path to certificate authority file> -keystore <path to trust store> -storepass <password>
```

表 3-3 参数说明

参数	说明
<path to certificate authority file>	ssl证书的存放地址。

参数	说明
<path to trust store>	信任库的存放地址。自定义即可，例如：./trust/certs.keystore
<password>	自定义密码。

在程序中设置JVM系统属性以指向正确的信任库和密钥库：

- System.setProperty("javax.net.ssl.trustStore","<path to trust store>");
- System.setProperty("javax.net.ssl.trustStorePassword","<password>");

可参考以下示例：

```
public class Connector {
    public static void main(String[] args) {
        try {
            System.setProperty("javax.net.ssl.trustStore", "./trust/certs.keystore");
            System.setProperty("javax.net.ssl.trustStorePassword", "123456");
            ConnectionString connString = new ConnectionString("mongodb://
<username>:<password>@<instance_ip>:<instance_port>/<database_name>?
authSource=admin&ssl=true");
            MongoClientSettings settings = MongoClientSettings.builder()
                    .applyConnectionString(connString)
                    .applyToSslSettings(builder -> builder.enabled(true))
                    .applyToSslSettings(builder -> builder.invalidHostNameAllowed(true))
                    .build();
            MongoClient mongoClient = MongoClients.create(settings);
            MongoDatabase database = mongoClient.getDatabase("admin");
            //ping数据库，如果失败抛出异常
            BsonDocument command = new BsonDocument("ping", new BsonInt64(1));
            Document commandResult = database.runCommand(command);
            System.out.println("Connect to database successfully");
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Test failed");
        }
    }
}
```

- **无证书连接**

  说明

  该方式属于非SSL连接模式，不对服务端进行证书校验，用户无需下载SSL证书。

  通过Java连接集群实例时，代码中的Java链接格式如下：

  ```
  mongodb://<username>:<password>@<instance_ip>:<instance_port>/<database_name>?
  authSource=admin
  ```

  表 3-4 参数说明

参数	说明
<username>	当前用户名。
<password>	当前用户的密码。
<instance_ip>	如果通过弹性云服务器连接，“instance_ip”是主机IP，即“基本信息”页面该实例的“内网地址”。

参数	说明
	如果通过连接了公网的设备访问，“instance_ip”为该实例已绑定的“弹性公网IP”。
	如果需要配置多个主机地址，按照<instance_ip1>:<instance_port1>,<instance_ip2>:<instance_port2>.....列出所需主机地址即可。例如：mongodb://username:*****@127.***.***.1:8635,127.***.***.2:8635/?authSource=admin
<instance_port>	端口，默认8635，当前端口，参考“基本信息”页面该实例的“数据库端口”。
<database_name>	数据库名，即需要连接的数据库名。
authSource	鉴权用户数据库，取值为admin。

可参考以下示例：

```
public class Connector {
    public static void main(String[] args) {
        try {
            ConnectionString connString = new ConnectionString("mongodb://
<username>:<password>@<instance_ip>:<instance_port>/<database_name>?
authSource=admin");
            MongoClientSettings settings = MongoClientSettings.builder()
                    .applyConnectionString(connString)
                    .retryWrites(true)
                    .build();
            MongoClient mongoClient = MongoClients.create(settings);
            MongoDatabase database = mongoClient.getDatabase("admin");
            //ping数据库，如果失败抛出异常
            BsonDocument command = new BsonDocument("ping", new BsonInt64(1));
            Document commandResult = database.runCommand(command);
            System.out.println("Connect to database successfully");
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Test failed");
        }
    }
}
```

基于 Spring MongoTemplate 操作 MongoDB

- **使用介绍**

  本章节介绍如何使用Spring集成的MongoDB，使用MongoTemplate操作MongoDB。详情请参考**mongo官网**。

- **前提条件**

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
    <exclusions>
        <exclusion>
            <artifactId>spring-boot-starter-logging</artifactId>
            <groupId>org.springframework.boot</groupId>
```



```
        </exclusion>
    </exclusions>
</dependency>
```

- **配置指导**

```
spring:
  data:
    mongodb:          #MongoDB配置，作为参考
      // 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放(密码应密文存放、使用时解密)，确保安全；
      // 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
      String userName = System.getenv("EXAMPLE_USERNAME_ENV");
      String rwuserPassword = System.getenv("EXAMPLE_PASSWORD_ENV");
      uri: mongodb://" + userName + ":" + rwuserPassword + "@192.***.***.***:8635,192.***.***.***:8635/${mongodb.database}
      database: ${mongodb.database}
```

- **开发指导**

```
/**
 * mongoDB执行
 */
@Autowired
private MongoTemplate template;

/**
 * 日志配置
 */
@Autowired
private LoggingProperties properties;

@Override
public void write(BaseLog businessLog, LoggingOption option) {
    if (template != null) {
        LoggingConfig config = properties.getBusinessConfig(businessLog.getCategory());
        String collection = config.getMeta().get("collection");
        if (StringUtils.isNotEmpty(collection)) {
            Object data = mapping(businessLog, config);
            template.save(data, collection);
            if (log.isDebugEnabled()) {
                log.debug("save audit log to mongodb successfully!, message: {}",

StringEscapeUtils.escapeJava(TransformUtil.toJsonByJackson(businessLog)));
            }
        } else {
            log.warn("mongo log write log failed, mongoconfig is null");
        }
    } else {
        log.warn("mongo log write log failed, mongoTemplate is null");
    }
}
```

- **注意事项**

a. SSL方式连接，需要手动生成trustStore文件。

b. 认证数据库必须为“admin”，之后再切换至业务数据库。

4 其他云 MongoDB 迁移到 DDS

数据复制服务（Data Replication Service，简称DRS）支持将其他云MongoDB数据库的数据迁移到本云文档数据库服务（Document Database Service，以下简称DDS）实例。通过DRS提供的实时迁移任务，实现在数据库迁移过程中业务和数据库不停机，业务中断时间最小化。

本章节主要介绍了通过DRS将其他云数据库实时迁移至本云DDS的任务配置流程。包括以下迁移场景：

- 其他云MongoDB数据库实时迁移至本云DDS。
- 其他云内云主机自建自维护的MongoDB数据库迁移至本云DDS。

资源规划

表 4-1 资源规划

类别	子类	规划	备注
VPC	VPC名称	vpc-dds	自定义，易理解可识别。
	所属Region	华南-广州	选择和自己业务区最近的Region，减少网络时延。
	可用区	可用区一	-
	子网网段	10.0.0.0/24	子网选择时建议预留足够的网络资源。
	子网名称	subnet-default	自定义，易理解可识别。
其他云MongoDB	数据库版本	MongoDB 4.4	-
	IP地址	192.168.0.1	仅作为示例。
	端口	8635	仅作为示例。
DDS实例	实例名称	dds-test	自定义，易理解可识别。
	数据库版本	DDS 4.4	-
	可用区类型	单可用区	仅作为示例。

类别	子类	规划	备注
	可用区	可用区一	本示例中为可用区一。
	性能规格	增强 II 型 4 vCPUs | 16 GB	本示例中选择的规格。实际选择的规格需要结合业务场景选择。
DRS迁移任务	迁移任务名	DRS-test-migrate	自定义。
	源数据库引擎	MongoDB	-
	目标数据库引擎	DDS	-
	网络类型	公网网络	本示例中采用公网网络。

网络示意图

图 4-1 其他云 MongoDB 数据库实时迁移示意图

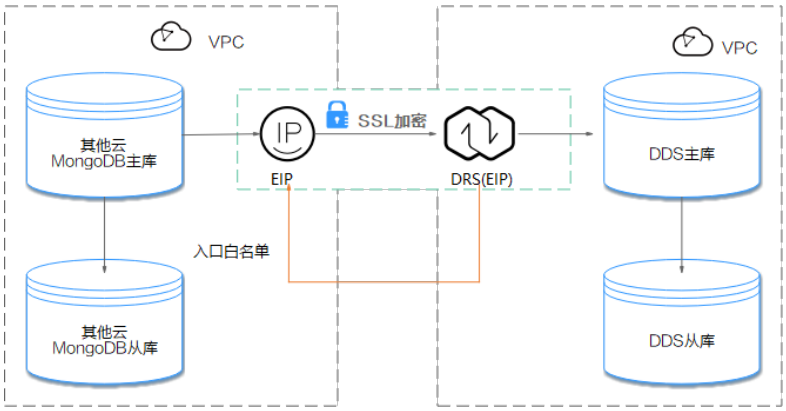
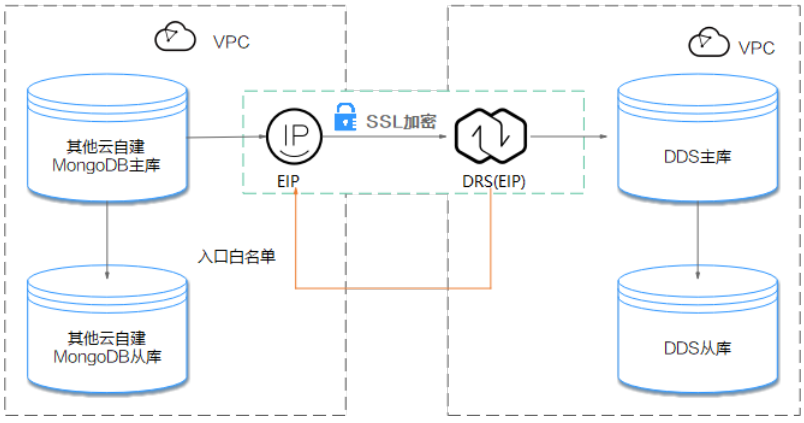
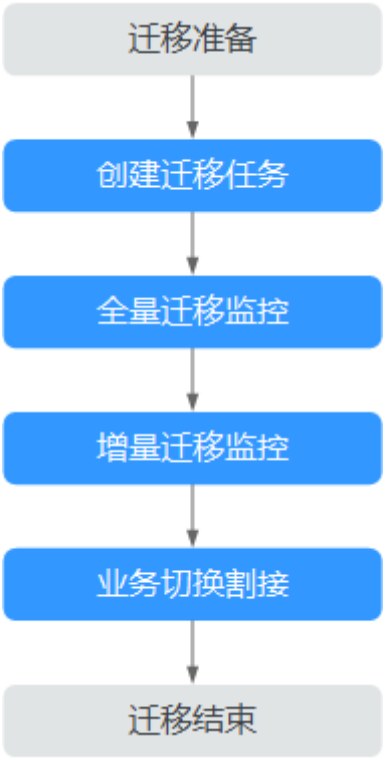


图 4-2 其他云自建 MongoDB 数据库实时迁移示意图



迁移流程

图 4-3 迁移流程图



迁移建议（重要）

- 数据库迁移与环境多样性和人为操作均有密切关系，为了确保迁移的平顺，建议您在进行正式的数据库迁移之前进行一次演练，可以帮助您提前发现问题并解决问题，如何最小化对数据库的影响请参考如下建议。
- 强烈建议您在启动任务时选择“稍后启动”功能，将启动时间设置在业务低峰期，相对静止的数据可以有效提升一次性迁移成功率，避免迁移对业务造成性能影响。

迁移须知（重要）

须知

在创建迁移任务之前，请您务必仔细阅读迁移须知。

参考《数据复制服务实时迁移》中具体链路的“**使用须知**”。

迁移准备

1. **权限准备：**

   当使用 DRS 将其他云MongoDB数据库的数据迁移到本云DDS实例时，在不同迁移类型情况下，对源数据库和目标数据库的账号权限要求如**表4-2**：

表 4-2 迁移账号权限

迁移类型	全量迁移	全量+增量迁移
源数据库	• 副本集：连接源数据库的用户需要对待迁移库有read权限。 • 单节点：连接源数据库的用户需要对待迁移库有read权限。 • 集群：连接源数据库的用户需要对待迁移库有read权限，对config数据库有read权限。 • 如果需要迁移源数据库用户和角色信息，连接源数据库的用户需要对admin数据库的系统表system.users，system.roles有读权限。	• 副本集：连接源数据库的用户需要对待迁移库有read权限，对local数据库有read权限。 • 单节点：连接源数据库的用户需要对待迁移库有read权限，对local数据库有read权限。 • 集群：连接源数据库mongos节点的用户需要对待迁移库有read权限，对config数据库有read权限，连接源数据库分片节点的用户需要对admin数据库有readAnyDatabase权限，对local数据库有read权限。 • 如果需要迁移源数据库用户和角色信息，连接源数据库的用户需要对admin数据库的系统表system.users，system.roles有读权限。
目标数据库	连接目标数据库的用户需要对admin数据库有dbAdminAnyDatabase权限，对目标数据库有readWrite权限。对于目标数据库是集群的实例，迁移账号还要有对config数据的读权限	

- **源数据库权限设置：**

  需要确保源数据库MongoDB的账号权限具备表4-2的要求。若权限不足，需要在源数据库端开通高权限的账号。

- **目标数据库权限设置：**

  本云DDS实例使用初始账号即可。

2. **网络准备：**

   源数据库需要开放公网访问。

   - **源数据库的网络设置：**

     源数据库MongoDB实例需要开放公网域名的访问。

- **目标数据库的网络设置：**目标端不需要进行设置。

3. **安全规则准备：**
   - **源数据库安全组规则设置：**

     源数据库MongoDB实例需要将具体的DRS迁移实例的弹性公网IP添加到其网络白名单中，确保源数据库MongoDB实例可以与上述弹性公网IP连通。

     在设置网络白名单之前，需要先获取DRS迁移实例的弹性公网IP，具体操作如下：

     - 迁移实例创建成功后，可在“源库及目标库”页面获取DRS迁移实例的弹性公网IP。如图4-4所示：

       图 4-4 迁移实例公网弹性 IP

     以上讲述的是精细配置白名单的方法，还有一种简单设置白名单的方法，**在安全允许的情况下**，可以将源数据库MongoDB实例的网络白名单设置为0.0.0.0/0，代表允许任何IP地址访问该实例。

     上述的网络白名单是为了进行数据迁移设置的，迁移结束后可以删除。
   - **目标数据库安全组规则设置：**

     目标数据库默认与DRS迁移实例处在同一个VPC，网络是互通的，DRS可以直接写入数据到目标数据库，不需要进行任何设置。
4. **其他事项准备：**

   由于迁移过程不会迁移MongoDB数据库的用户信息以及相关参数，需要自行将上述信息导出后手动添加到目标DDS中。

迁移步骤

步骤1 创建迁移任务。

1. 登录管理控制台，在服务列表中选择“数据库 > 数据复制服务”，进入数据复制服务信息页面。
2. 在“实时迁移管理”页面，单击右上角“创建迁移任务”，进入迁移任务信息页面。
3. 在“迁移实例”页面，填选任务名称、描述和迁移实例信息。

图 4-5 迁移实例信息

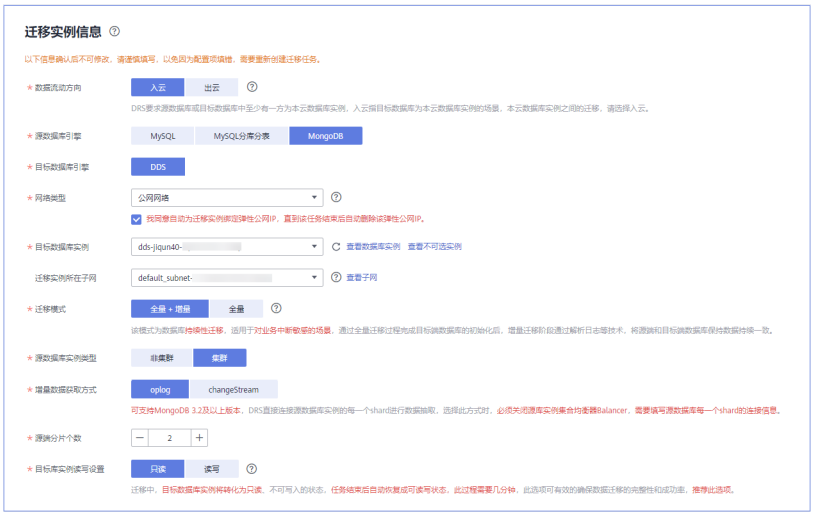


表 4-3 任务和描述

参数	描述
区域	当前所在区域，可进行切换。为了降低访问时延、提高访问速度，请就近选择靠近您业务的区域。
项目	当前区域对应的项目，可进行切换。
任务名称	任务名称在4-50位之间，必须以字母开头，不区分大小写，可以包含字母、数字、中划线或下划线，不能包含其他的特殊字符。
描述	描述不能超过256位，且不能包含! = < > & ' " \ 特殊字符。

表 4-4 迁移实例信息

参数	描述
数据流动方向	选择“入云”。
源数据库引擎	选择“MongoDB”。
目标数据库引擎	选择“DDS”。
网络类型	选择“公网网络”。
目标数据库实例	选择您所创建的本云DDS实例。
迁移实例所在子网	选择迁移实例所在的子网。也可以单击“查看子网”，跳转至“网络控制台”查看实例所在子网帮助选择。 默认值为当前所选数据库实例所在子网，请选择有可用IP地址的子网。为确保迁移实例创建成功，仅显示已经开启DHCP的子网。

参数	描述
迁移模式	– 全量 全量迁移为数据库一次性迁移，适用于可中断业务的数据库迁移场景，全量迁移将非系统数据库的全部数据库对象和数据一次性迁移至目标端数据库，包括：集合、索引等。 – 全量+增量 全量+增量迁移为数据库持续性迁移，适用于对业务中断敏感的场景，通过全量迁移过程完成目标端数据库的初始化后，增量迁移阶段通过解析日志等技术，将源端和目标端数据库保持数据持续一致。
源数据库实例类型	迁移模式为“全量+增量”时，需要根据源数据库的具体来源进行设置。 – 当源库类型属于集群时，该项需要设置为集群。 – 当源库类型属于副本集或者单节点时，该项需要设置为非集群。
增量数据获取方式	当源端实例类型设置为“集群”时，增量同步时需要选择数据获取方式。 – oplog：支持MongoDB 3.2及以上版本，DRS直接连接源数据库实例的每一个Shard进行数据抽取。选择此方式时，必须关闭源库实例集合均衡器Balancer，测试连接时需要填写源数据库每一个Shard的连接信息。 – changeStream：支持MongoDB 4.0及以上版本，DRS连接源数据库实例的mongos进行数据抽取，选择此方式时，源数据库实例必须开启WiredTiger存储引擎，推荐此选项。 **说明** “changeStream”方式目前仅支持白名单用户，需要提交工单申请才能使用。您可以在管理控制台右上角，选择“工单 > 新建工单”，完成工单提交。
源端分片个数	当源端实例类型设置为“集群”且增量数据获取方式为“oplog”时，需要填写源端数据库分片个数。 源端数据库分片个数默认最小值为2，最大值为32，你需要根据源库实际的集群分片个数设置该值大小。

4. 在“源库及目标库”页面，迁移实例创建成功后，填选源库信息和目标库信息，单击“源库和目标库”处的“测试连接”，分别测试并确定与源库和目标库连通后，勾选协议，单击“下一步”。

图 4-6 源库信息页面

表 4-5 源库信息

参数	描述
mongosIP地址或域名	源数据库的IP地址或域名，格式为IP地址/域名:端口。其中源数据库服务端口，可输入范围为1~65534间的整数。 该输入框最多支持填写3组源数据库的IP地址或者域名信息，多个值需要使用英文逗号隔开。例如：192.168.0.1:8080,192.168.0.2:8080。同时需要确保所填写的多个IP地址或域名属于同一个分片集群。 **说明** 此处若填写的是多组IP地址或者域名信息，在进行测试连接的过程中，只要存在一组IP地址或者域名可以连通，那么测试连接就提示成功。所以需要您保证填写的IP地址或域名的正确性。
账号认证数据库	填写的数据库账号所属的数据库名称。例如：华为云DDS实例默认的账号认证数据库为admin。
mongos用户名	访问源数据库MongoDB的用户名。
mongos密码	访问源数据库MongoDB的用户名所对应的密码。
SSL安全连接	通过该功能，用户可以选择是否开启对迁移链路的加密。如果开启该功能，需要用户上传SSL CA根证书。
分片数据库	根据源库实际的集群分片个数，填写对应的分片数据库信息。

- 目标库信息配置

图 4-7 目标库信息

表 4-6 目标库信息

参数	描述
数据库实例名称	默认为创建迁移任务时选择的数据库实例，不可进行修改。
数据库用户名	目标数据库对应的数据库用户名。
数据库密码	目标数据库的登录密码。

5. 在“迁移设置”页面，设置迁移对象，单击“下一步”。

图 4-8 设置迁移对象

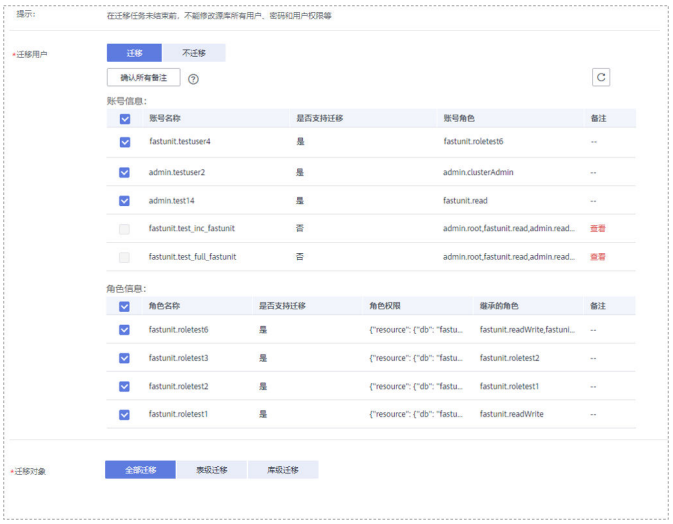
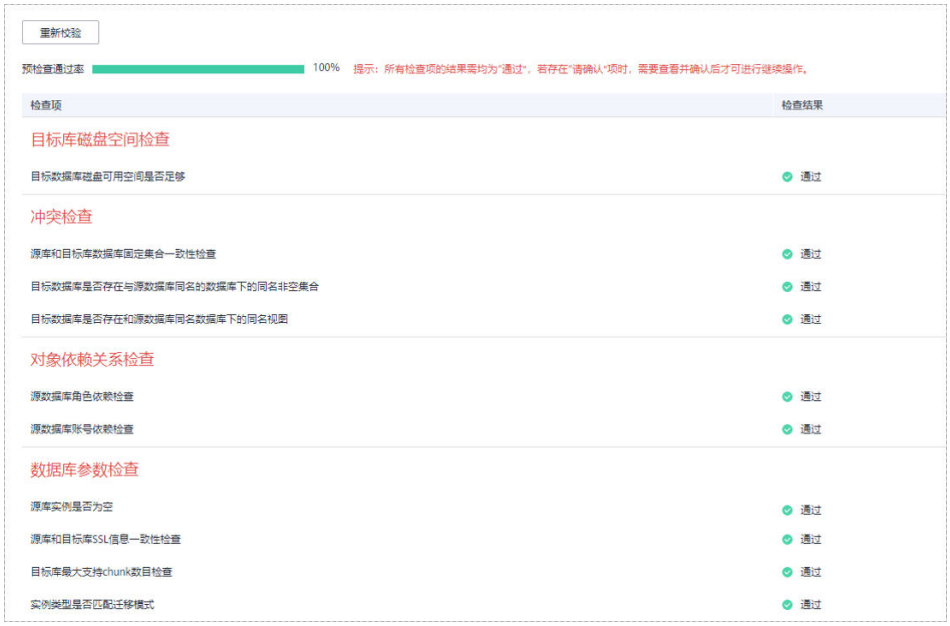


表 4-7 迁移对象

参数	描述
迁移用户	常见的迁移用户一般分为两类：支持迁移的用户和不支持迁移的用户。您可以根据业务需求选择“迁移”或者“不迁移”，其中，不支持迁移的账号或者未选择迁移的账号将在目标数据库中缺失，需要先确保业务不受影响。 – 迁移 当您选择迁移用户时，请参见《数据复制服务用户指南》中“**迁移用户**”章节进行数据库用户及角色的处理。 – 不迁移 迁移过程中，将不进行数据库用户及角色的迁移。
迁移对象	您可以根据业务需求，选择全部对象迁移、表级迁移或者库级迁移。 – 全部迁移：将源数据库中的所有对象全部迁移至目标数据库，对象迁移到目标数据库实例后，对象名将会保持与源数据库实例对象名一致且无法修改。 – 表级迁移：将选择的表级对象迁移至目标数据库。 – 库级迁移：将选择的库级对象迁移至目标数据库。 如果有切换源数据库的操作或源库迁移对象变化的情况，请务必在选择迁移对象前单击右上角的，以确保待选择的对象为最新源数据库对象。 **说明** – 若选择部分数据库进行迁移时，由于存储过程、视图等对象可能与其他数据库的表存在依赖关系，若所依赖的表未迁移，则会导致迁移失败。建议您在迁移之前进行确认，或选择全部数据库进行迁移。 – 选择对象的时候，对象名称的前后空格不显示，中间如有多个空格只显示一个空格。 – 选择对象的时候支持搜索，以便您快速选择需要的数据库对象。

6. 在“预检查”页面，进行迁移任务预校验，校验是否可进行任务迁移。
   – 查看检查结果，如有不通过的检查项，需要修复不通过项后，单击“重新校验”按钮重新进行迁移任务预校验。
   预检查不通过项处理建议请参见《数据复制服务用户指南》中的“**预检查不通过项修复方法**”。
   – 预检查完成后，且所有检查项结果均通过时，单击“下一步”。

图 4-9 预检查



说明

所有检查项结果均通过时，若存在待确认项，需要阅读并确认详情后才可以继续执行下一步操作。

7. 在“任务确认”页面，设置迁移任务的启动时间、任务异常通知设置、SMN主题、时延阈值、任务异常自动结束时间，并确认迁移任务信息无误后，单击“启动任务”，提交迁移任务。

图 4-10 任务启动设置

表 4-8 任务启动设置

参数	描述
启动时间	迁移任务的启动时间可以根据业务需求，设置为“立即启动”或“稍后启动”，优选“稍后启动”。 **说明** 预计迁移任务启动后，会对源数据库和目标数据库的性能产生影响，建议您将任务启动时间设定在业务低峰期，同时预留2-3天校对数据。

参数	描述
任务异常通知设置	该项为可选参数，开启之后，选择对应的SMN主题。当迁移任务状态异常时，系统将发送通知。
SMN主题	“任务异常通知设置”项开启后可见，需提前在SMN上申请主题并添加订阅。 SMN主题申请和订阅可参考**《消息通知服务用户指南》**。
时延阈值	在增量迁移阶段，源数据库和目标数据库之间的实时同步有时会存在一个时间差，称为时延，单位为秒。 时延阈值设置是指时延超过一定的值后（时延阈值范围为0—3600s），DRS可以发送告警通知。告警通知将在时延稳定超过设定的阈值6min后发送，避免出现由于时延波动反复发送告警通知的情况。 **说明** – 首次进入增量迁移阶段，会有较多数据等待同步，存在较大的时延，属于正常情况，不在此功能的监控范围之内。 – 设置时延阈值之前，需要设置任务异常通知。 – 当时延阈值设置为0时，不会发送通知给收件人。
任务异常自动结束时间（天）	设置任务异常自动结束天数，输入值必须在14-100之间，默认值14。 **说明** 异常状态下的任务仍然会计费，而长时间异常的任务无法续传和恢复。设置任务异常自动结束天数后，异常且超时的任务将会自动结束，以免产生不必要的费用。

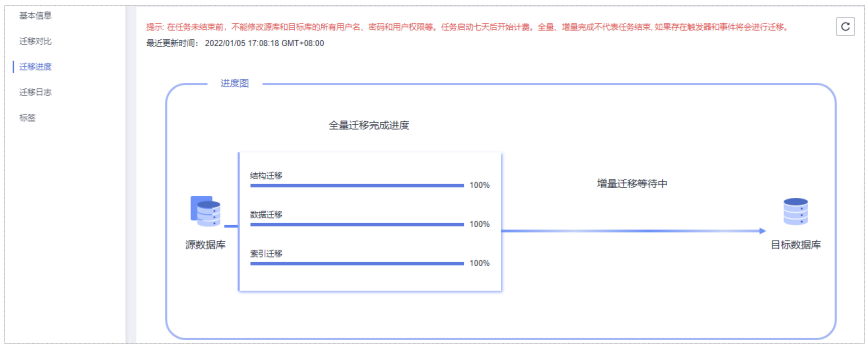
8. 迁移任务提交后，开始启动迁移任务，您可以返回“实时迁移管理”页面，查看迁移任务状态。

步骤2 任务管理。

迁移任务启动后，会经历全量迁移和增量迁移两个阶段，对于不同阶段的迁移任务，您可以进行任务管理。

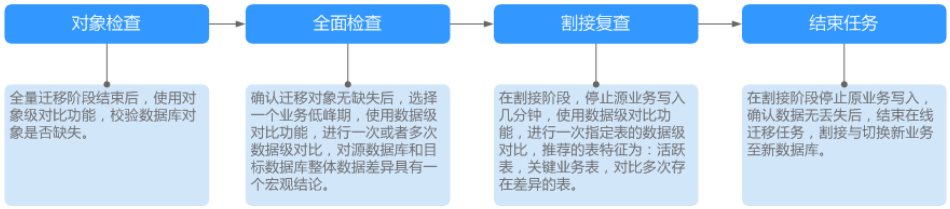
- 全量迁移
  - 查看迁移进度：全量迁移中的任务，您可单击任务名称，在“迁移进度”页签下，查看全量迁移进度，您可以查看结构、数据、索引迁移的进度，也查看具体迁移对象的迁移进度。当全量迁移进度显示为100%，表示全量迁移已经完成。
  - 查看迁移明细：迁移明细中，您可以查看具体迁移对象的迁移进度，当“对象数目”和“已迁移对象”相等时，表示该对象已经迁移完成，可通过“查看详情”查看每个对象的迁移进度。仅白名单用户该支持功能，您可以通过提交工单的方式进行申请使用。
- 增量迁移
  - 查看时延监控：全量迁移完成后，开始进行增量迁移。对于增量迁移中的任务，您可单击任务名称，在“迁移进度”页签下，查看增量迁移同步时延，**当时延为0s时，说明源数据库和目标数据库的数据是实时同步的。**您也可以使用“迁移对比”页签查看一致性。

图 4-11 查看时延监控



- 查看迁移对比：为了尽可能减少业务的影响和业务中断时间，增量迁移中的任务，您可单击任务名称，在“迁移对比”页签下，建议按照如下流程进行迁移对比，以便确定合适的业务割接时机。

图 4-12 迁移对比流程



具体的迁移对比操作及注意事项请参考《数据复制服务用户指南》中“**对比迁移项**”章节。

步骤3 割接建议。

建议您选择一个业务低峰期，开始正式系统割接流程。割接前，请您确认至少在业务低峰期有过一次完整的数据对比。可以结合数据对比的“稍后启动”功能，选择业务低峰期进行数据对比，以便得到更为具有参考性的对比结果。由于同步具有轻微的时差，在数据持续操作过程中进行对比任务，可能会出现少量数据不一致对比结果，从而失去参考意义。

1. 先中断业务（如果业务负载非常轻，也可以尝试不中断业务）。
2. 在源数据库端执行如下语句，并观察在1-5分钟内若无任何新会话执行SQL，则可认为业务已经完全停止。

```
db.currentOp()
```

说明

上述语句查询到的进程列表中，包括DRS迁移实例的连接，您需要确认除DRS迁移实例的连接外无任何新会话执行SQL，即可认为业务已经完全停止。

3. 通过DRS迁移任务监控页面进行观察同步时延，保持实时同步时延为0，并稳定保持一段时间；同时，您可以使用数据级对比功能，进行割接前的最后一次数据级对比，耗时可参考之前的对比记录。
   - 如果时间允许，则选择全部对比。
   - 如果时间不允许，则推荐对比活跃表，关键业务表，第二步对比多次存在差异的表等。
4. 确定系统割接时机，业务系统指向本云数据库，业务对外恢复使用，迁移完成。

步骤4 迁移结束。

1. 结束迁移任务：业务系统和数据库切换至本云后，为了防止源数据库的操作继续同步到目标数据库，造成数据覆盖问题，此时您可选择结束迁移任务，该操作仅删除了迁移实例，迁移任务仍显示在任务列表中，您可以进行查看或删除。结束迁移任务后，DRS将不再计费。
2. 删除迁移任务：对于已结束的迁移任务，您可选择删除任务。该操作将一并删除迁移任务，删除迁移任务后，该任务将不会出现在任务列表中。

----结束

5 本地 MongoDB 迁移到 DDS

数据复制服务（Data Replication Service，简称DRS）支持将本地MongoDB数据库的数据迁移至本云文档数据库服务（Document Database Service，以下简称DDS）实例。通过DRS提供的实时迁移任务，实现在数据库迁移过程中业务和数据库不停机，业务中断时间最小化。

本章节主要介绍了通过DRS将本地MongoDB数据库实时迁移至本云DDS的任务配置流程。支持以下网络类型：

- VPN（Virtual Private Network，虚拟专用网络）网络
- 公网网络

资源规划

表 5-1 资源规划

类别	子类	规划	备注
VPC	VPC名称	vpc-dds	自定义，易理解可识别。
	所属Region	华南-广州	选择和自己业务区最近的Region，减少网络时延。
	可用区	可用区一	-
	子网网段	10.0.0.0/24	子网选择时建议预留足够的网络资源。
	子网名称	subnet-default	自定义，易理解可识别。
本地MongoDB	数据库版本	MongoDB 4.4	自定义，易理解可识别。
DDS	DDS 实例名	dds-test	自定义，易理解可识别。
	数据库引擎	DDS	-
	兼容的数据库版本	4.4	-
	可用区类型	单可用区	-

类别	子类	规划	备注
	可用区	可用区一	-
	性能规格	增强 II 型	-
	CPU架构	X86 8 vCPUs | 32GB	-
DRS迁移任务	迁移任务名	DRS-dds	自定义。
	源数据库引擎	MongoDB	本示例中源数据库为自建MongoDB，即在本地服务器上安装社区版MongoDB。
	目标数据库引擎	DDS	本示例中目标数据库为华为云DDS实例。
	网络类型	公用网络	本示例中采用公用网络。

网络示意图

图 5-1 VPN 网络

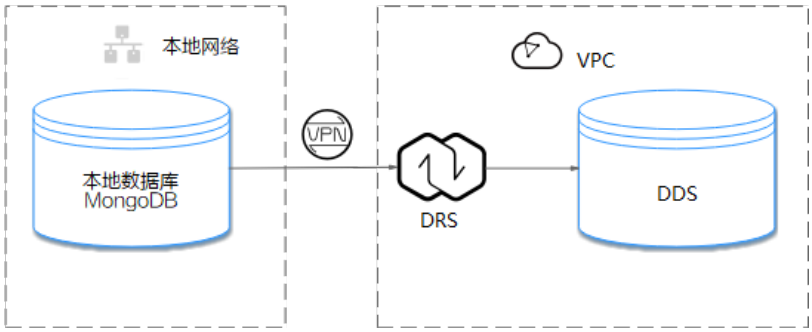
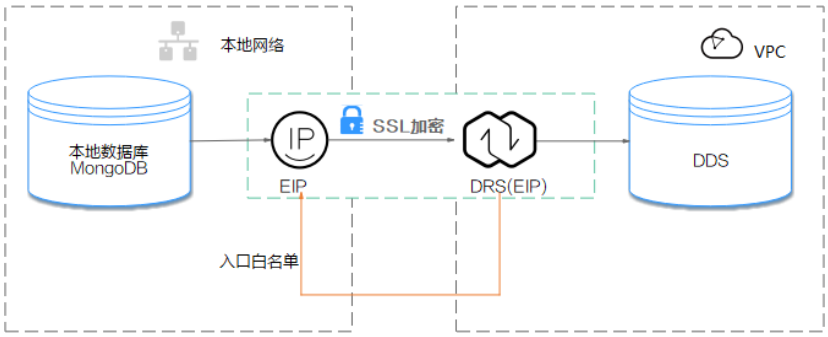
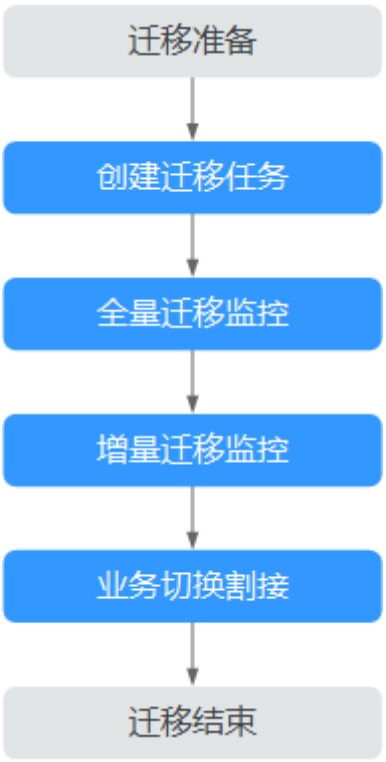


图 5-2 公网网络+SSL 安全连接



迁移流程

图 5-3 迁移流程图



迁移建议（重要）

- 数据库迁移与环境多样性和人为操作均有密切关系，为了确保迁移的平顺，建议您在进行正式的数据库迁移之前进行一次演练，可以帮助您提前发现问题并解决问题，如何最小化对数据库的影响请参考如下建议。
- 强烈建议您在启动任务时选择“稍后启动”功能，将启动时间设置在业务低峰期，相对静止的数据可以有效提升一次性迁移成功率，避免迁移对业务造成性能影响。

迁移须知（重要）

须知

在创建迁移任务之前，请您务必仔细阅读迁移须知。

参考《数据复制服务实时迁移》中具体链路的“**使用须知**”。

迁移准备

1. **权限准备：**

   当使用 DRS 将本地数据库的数据迁移到本云DDS实例时，在不同迁移类型情况下，对源数据库和目标数据库的账号权限要求如**表5-2**所示：

表 5-2 迁移账号权限

迁移类型	全量迁移	全量+增量迁移
源数据库	• 副本集：连接源数据库的用户需要对待迁移库有read权限。 • 单节点：连接源数据库的用户需要对待迁移库有read权限。 • 集群：连接源数据库的用户需要对待迁移库有read权限，对config数据库有read权限。 • 如果需要迁移源数据库用户和角色信息，连接源数据库的用户需要对admin数据库的系统表system.users，system.roles有读权限。	• 副本集：连接源数据库的用户需要对待迁移库有read权限，对local数据库有read权限。 • 单节点：连接源数据库的用户需要对待迁移库有read权限，对local数据库有read权限。 • 集群：连接源数据库mongos节点的用户需要对待迁移库有read权限，对config数据库有read权限，连接源数据库分片节点的用户需要对admin数据库有readAnyDatabase权限，对local数据库有read权限。 • 如果需要迁移源数据库用户和角色信息，连接源数据库的用户需要对admin数据库的系统表system.users，system.roles有读权限。
目标数据库	连接目标数据库的用户需要对admin数据库有dbAdminAnyDatabase权限，对目标数据库有readWrite权限。对于目标数据库是集群的实例，迁移账号还要有对config数据的读权限	

- **源数据库的权限设置：**

  需要确保源数据库MongoDB的账号具备表5-2的权限，若权限不足，需要在源数据库端创建高权限的账号。

- **目标数据库的权限设置：**

  本云DDS实例使用初始账号即可。

2. **网络准备：**
   - **源数据库的网络设置：**

     本地MongoDB数据库实时迁移至本云DDS的场景，一般可以使用VPN网络和公网网络两种方式进行迁移，您可以根据实际情况为本地MongoDB数据库开

放公网访问或建立VPN访问。一般推荐使用公网网络进行迁移，该方式下的数据迁移过程较为方便和经济。

- **目标数据库的网络设置：**
  - 若通过VPN访问，请先开通VPN服务，确保源数据库和目标DDS网络互通。
  - 若通过公网网络访问，目标DDS不需要进行设置。

3. **安全规则准备：**

   a. **源数据库的白名单设置：**

   - 若通过公网网络进行迁移，源数据库MongoDB实例需要将具体的DRS迁移实例的弹性公网IP添加到其网络白名单中，确保源数据库MongoDB实例可以与上述弹性公网IP连通。在设置网络白名单之前需要获取DRS迁移实例，具体方法如下：

     迁移实例创建成功后，可在“源库及目标库”页面获取DRS迁移实例的弹性公网IP。如图5-4所示：

     图 5-4 迁移实例公网弹性 IP

     以上讲述的是精细配置白名单的方法，还有一种简单设置白名单的方法，**在安全允许的情况下**，可以将源数据库MongoDB实例的网络白名单设置为0.0.0.0/0，代表允许任何IP地址访问该实例。

   - 若通过VPN网络进行迁移，源库需要将DRS迁移实例的私有IP添加到其网络白名单内，确保源端和目标端网络互通。

   上述的网络白名单是为了进行数据迁移设置的，迁移结束后可以删除。

   b. **目标数据库安全组规则设置：**

   目标数据库默认与DRS迁移实例处在同一个VPC，网络是互通的，DRS可以直接写入数据到目标数据库，不需要进行任何设置。

4. **其他事项准备：**

   由于迁移过程不会迁移MongoDB数据库的用户信息以及相关参数，需要自行将上述信息导出后手动添加到目标DDS中。

迁移步骤

以下操作以公网网络迁移的方式为例，指导您通过DRS将本地MongoDB数据库实时迁移至本云DDS实例。

步骤1 创建迁移任务。

1. 登录管理控制台，在服务列表中选择“数据库 > 数据复制服务”，进入数据复制服务信息页面。
2. 在“实时迁移管理”页面，单击右上角“创建迁移任务”，进入迁移任务信息页面。
3. 在“迁移实例”页面，填选任务名称、通知收件人、描述和迁移实例信息。

图 5-5 迁移实例信息

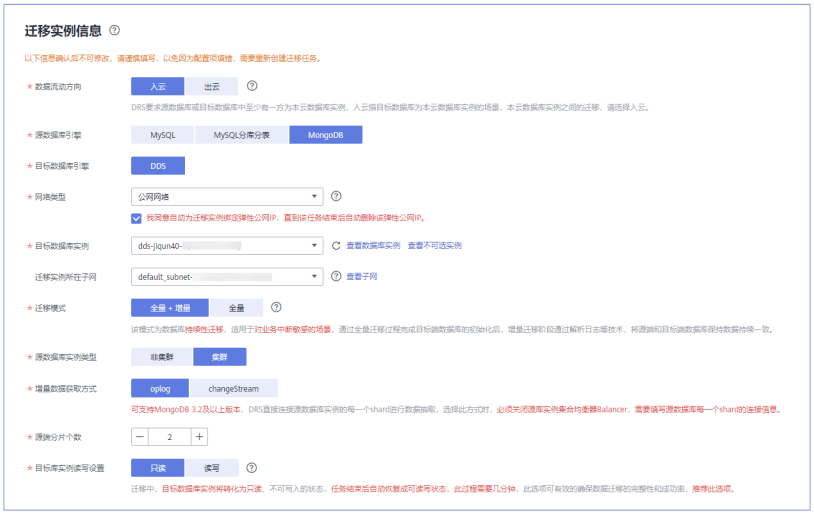


表 5-3 任务和描述

参数	描述
区域	当前所在区域，可进行切换。为了降低访问时延、提高访问速度，请就近选择靠近您业务的区域。
项目	当前区域对应的项目，可进行切换。
任务名称	任务名称在4-50位之间，必须以字母开头，不区分大小写，可以包含字母、数字、中划线或下划线，不能包含其他的特殊字符。
描述	描述不能超过256位，且不能包含! = < > & ' " \ 特殊字符。

表 5-4 迁移实例信息

参数	描述
数据流动方向	选择“入云”。
源数据库引擎	选择“MongoDB”。
目标数据库引擎	选择“DDS”。
网络类型	选择“公网网络”。建议您开启SSL安全连接，SSL约降低20-30%的迁移性能，但保证了数据的安全性。
目标数据库实例	选择您所创建的本云DDS实例。
迁移模式	- 全量 全量为一次性迁移，如果您只进行全量迁移时，建议停止对源数据库的操作，否则迁移过程中源数据库产生的新数据不会同步到目标数据库。 - 全量+增量 增量可以在全量迁移完成的基础上实现数据的持续同步。

参数	描述
源数据库实例类型	迁移模式为“全量+增量”时，需要根据源数据库的具体来源进行设置。 – 当源库类型属于集群时，该项需要设置为集群。 – 当源库类型属于副本集或者单节点时，该项需要设置为非集群。
增量数据获取方式	当源端实例类型设置为“集群”时，增量同步时需要选择数据获取方式。 – oplog：支持MongoDB 3.2及以上版本，DRS直接连接源数据库实例的每一个Shard进行数据抽取。选择此方式时，必须关闭源库实例集合均衡器Balancer，测试连接时需要填写源数据库每一个Shard的连接信息。 – changeStream：支持MongoDB 4.0及以上版本，DRS连接源数据库实例的mongos进行数据抽取，选择此方式时，源数据库实例必须开启WiredTiger存储引擎，推荐此选项。 **说明** “changeStream”方式目前仅支持白名单用户，需要提交工单申请才能使用。您可以在管理控制台右上角，选择“工单 > 新建工单”，完成工单提交。
源端分片个数	当源端实例类型设置为“集群”且增量数据获取方式为“oplog”时，需要填写源端数据库分片个数。 源端数据库分片个数默认最小值为2，最大值为32，你需要根据源库实际的集群分片个数设置该值大小。

4. 在“源库及目标库”页面，迁移实例创建成功后，填选源库信息和目标库信息，单击“源库和目标库”处的“测试连接”，分别测试并确定与源库和目标库连通后，勾选协议，单击“下一步”。

图 5-6 源库信息页面

表 5-5 源库信息

参数	描述
mongosIP地址或域名	源数据库的IP地址或域名，格式为IP地址/域名:端口。其中源数据库服务端口，可输入范围为1~65534间的整数。 该输入框最多支持填写3组源数据库的IP地址或者域名信息，多个值需要使用英文逗号隔开。例如：192.168.0.1:8080,192.168.0.2:8080。同时需要确保所填写的多个IP地址或域名属于同一个分片集群。 **说明** 此处若填写的是多组IP地址或者域名信息，在进行测试连接的过程中，只要存在一组IP地址或者域名可以连通，那么测试连接就提示成功。所以需要您保证填写的IP地址或域名的正确性。
账号认证数据库	填写的数据库账号所属的数据库名称。例如：华为云DDS实例默认的账号认证数据库为admin。
mongos用户名	访问源数据库MongoDB的用户名。
mongos密码	访问源数据库MongoDB的用户名所对应的密码。
SSL安全连接	通过该功能，用户可以选择是否开启对迁移链路的加密。如果开启该功能，需要用户上传SSL CA根证书。
分片数据库	根据源库实际的集群分片个数，填写对应的分片数据库信息。

- 目标库信息配置

图 5-7 目标库信息

表 5-6 目标库信息

参数	描述
数据库实例名称	默认为创建迁移任务时选择的数据库实例，不可进行修改。
数据库用户名	目标数据库对应的数据库用户名。

参数	描述
数据库密码	目标数据库的登录密码。

5. 在“迁移设置”页面，设置迁移对象，单击“下一步”。

图 5-8 设置迁移对象

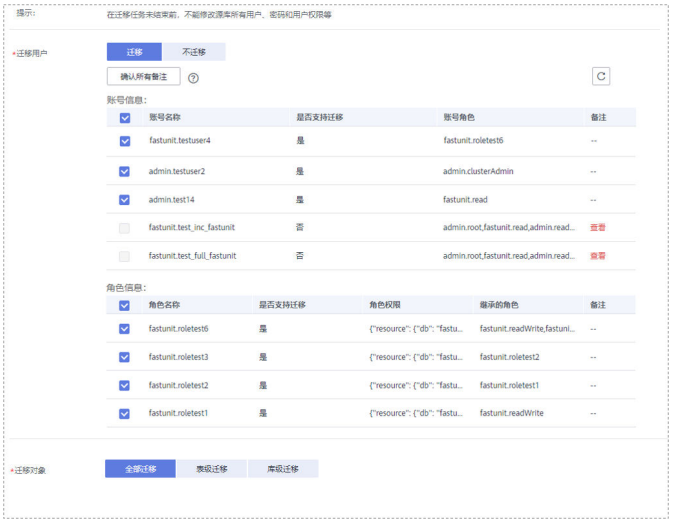


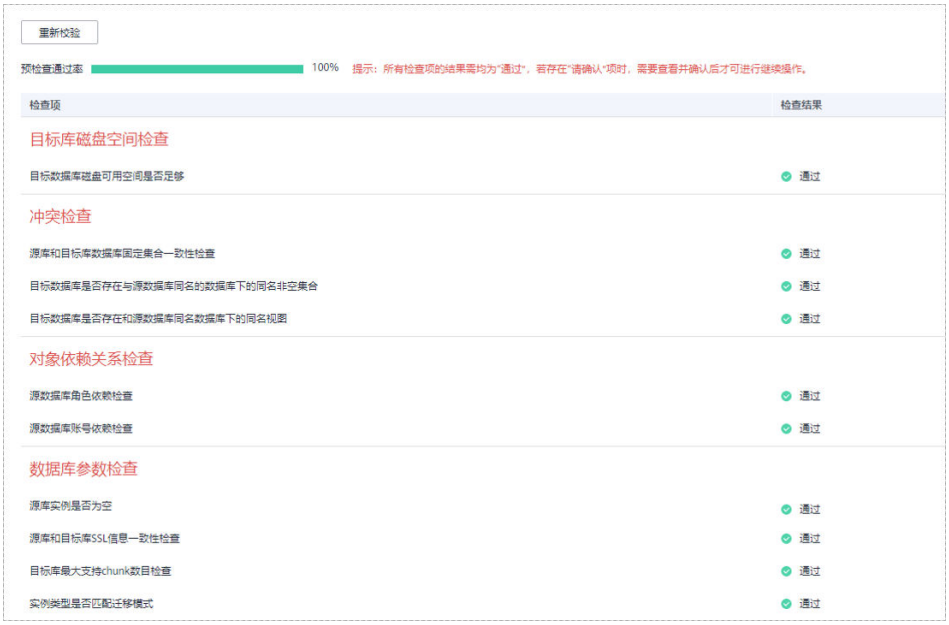
表 5-7 迁移对象

参数	描述
迁移用户	常见的迁移用户一般分为两类：支持迁移的用户和不支持迁移的用户。您可以根据业务需求选择“迁移”或者“不迁移”，其中，不支持迁移的账号或者未选择迁移的账号将在目标数据库中缺失，需要先确保业务不受影响。 – 迁移 当您选择迁移用户时，请参见《数据复制服务用户指南》中“**迁移用户**”章节进行数据库用户及角色的处理。 – 不迁移 迁移过程中，将不进行数据库用户及角色的迁移。

参数	描述
迁移对象	您可以根据业务需求，选择全部对象迁移、表级迁移或者库级迁移。 – 全部迁移：将源数据库中的所有对象全部迁移至目标数据库，对象迁移到目标数据库实例后，对象名将会保持与源数据库实例对象名一致且无法修改。 – 表级迁移：将选择的表级对象迁移至目标数据库。 – 库级迁移：将选择的库级对象迁移至目标数据库。 如果有切换源数据库的操作或源库迁移对象变化的情况，请务必在选择迁移对象前单击右上角的，以确保待选择的对象为最新源数据库对象。 **说明** – 若选择部分数据库进行迁移时，由于存储过程、视图等对象可能与其他数据库的表存在依赖关系，若所依赖的表未迁移，则会导致迁移失败。建议您在迁移之前进行确认，或选择全部数据库进行迁移。 – 选择对象的时候，对象名称的前后空格不显示，中间如有多个空格只显示一个空格。 – 选择对象的时候支持搜索，以便您快速选择需要的数据库对象。

6. 在“预检查”页面，进行迁移任务预校验，校验是否可进行任务迁移。
   – 查看检查结果，如有不通过的检查项，需要修复不通过项后，单击“重新校验”按钮重新进行迁移任务预校验。
   预检查不通过项处理建议请参见《数据复制服务用户指南》中的“**预检查不通过项修复方法**”。
   – 预检查完成后，且所有检查项结果均通过时，单击“下一步”。

图 5-9 预检查



说明

所有检查项结果均通过时，若存在待确认项，需要阅读并确认详情后才可以继续执行下一步操作。

7. 在“任务确认”页面，设置迁移任务的启动时间、任务异常通知设置、SMN主题、时延阈值、任务异常自动结束时间，并确认迁移任务信息无误后，单击“启动任务”，提交迁移任务。

图 5-10 任务启动设置

表 5-8 任务启动设置

参数	描述
启动时间	迁移任务的启动时间可以根据业务需求，设置为“立即启动”或“稍后启动”，优选“稍后启动”。 **说明** 预计迁移任务启动后，会对源数据库和目标数据库的性能产生影响，建议您将任务启动时间设定在业务低峰期，同时预留2-3天校对数据。
任务异常通知设置	该项为可选参数，开启之后，选择对应的SMN主题。当迁移任务状态异常时，系统将发送通知。
SMN主题	“任务异常通知设置”项开启后可见，需提前在SMN上申请主题并添加订阅。 SMN主题申请和订阅可参考《消息通知服务用户指南》。
时延阈值	在增量迁移阶段，源数据库和目标数据库之间的实时同步有时会存在一个时间差，称为时延，单位为秒。 时延阈值设置是指时延超过一定的值后（时延阈值范围为0—3600s），DRS可以发送告警通知。告警通知将在时延稳定超过设定的阈值6min后发送，避免出现由于时延波动反复发送告警通知的情况。 **说明** – 首次进入增量迁移阶段，会有较多数据等待同步，存在较大的时延，属于正常情况，不在此功能的监控范围之内。 – 设置时延阈值之前，需要设置任务异常通知。 – 当时延阈值设置为0时，不会发送通知给收件人。

参数	描述
任务异常自动结束时间（天）	设置任务异常自动结束天数，输入值必须在14-100之间，默认值14。 **说明** 异常状态下的任务仍然会计费，而长时间异常的任务无法续传和恢复。设置任务异常自动结束天数后，异常且超时的任务将会自动结束，以免产生不必要的费用。

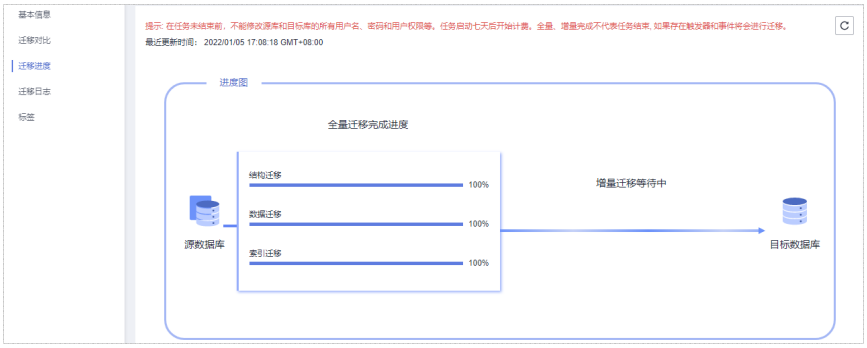
8. 迁移任务提交后，开始启动迁移任务，您可以返回“实时迁移管理”页面，查看迁移任务状态。

步骤2 任务管理。

迁移任务启动后，会经历全量迁移和增量迁移两个阶段，对于不同阶段的迁移任务，您可以进行任务管理。

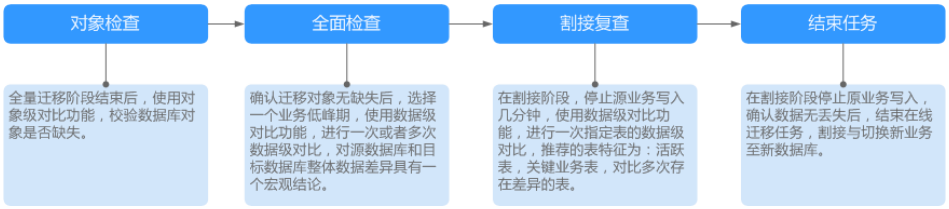
- 全量迁移
  - 查看迁移进度：全量迁移中的任务，您可单击任务名称，在“迁移进度”页签下，查看全量迁移进度，您可以查看结构、数据、索引迁移的进度，也查看具体迁移对象的迁移进度。当全量迁移进度显示为100%，表示全量迁移已经完成。
  - 查看迁移明细：迁移明细中，您可以查看具体迁移对象的迁移进度，当“对象数目”和“已迁移对象”相等时，表示该对象已经迁移完成，可通过“查看详情”查看每个对象的迁移进度。仅白名单用户该支持功能，您可以通过提交工单的方式进行申请使用。
- 增量迁移
  - 查看时延监控：全量迁移完成后，开始进行增量迁移。对于增量迁移中的任务，您可单击任务名称，在“迁移进度”页签下，查看增量迁移同步时延，**当时延为0s时，说明源数据库和目标数据库的数据是实时同步的。**您也可以使用“迁移对比”页签查看一致性。

图 5-11 查看时延监控



  - 查看迁移对比：为了尽可能减少业务的影响和业务中断时间，增量迁移中的任务，您可单击任务名称，在“迁移对比”页签下，建议按照如下流程进行迁移对比，以便确定合适的业务割接时机。

图 5-12 迁移对比流程



具体的迁移对比操作及注意事项请参考《数据复制服务用户指南》中“**对比迁移项**”章节。

步骤3 割接建议。

建议您选择一个业务低峰期，开始正式系统割接流程。割接前，请您确认至少在业务低峰期有过一次完整的数据对比。可以结合数据对比的“稍后启动”功能，选择业务低峰期进行数据对比，以便得到更为具有参考性的对比结果。由于同步具有轻微的时差，在数据持续操作过程中进行对比任务，可能会出现少量数据不一致对比结果，从而失去参考意义。

1. 先中断业务（如果业务负载非常轻，也可以尝试不中断业务）。
2. 在源数据库端执行如下语句，并观察在1-5分钟内若无任何新会话执行SQL，则可认为业务已经完全停止。

```
db.currentOp()
```

说明

上述语句查询到的进程列表中，包括DRS迁移实例的连接，您需要确认除DRS迁移实例的连接外无任何新会话执行SQL，即可认为业务已经完全停止。

3. 通过DRS迁移任务监控页面进行观察同步时延，保持实时同步时延为0，并稳定保持一段时间；同时，您可以使用数据级对比功能，进行割接前的最后一次数据级对比，耗时可参考之前的对比记录。
   - 如果时间允许，则选择全部对比。
   - 如果时间不允许，则推荐对比活跃表，关键业务表，第二步对比多次存在差异的表等。
4. 确定系统割接时机，业务系统指向本云数据库，业务对外恢复使用，迁移完成。

步骤4 迁移结束。

1. 结束迁移任务：业务系统和数据库切换至本云后，为了防止源数据库的操作继续同步到目标数据库，造成数据覆盖问题，此时您可选择结束迁移任务，该操作仅删除了迁移实例，迁移任务仍显示在任务列表中，您可以进行查看或删除。结束迁移任务后，DRS将不再计费。
2. 删除迁移任务：对于已结束的迁移任务，您可选择删除任务。该操作将一并删除迁移任务，删除迁移任务后，该任务将不会出现在任务列表中。

----结束

6 ECS 自建 MongoDB 迁移 DDS

操作场景

数据复制服务（Data Replication Service，简称DRS）支持将ECS自建数据库的数据迁移到本云文档数据库服务（Document Database Service，以下简称DDS）实例。通过DRS提供的实时迁移任务，实现在数据库迁移过程中业务和数据库不停机，业务中断时间最小化。

本章节主要介绍了通过DRS将ECS自建数据库实时迁移至本云DDS的任务配置流程。支持以下网络场景：

- 源数据库和目标数据库属于同一个VPC网络内
- 源数据库和目标数据库属于不同VPC网络内

资源规划

表 6-1 资源规划

类别	子类	规划	备注
VPC	VPC名称	vpc-dds	自定义，易理解可识别。
	所属Region	华南-广州	选择和自己业务区最近的Region，减少网络时延。
	可用区	可用区一	-
	子网网段	10.0.0.0/24	子网选择时建议预留足够的网络资源。
	子网名称	subnet-default	自定义，易理解可识别。
ECS	ECS名称	ecs-mongodb	自定义，易理解可识别。
	规格	s6.xlarge.2 4vCPUs|8GiB	本示例中选择的规格。 实际选择的规格需要结合业务场景选择，请参考**弹性云服务器的实例规格**。
	操作系统	CentOS 7.6 64	-
	系统盘	通用型SSD 40GiB	-

类别	子类	规划	备注
	数据盘	超高IO 100GiB	-
	弹性IP	现在购买	因为迁移任务会选择“公网网络”，因此此处需要购买弹性IP。
DDS	DDS 实例名	dds-test	自定义，易理解可识别。
	数据库引擎	DDS	-
	兼容的数据库版本	4.4	-
	可用区类型	单可用区	-
	可用区	可用区一	-
	性能规格	增强 II 型	-
	CPU架构	X86 8 vCPUs | 32GB	-
DRS迁移任务	迁移任务名	DRS-dds	自定义。
	源数据库引擎	MongoDB	本示例中源数据库为自建MongoDB，即在华为云弹性云服务器上安装社区版MongoDB。
	目标数据库引擎	DDS	本示例中目标数据库为华为云DDS实例。
	网络类型	公用网络	本示例中采用公用网络。

网络示意图

图 6-1 同一 VPC 网络

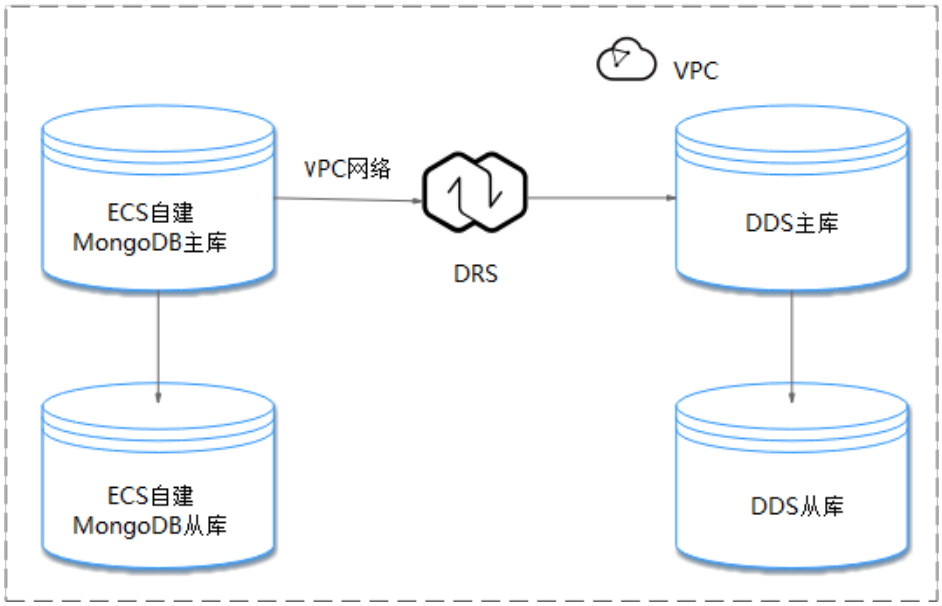
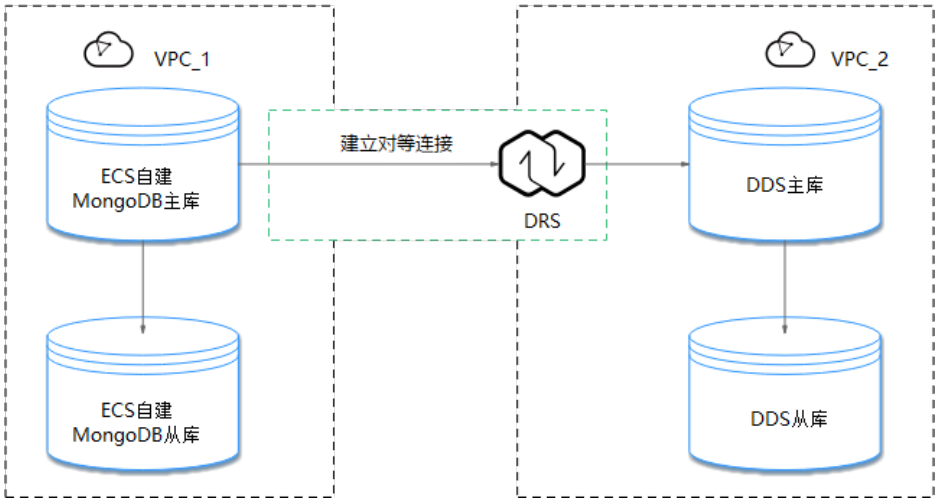
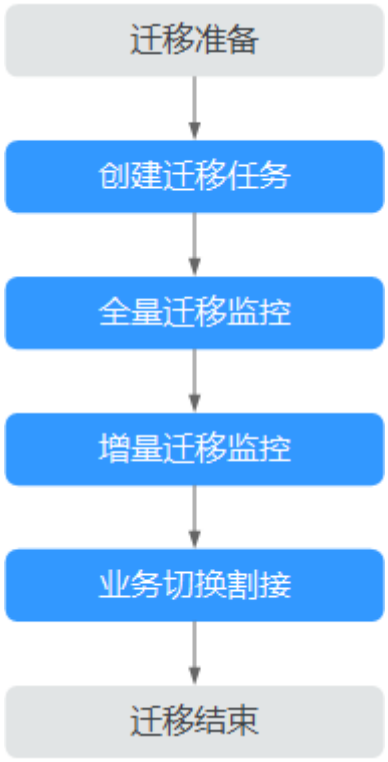


图 6-2 同区域不同 VPC 网络



迁移流程

图 6-3 迁移流程图



迁移建议（重要）

- 数据库迁移与环境多样性和人为操作均有密切关系，为了确保迁移的平顺，建议您在进行正式的数据库迁移之前进行一次演练，可以帮助您提前发现问题并解决问题，如何最小化对数据库的影响请参考如下建议。
- 强烈建议您在启动任务时选择“稍后启动”功能，将启动时间设置在业务低峰期，相对静止的数据可以有效提升一次性迁移成功率，避免迁移对业务造成性能影响。

迁移须知（重要）

须知

在创建迁移任务之前，请您务必仔细阅读迁移须知。

- 支持的源和目标数据库

  表 6-2 支持的数据库

源数据库	目标数据库
• 本地自建Mongo数据库（3.2、3.4、3.6、4.0、4.2、4.4、5.0版本） • ECS自建Mongo数据库（3.2、3.4、3.6、4.0、4.2、4.4、5.0版本） • 其他云上Mongo数据库（3.2、3.4、3.6、4.0、4.2、4.4、5.0版本 • DDS实例（3.2、3.4、4.0、4.2、4.4、5.0版本） **说明** 源数据库为DDS 3.2版本集群实例时，仅支持单全量迁移，不支持增量迁移。 DDS 5.0版本当前仅支持副本集，不支持集群。 1. 如果源库选择DDS实例，则此链路的数据库引擎是“DDS”，否则，此链路的数据库引擎是“MongoDB（数据库种类）。	• DDS实例（3.4、4.0、4.2、4.4、5.0版本） **说明** 仅支持目标库版本等于或高于源库版本。 DDS 5.0版本当前仅支持副本集，不支持集群。

  可通过以下命令，查询源数据库版本是否符合要求的内容：

```
db.version()
```

- 若迁移模式选择“全量+增量”迁移，源数据库Oplog日志必须打开，该日志会占用一定的磁盘空间，需要保证源端磁盘有一定冗余，Oplog日志保留周期最少3天，可以根据迁移数据量实际设置。

其他详情请参考《数据复制服务实时迁移》中具体链路的“**使用须知**”。

迁移准备

1. **权限准备：**

   当使用 DRS 将ECS自建MongoDB数据库的数据迁移到本云DDS实例时，在不同迁移类型情况下，对源数据库和目标数据库的账号权限要求如**表6-3**：

表 6-3 迁移账号权限

迁移类型	全量迁移	全量+增量迁移
源数据库	• 副本集：连接源数据库的用户需要对待迁移库有read权限。 • 单节点：连接源数据库的用户需要对待迁移库有read权限。 • 集群：连接源数据库的用户需要对待迁移库有read权限，对config数据库有read权限。 • 如果需要迁移源数据库用户和角色信息，连接源数据库的用户需要对admin数据库的系统表system.users，system.roles有读权限。	• 副本集：连接源数据库的用户需要对待迁移库有read权限，对local数据库有read权限。 • 单节点：连接源数据库的用户需要对待迁移库有read权限，对local数据库有read权限。 • 集群：连接源数据库mongos节点的用户需要对待迁移库有read权限，对config数据库有read权限，连接源数据库分片节点的用户需要对admin数据库有readAnyDatabase权限，对local数据库有read权限。 • 如果需要迁移源数据库用户和角色信息，连接源数据库的用户需要对admin数据库的系统表system.users，system.roles有读权限。
目标数据库	连接目标数据库的用户需要对admin数据库有dbAdminAnyDatabase权限，对目标数据库有readWrite权限。对于目标数据库是集群的实例，迁移账号还要有对config数据的读权限	

- **源数据库权限设置：**

  需要确保源数据库MongoDB的账号权限具备**表6-3**的要求。若权限不足，需要在源数据库端开通高权限的账号。
- **目标数据库权限设置：**

  本云DDS实例使用初始账号即可。

2. **网络准备：**
   - 源数据库所在的region需要和目标DDS所在的region保持一致。
   - 源数据库可以与目标DDS实例在同一个VPC，也可以不在同一个VPC。
     - 当不在同一个VPC的时候，要求源数据库实例和目标端DDS实例所处的子网处于不同网段，此时需要通过建立对等连接实现网络互通。

具体操作请参见《虚拟私有云用户指南》中“**VPC对等连接**”章节。

- 当在同一VPC的时候，网络默认是互通的。

3. **安全规则准备：**
   - 同一VPC场景下，默认网络是连通的，不需要单独设置安全组。
   - 不同VPC场景下，通过建立对等连接就可以实现网络互通，不需要单独设置安全组。
4. **其他事项准备：**

   由于迁移过程不会迁移MongoDB数据库的相关参数，需要自行将上述信息导出后，手动添加到目标DDS中。

迁移步骤

步骤1 创建迁移任务。

1. 登录管理控制台，在服务列表中选择“数据库 > 数据复制服务”，进入数据复制服务信息页面。
2. 在“实时迁移管理”页面，单击右上角“创建迁移任务”，进入迁移任务信息页面。
3. 在“迁移实例”页面，填选任务名称、通知收件人、描述和迁移实例信息。

图 6-4 迁移实例信息

表 6-4 任务和描述

参数	描述
区域	当前所在区域，可进行切换。为了降低访问时延、提高访问速度，请就近选择靠近您业务的区域。
项目	当前区域对应的项目，可进行切换。
任务名称	任务名称在4-50位之间，必须以字母开头，不区分大小写，可以包含字母、数字、中划线或下划线，不能包含其他的特殊字符。
描述	描述不能超过256位，且不能包含! = < > & ' " \ 特殊字符。

表 6-5 迁移实例信息

参数	描述
数据流动方向	选择入云。
源数据库引擎	选择“MongoDB”。
目标数据库引擎	选择“DDS”。
网络类型	选择“VPC网络”。
目标数据库实例	您所创建的本云DDS实例。
迁移模式	此处以全量+增量模式为示例。 – 全量：该模式为数据库一次性迁移，适用于可中断业务的数据库迁移场景，全量迁移将非系统数据库的全部数据库对象和数据一次性迁移至目标端数据库，包括：表、视图、存储过程等。 **说明** 如果用户只进行全量迁移时，建议停止对源数据库的操作，否则迁移过程中源数据库产生的新数据不会同步到目标数据库。 – 全量+增量：该模式为数据库持续性迁移，适用于对业务中断敏感的场景，通过全量迁移过程中完成的目标端数据库的初始化后，增量迁移阶段通过解析日志等技术，将源端和目标端数据库保持数据持续一致。 **说明** 选择“全量+增量”迁移模式，增量迁移可以在全量迁移完成的基础上实现数据的持续同步，无需中断业务，实现迁移过程中源业务和数据库继续对外提供访问。
源数据库实例类型	迁移模式为“全量+增量”时，需要根据源数据库的具体来源进行设置。此处以“非集群”为示例。 – 当源库类型属于集群时，该项需要设置为集群。 – 当源库类型属于副本集或者单节点时，该项需要设置为非集群。
增量数据获取方式	当源端实例类型设置为“集群”时，增量同步时需要选择数据获取方式。 – oplog：支持MongoDB 3.2及以上版本，DRS直接连接源数据库实例的每一个Shard进行数据抽取。选择此方式时，必须关闭源库实例集合均衡器Balancer，测试连接时需要填写源数据库每一个Shard的连接信息。 – changeStream：支持MongoDB 4.0及以上版本，DRS连接源数据库实例的mongos进行数据抽取，选择此方式时，源数据库实例必须开启WiredTiger存储引擎，推荐此选项。 **说明** “changeStream”方式目前仅支持白名单用户，需要提交工单申请才能使用。您可以在管理控制台右上角，选择“工单 > 新建工单”，完成工单提交。

参数	描述
源端分片个数	当源端实例类型设置为“集群”且增量数据获取方式为“oplog”时，需要填写源端数据库分片个数。 源端数据库分片个数默认最小值为2，最大值为32，你需要根据源库实际的集群分片个数设置该值大小。

4. 在“源库及目标库”信息页面，迁移实例创建成功后，填选源库信息和目标库信息，建议您单击“源库和目标库”处的“测试连接”，分别测试并确定与源库和目标库连通后，勾选协议，单击“下一步”。

图 6-5 源库及目标库信息

表 6-6 源库信息

参数	描述
源库类型	选择“自建库”。

参数	描述
VPC	源数据库实例所在的虚拟专用网络，可以对不同业务进行网络隔离。您需要创建或选择所需的虚拟私有云。如何创建虚拟私有云，请参见《虚拟私有云用户指南》中的“**创建虚拟私有云基本信息及默认子网**”章节。
子网	通过子网提供与其他网络隔离的、可以独享的网络资源，以提高网络安全。子网在可用分区内才会有效，创建源数据库实例的子网需要开启DHCP功能，在创建过程中也不能关闭已选子网的DHCP功能。如何创建子网，请参见《虚拟私有云用户指南》中的“**创建虚拟私有云基本信息及默认子网**”章节。
IP地址或域名	配置源MongoDB数据库实例的访问地址或域名。
端口	配置源MongoDB数据库实例的服务端口，可输入范围为1~65535间的整数。
数据库用户名	访问源MongoDB数据库的用户名。
数据库密码	访问源MongoDB数据库的用户名所对应的密码。
SSL安全连接	您可以选择开启SSL安全连接，对迁移链路进行加密，开启之后，需要您上传加密证书。

表 6-7 目标库信息

参数	描述
数据库实例名称	默认为创建迁移任务时选择的已创建的本云DDS实例，不可进行修改。
数据库用户名	访问目标数据库本云DDS的用户名。
数据库密码	访问目标数据库本云DDS的用户名所对应的密码。

5. 在“迁移设置”页面，设置迁移对象，单击“下一步”。

图 6-6 设置迁移对象

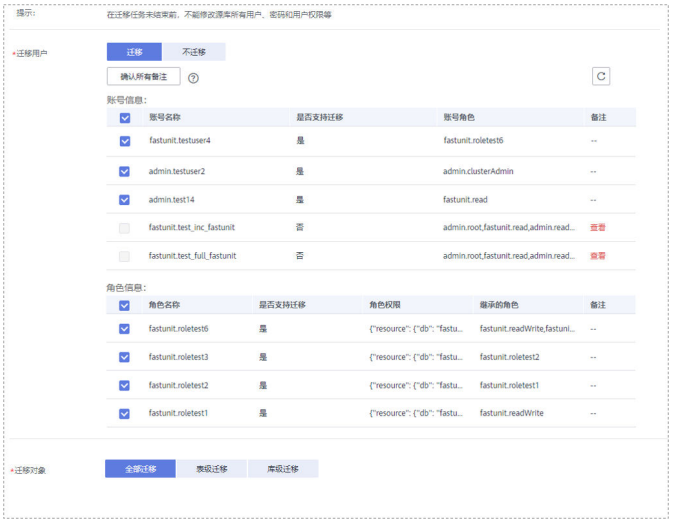


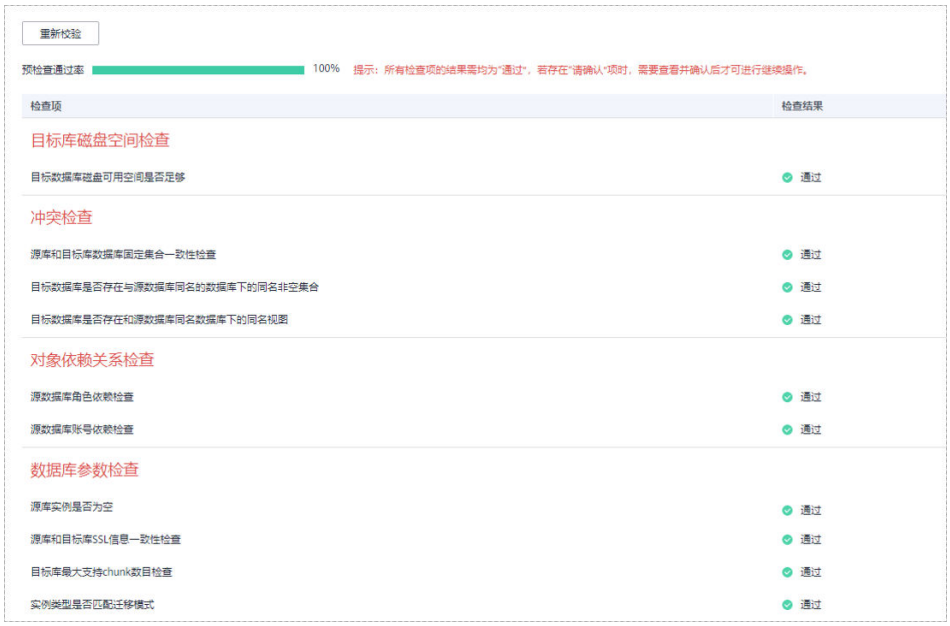
表 6-8 迁移对象

参数	描述
迁移用户	常见的迁移用户一般分为两类：支持迁移的用户和不支持迁移的用户。您可以根据业务需求选择“迁移”或者“不迁移”，其中，不支持迁移的账号或者未选择迁移的账号将在目标数据库中缺失，需要先确保业务不受影响。 – 迁移 当您选择迁移用户时，请参见《数据复制服务用户指南》中“迁移用户”章节进行数据库用户及角色的处理。 – 不迁移 迁移过程中，将不进行数据库用户及角色的迁移。
迁移对象	您可以根据业务需求，选择全部对象迁移、表级迁移或者库级迁移。 – 全部迁移：将源数据库中的所有对象全部迁移至目标数据库，对象迁移到目标数据库实例后，对象名将会保持与源数据库实例对象名一致且无法修改。 – 表级迁移：将选择的表级对象迁移至目标数据库。 – 库级迁移：将选择的库级对象迁移至目标数据库。 如果有切换源数据库的操作或源库迁移对象变化的情况，请务必在选择迁移对象前单击右上角的，以确保待选择的对象为最新源数据库对象。 **说明** – 若选择部分数据库进行迁移时，由于存储过程、视图等对象可能与其他数据库的表存在依赖关系，若所依赖的表未迁移，则会导致迁移失败。建议您在迁移之前进行确认，或选择全部数据库进行迁移。 – 选择对象的时候，对象名称的前后空格不显示，中间如有多个空格只显示一个空格。 – 选择对象的时候支持搜索，以便您快速选择需要的数据库对象。

6. 在“预检查”页面，进行迁移任务预校验，校验是否可进行任务迁移。
   - 查看检查结果，如有不通过的检查项，需要修复不通过项后，单击“重新校验”按钮重新进行迁移任务预校验。

     预检查不通过项处理建议请参见《数据复制服务用户指南》中的“**预检查不通过项修复方法**”。
   - 预检查完成后，且所有检查项结果均通过时，单击“下一步”。

图 6-7 预检查



说明

所有检查项结果均通过时，若存在待确认项，需要阅读并确认详情后才可以继续执行下一步操作。

7. 在“任务确认”页面，设置迁移任务的启动时间、任务异常通知设置、SMN主题、时延阈值、任务异常自动结束时间，并确认迁移任务信息无误后，单击“启动任务”，提交迁移任务。

图 6-8 任务启动设置

表 6-9 任务启动设置

参数	描述
启动时间	迁移任务的启动时间可以根据业务需求，设置为“立即启动”或“稍后启动”，优选“稍后启动”。 **说明** 预计迁移任务启动后，会对源数据库和目标数据库的性能产生影响，建议您将任务启动时间设定在业务低峰期，同时预留2-3天校对数据。
任务异常通知设置	该项为可选参数，开启之后，选择对应的SMN主题。当迁移任务状态异常时，系统将发送通知。
SMN主题	“任务异常通知设置”项开启后可见，需提前在SMN上申请主题并添加订阅。 SMN主题申请和订阅可参考《消息通知服务用户指南》。
时延阈值	在增量迁移阶段，源数据库和目标数据库之间的实时同步有时会存在一个时间差，称为时延，单位为秒。 时延阈值设置是指时延超过一定的值后（时延阈值范围为0—3600s），DRS可以发送告警通知。告警通知将在时延稳定超过设定的阈值6min后发送，避免出现由于时延波动反复发送告警通知的情况。 **说明** – 首次进入增量迁移阶段，会有较多数据等待同步，存在较大的时延，属于正常情况，不在此功能的监控范围之内。 – 设置时延阈值之前，需要设置任务异常通知。 – 当时延阈值设置为0时，不会发送通知给收件人。
任务异常自动结束时间（天）	设置任务异常自动结束天数，输入值必须在14-100之间，默认值14。 **说明** 异常状态下的任务仍然会计费，而长时间异常的任务无法续传和恢复。设置任务异常自动结束天数后，异常且超时的任务将会自动结束，以免产生不必要的费用。

8. 迁移任务提交后，您可以返回“实时迁移管理”页面，查看迁移任务状态。

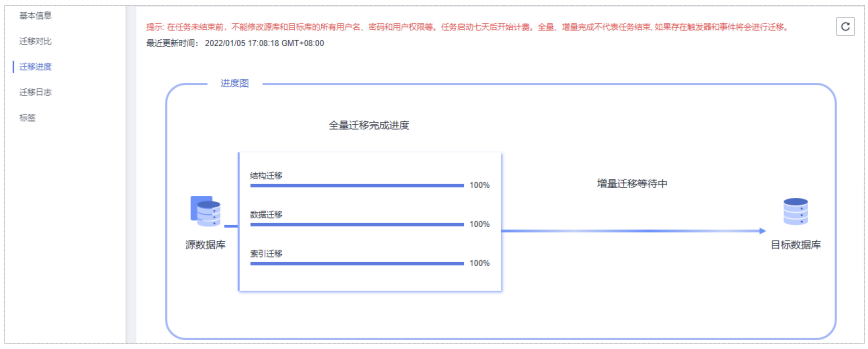
步骤2 任务管理。

迁移任务启动后，会经历全量迁移和增量迁移两个阶段，对于不同阶段的迁移任务，您可以进行任务管理。

- 全量迁移
  - 查看迁移进度：全量迁移中的任务，您可单击任务名称，在“迁移进度”页签下，查看全量迁移进度，您可以查看结构、数据、索引迁移的进度，也查看具体迁移对象的迁移进度。当全量迁移进度显示为100%，表示全量迁移已经完成。
  - 查看迁移明细：迁移明细中，您可以查看具体迁移对象的迁移进度，当“对象数目”和“已迁移对象”相等时，表示该对象已经迁移完成，可通过“查看详情”查看每个对象的迁移进度。仅白名单用户该支持功能，您可以通过提交工单的方式进行申请使用。
- 增量迁移

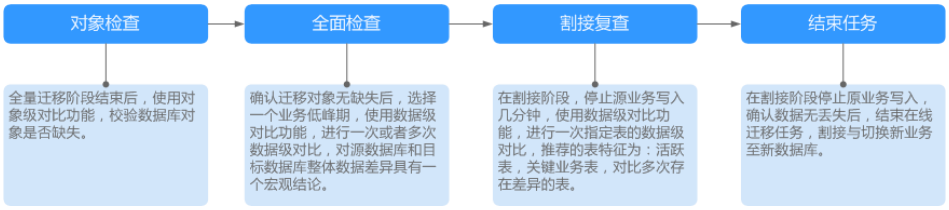
- 查看时延监控：全量迁移完成后，开始进行增量迁移。对于增量迁移中的任务，您可单击任务名称，在“迁移进度”页签下，查看增量迁移同步时延，**当时延为0s时，说明源数据库和目标数据库的数据是实时同步的。**您也可以使用“迁移对比”页签查看一致性。

图 6-9 查看时延监控



- 查看迁移对比：为了尽可能减少业务的影响和业务中断时间，增量迁移中的任务，您可单击任务名称，在“迁移对比”页签下，建议按照如下流程进行迁移对比，以便确定合适的业务割接时机。

图 6-10 迁移对比流程



具体的迁移对比操作及注意事项请参考《数据复制服务用户指南》中“**对比迁移项**”章节。

步骤3 割接建议。

建议您选择一个业务低峰期，开始正式系统割接流程。割接前，请您确认至少在业务低峰期有过一次完整的数据对比。可以结合数据对比的“稍后启动”功能，选择业务低峰期进行数据对比，以便得到更为具有参考性的对比结果。由于同步具有轻微的时差，在数据持续操作过程中进行对比任务，可能会出现少量数据不一致对比结果，从而失去参考意义。

1. 先中断业务（如果业务负载非常轻，也可以尝试不中断业务）。
2. 在源数据库端执行如下语句，并观察在1-5分钟内若无任何新会话执行SQL，则可认为业务已经完全停止。

```
db.currentOp()
```

说明

上述语句查询到的进程列表中，包括DRS迁移实例的连接，您需要确认除DRS迁移实例的连接外无任何新会话执行SQL，即可认为业务已经完全停止。

3. 通过DRS迁移任务监控页面进行观察同步时延，保持实时同步时延为0，并稳定保持一段时间；同时，您可以使用数据级对比功能，进行割接前的最后一次数据级对比，耗时可参考之前的对比记录。
   - 如果时间允许，则选择全部对比。

- 如果时间不允许，则推荐对比活跃表，关键业务表，第二步对比多次存在差异的表等。

4. 确定系统割接时机，业务系统指向本云数据库，业务对外恢复使用，迁移完成。

步骤4 迁移结束。

1. 结束迁移任务：业务系统和数据库切换至本云后，为了防止源数据库的操作继续同步到目标数据库，造成数据覆盖问题，此时您可选择结束迁移任务，该操作仅删除了迁移实例，迁移任务仍显示在任务列表中，您可以进行查看或删除。结束迁移任务后，DRS将不再计费。
2. 删除迁移任务：对于已结束的迁移任务，您可选择删除任务。该操作将一并删除迁移任务，删除迁移任务后，该任务将不会出现在任务列表中。

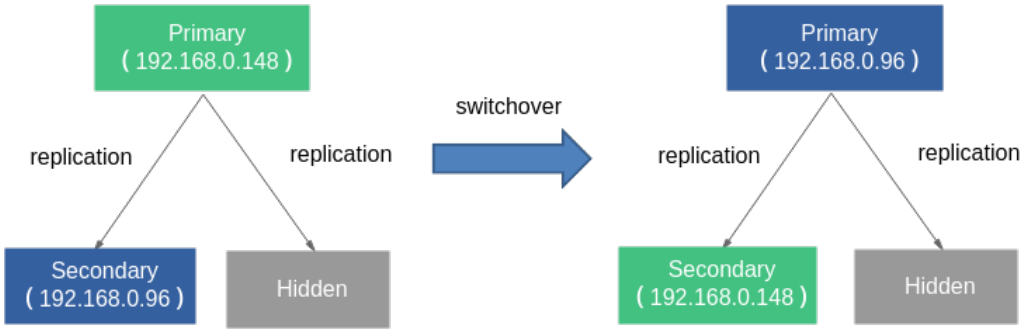
----结束

7 如何实现副本集高可用和读写分离

DDS副本集不仅可以通过**存储多份数据副本**来保证数据的高可靠性，还可以通过**自动主备切换**机制来保证服务的高可用。通过客户端读取不同的数据副本，可以提高数据的读取性能。请使用高可用方式连接副本集实例，您也可以通过设置来实现读写分离。否则，将无法体验副本集的高可用性和高读取性能。

副本集的**主节点不是固定的**，当出现副本集配置改变、主节点宕机、人为主备切换等状况，会导致主备节点切换，此时副本集可能会选举出新主节点，原来的主节点则降为备节点。如下图所示：

图 7-1 主备切换示意图



连接副本集实例（高可用方式）

DDS副本集包含主、备、隐藏节点。其中，隐藏节点对用户不可见。您需要使用一种高可用的方式（即同时连接主节点和备节点的IP和端口）连接副本集实例，从而实现副本集的读写分离和高可用。

以下主要介绍如何使用URL和Java实现高可用连接实例的方法。

方法一：通过URL连接副本集实例

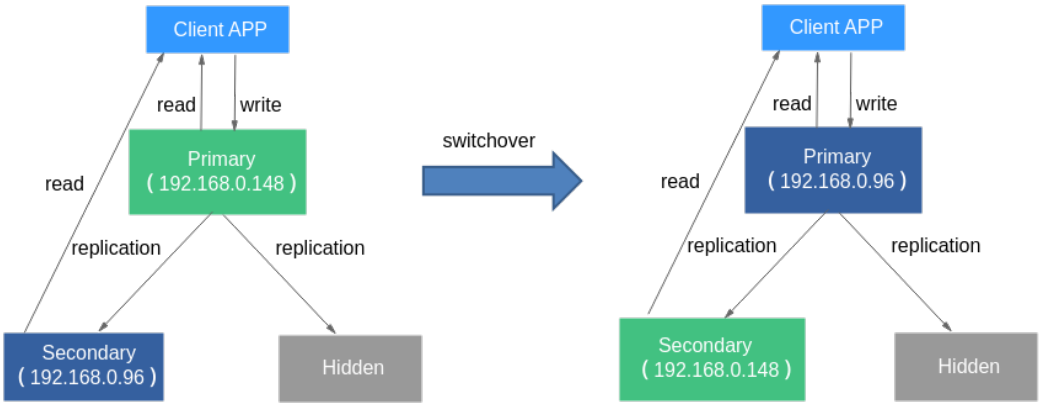
您可以在实例管理页面，单击目标实例名称，进入基本信息页面。单击“连接管理 > 内网连接”，在“内网高可用连接地址”处即可获取到当前实例的连接地址。

图 7-2 获取内网高可用地址

示例：**mongodb://rwuser:****@*192.168.0.148:8635,192.168.0.96:8635*/test?authSource=admin&replicaSet=replica**

其中，“192.168.0.148:8635,192.168.0.96:8635”中包含一个主节点和一个备节点的IP及端口号。即使发生主备切换而更换了主节点，客户端仍然可以成功连接副本集实例。同时，提供副本集多个IP及端口号，可以扩展整个数据库的读写性能，如下图所示：

图 7-3 数据读写示意图



方法二：Java驱动连接副本集实例

示例代码：

```
MongoClientURI connectionString = new MongoClientURI("mongodb://
rwuser:****@192.168.0.148:8635,192.168.0.96:8635/test?authSource=admin&replicaSet=replica");
MongoClient client = new MongoClient(connectionString);
MongoDatabase database = client.getDatabase("test");
MongoCollection<Document> collection = database.getCollection("mycoll");
```

表 7-1 参数说明

参数	说明
rwuser:****	启动鉴权的用户名和密码。
192.168.0.148:8635, 192.168.0.96:8635	副本集主、备节点的IP及端口号。

参数	说明
test	待连接的数据库名称。
authSource=admin	表示鉴权时，用户名所属的数据库。
replicaSet=replica	副本集实例类型名称。

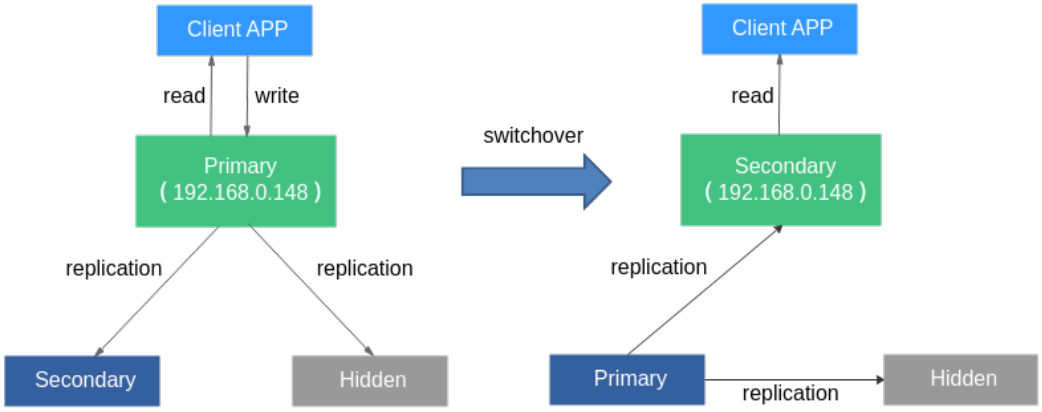
连接副本集实例（非友好方式）

通过连接地址连接副本集实例

mongodb://rwuser:**@*192.168.0.148:8635*/test?authSource=admin&replicaSet=replica**

其中，“192.168.0.148:8635”为暂时的主节点IP及端口号。当发生主备切换而更换了主节点，客户端将无法连接到副本集实例主节点，因为客户端无法知道新选举出的主节点的IP及端口号信息，导致整个数据库服务发生故障。同时，只提供主节点IP及端口号，读写只能在固定的主机上操作，无法扩展数据库的读写性能，如图下图所示：

图 7-4 数据读写示意图



读写分离

通过如下高可用地址举例说明连接DDS副本集实例：

```
mongodb://rwuser:<password>@192.168.xx.xx:8635,192.168.xx.xx:8635/test?
authSource=admin&replicaSet=replica&readPreference=secondaryPreferred
```

其中：数据库账号为rwuser，所属数据库为admin。

说明

连接实例后，读请求将优先发给Secondary节点实现读写分离。当主备关系发生变化时，自动将写操作切换到新的Primary节点上，以保证服务的高可用。

8 通过设置数据分片提升性能

对于DDS集群实例，如果某个集合的存储量很高，建议对该集合设置数据分片。分片是将数据按照某种方式拆分，将其分散存放在不同的机器上，以充分利用各分片节点的**存储空间**和**计算性能**。

⚠ 注意

对于非空集合开启分片过程中，对应集合会加锁，阻塞业务。集合数据量越大，分片（sharding collection）耗时越长，因此分片前务必确保对应分片表上的业务处于低谷期。

设置数据分片

下面以数据库mytable，集合mycoll，字段“name”为分片键举例说明。

步骤1 通过mongo shell登录分片集群实例。

步骤2 判断集合是否已分片。

```
use <database>
db.<collection>.getShardDistribution()
```

示例：

```
use mytable
db.mycoll.getShardDistribution()
```

```
mongos> db.mycoll.getShardDistribution()
Collection test.mycoll is not sharded.
```

步骤3 对集合所属的数据库启用分片功能。

- 方式一

  ```
  sh.enableSharding("<database>")
  ```

  示例：

  ```
  sh.enableSharding("mytable")
  ```

- 方式二

  ```
  use admin
  db.runCommand({enablesharding:"<database>"})
  ```

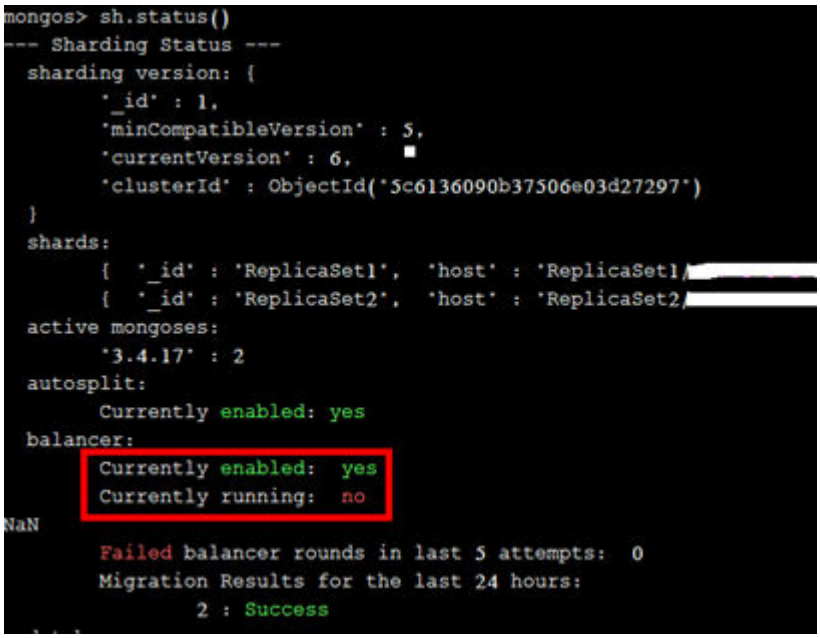

步骤4 对集合进行分片。

- 方式一
  ```
  sh.shardCollection("<database>.<collection>",{"<keyname>":<value> })
  ```
  示例：
  ```
  sh.shardCollection("mytable.mycoll",{"name":"hashed"},false,{numInitialChunks:5})
  ```
- 方式二
  ```
  use admin
  db.runCommand({shardcollection:"<database>.<collection>",key:{"keyname":<value> }})
  ```

表 8-1 参数说明

参数	说明
<database>	数据库名称。
<collection>	集合名称。
<keyname>	分片键。 集群实例将根据该值进行数据分片，请结合实际业务为集合选择合适的分片键，具体操作请参见**选择合适的分片键**。
<value>	基于分片键的范围查询的排序方式。 • 1：表示索引升序。 • -1：表示索引降序。 • hashed：表示使用Hash分片，通常能将写入均衡分布到各个分片节点。 更多信息，请参见**sh.shardCollection()**。
numInitialChunks	可选。当使用Hash分片键对空集合进行分片时，指定初始创建的最小分片数。

步骤5 查看数据库在各分片节点的数据存储情况。
```
sh.status()
```
示例：



----结束

选择合适的分片键

- **背景**

  分片集群中数据的分片以集合为基础单位，集合中的数据通过分片键被分成多个部分。分片键是在集合中选择的一个合适的字段，数据拆分时以该分片键的值为依据均衡地分布到所有分片中。如果您没有选择到合适的分片键，可能会降低集群的使用性能，出现执行分片语句时执行过程卡住的问题。

  分片键一旦设置后不能再更改。如果未选择到合适的分片键，需要使用正确的分片策略，将数据迁移到新的集合后重新执行分片。

- **合适的分片键的特点**
  - 所有的插入、更新以及删除操作，将会均匀分发到集群中的所有分片中。
  - key的分布足够离散。
  - 尽量避免scatter-gather查询。

  如果所选分片键不具备以上所有特点，将会影响集群的读写扩展性。例如，通过**find()**操作读取的工作量在分片中非均匀分布，最终会产生查询热分片。同样，如果写工作量（插入、更新和修改）在分片中非均匀分布，最终会产生写热分片，严重限制分片的优势。因此，您需要根据应用读写状态（重读取还是重写入）、经常查询及写入的数据等业务需求，调整您的分片键。

  需要注意，对已有数据分片后，如果update请求的filter中未携带片键字段并且选项upsert:true或者multi:false，那么update 请求会报错，并返回“An upsert on a sharded collection must contain the shard key and have the simple collation.”

- **判断标准**

  您可以通过**表8-2**中的几个维度，判断所选分片键是否能够满足业务需求。

表 8-2 合理分片键的判断依据

判断依据	说明
片键基数	片键基数是指划分数据块的能力。例如，要记录某个学校的学生信息，由于学生的年龄比较集中，如果选择年龄作为分片键，同一个数据段中将存储很多同龄学生的数据，影响集群的性能以及可管理性。由于学生的学号唯一，如果选择学号作为分片键，分片基数较大，有利于数据的均匀分布。
写分布	若用户业务在同一时间段有大量写操作，则希望这些写操作能够均匀分布到各个分片上。如果数据分布策略为范围分片，并以一个单调递增的值作为分片键，此时，大量写入的数据同样是片键字段递增，数据将写入同一个分片。
读分发	若用户业务在同一时间段有大量读操作，则希望这些读操作能够均匀分布到各个分片上，以充分利用各分片节点的计算性能。
定向读	dds mongos查询路由器可以执行定向查询（只查询一个分片）或scatter/gather查询（查询所有分片）。只有查询中存在分片键，dds mongos才能定位到单一分片，因此，您需要选择在业务运行时可用于普遍查询的分片键。如果您选择合成的分片键，将无法在定向查询中使用该片键，所有的查询方式将变成scatter/gather查询，从而限制扩展读数据的能力。

选择合适的数据分布策略

分片集群支持将单个集合的数据分散存储在多个分片上，用户可以根据集合内文档的分片键来分布数据。

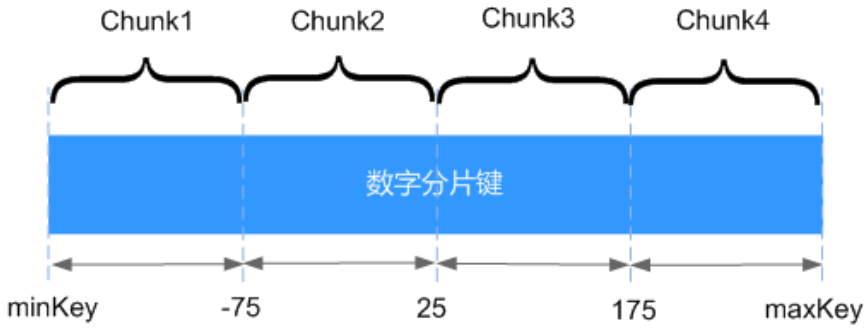
目前，主要支持两种数据分布策略，即范围分片（Range based sharding）和Hash分片（Hash based sharding），设置方式请参见**步骤4**。

下面分别介绍这两种数据分布策略以及各自的优缺点。

- **范围分片**

  基于范围进行分片，即集群按照分片键的范围把数据分成不同部分。假设有一个数字分片键，为一条从负无穷到正无穷的直线，每一个片键的值均在直线上进行标记。可以理解为将该直线划分为更短的不重叠的片段，并称之为数据块，每个数据块包含了分片键在一定的范围内的数据。

  图 8-1 数据分布示意图



如上图所示，x表示范围分片的片键，x的取值范围为[minKey,maxKey]，且为整型。将整个取值范围划分为多个chunk，每个chunk（通常配置为64MB）包含其中一小段的数据。其中，chunk1包含x值在[minKey, -75]中的所有文档，每个chunk的数据都存储在同一个分片上，每个分片可以存储多个chunk，并且chunk存储在分片中的数据会存储在config服务器中，dds mongos也会根据各分片上的chunk的数据自动执行负载均衡。

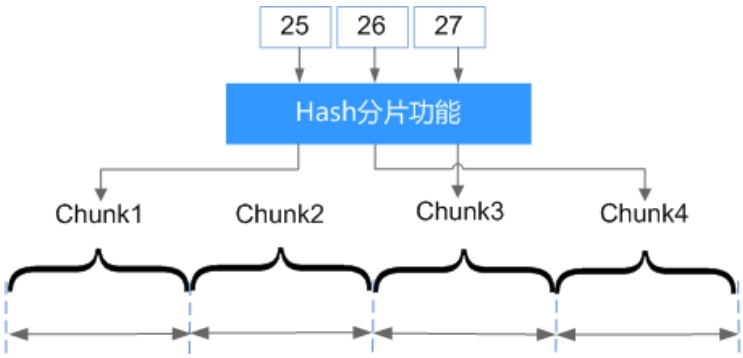
范围分片能够很好地满足范围查询的需求，例如，查询x的取值在[-60,20]中的文档，仅需dds mongos将请求路由到chunk2。

范围分片的缺点在于，如果分片键有明显递增（或递减）趋势，新插入的文档很大程度上会分布到同一个chunk，从而无法扩展写的能力。例如，使用“_id”作为分片键，集群自动生成id的高位值将是递增的时间戳。

- **Hash分片**

  根据用户的分片键值计算出Hash值（长度64bit且为整型），再按照范围分片策略，根据Hash值将文档分布到不同的chunk中。基于Hash分片主要的优势为保证数据在各节点上分布基本均匀，具有“相近”片键的文档很可能不会存储在同一个数据块中，数据的分离性更高。

图 8-2 数据分布示意图



Hash分片与范围分片互补，能将文档随机分散到各个chunk，充分扩展写能力，弥补范围分片的不足。但所有的范围查询要分发到后端所有的分片，才能获取满足条件的文档，查询效率低。

9 如何通过 SQL 优化来提升 DDS 性能

文档数据库属于NoSQL数据库，提供了可扩展的高性能数据解决方案，与关系型数据库（例如MySQL、SQLServer、Oracle）一样，在数据库设计、语句优化、索引创建等方面都会影响数据库的使用性能。

下面从不同维度，给出提升DDS使用性能的建议：

数据库和集合的创建

1. 使用**短字段名**，以节约存储空间。文档数据库与关系型数据库不同，集合中的每个文档都存储字段名，使用短字段名可以有效的节约存储空间。
2. 有效的控制集合中的文档数量，避免影响查询性能。如果有必要，可以进行定期归档。
3. 每条文档都提供默认的“_id”值，**禁止向“_id”中保存自定义值**。
4. **固定集合**相较其他集合，插入速度快，并且能够自动删除旧数据。用户可以根据业务需要，选择创建固定集合以提高性能。

更多规范建议请参见《文档数据库服务开发指南》中“**使用规范**”章节。

查询操作

索引

1. 根据业务需求，对经常查询的数据字段创建适当的索引。需注意，索引会占用一些空间，并且插入操作和索引更新会消耗资源。因此，建议每个集合的索引数量不超过5个。

   案例：出现数据查询缓慢，如果没有创建索引，建议对经常查询的数据字段创建适当的索引，优化查询速度。
2. 对于包含多个键的查询，建议创建包含这些键的复合索引。复合索引的键值顺序很关键，需遵循索引最左前缀原则，查询应包含最左索引字段，以索引创建顺序为准，与查询字段顺序无关。
3. 给索引添加TTL属性，自动筛选过期文档并删除。创建TTL的索引必须是日期类型。TTL索引是单字段索引，而非复合索引。
4. 需要在集合中某个字段上创建索引，但当集合中大量文档不包含该键值时，建议创建稀疏索引。
5. 创建文本索引时，字段指定text，而不是1或者-1。每个集合只有一个文本索引，但它可以为任意多个字段建立索引。

命令使用

1. 使用findOne方法，在数据库中查询匹配多个项目，将会在自然排序文件集合中返回第一个项目。如果需要返回多个文档，则使用find方法。
2. 如果查询无需返回整个文档，或只是用来判断键值是否存在，可以通过投影$project来限制返回字段，减少网络流量和客户端的内存使用。
3. 除了前缀样式查询，正则表达式查询执行的时间比大多数选择器更久，不建议使用索引。
4. 查询中的某些含“$”的操作符可能会降低使用性能。在业务中尽量不要使用该类操作符：$or、$nin、$not、$ne、$exists。

   说明

   - $or：有多少个条件就会查询多少次，最后合并结果集，建议替换为$in。
   - $nin：可能会匹配到大多数的索引，此时，查询优化器会退化为全表扫描。
   - $not：可能会导致查询优化器无法匹配到具体的索引，退化为全表扫描。
   - $ne：选择字段值不等于指定值的文档，如果多数为取相反值的文档，将会扫描整个索引。
   - $exists：对于松散的文档结构，查询必须遍历每一个文档。

   更多信息，请参见**MongoDB官方文档**。

注意事项

1. 操作符$where和$exists中不能使用索引。
2. 如果查询结果需要排序，尽量控制结果集的数量。
3. 涉及多个字段的索引时，尽量将用于精确匹配的字段放在索引的前面。
4. 如果查询条件中的键值顺序和复合索引中的创建顺序不一致，DDS会自动优化为查询跟索引顺序一致。
   - 修改操作

     通过操作符对文档进行修改，通常可以获得更好的性能。该方式不需要往返服务器来获取并修改文档数据，在序列化和传输数据上花费更少的时间。
   - 批量插入

     批量插入（batchInsert）可以减少数据向服务器的提交次数，提高使用性能。批量提交的数据的BSON Size不超过48MB。
   - 聚合运算

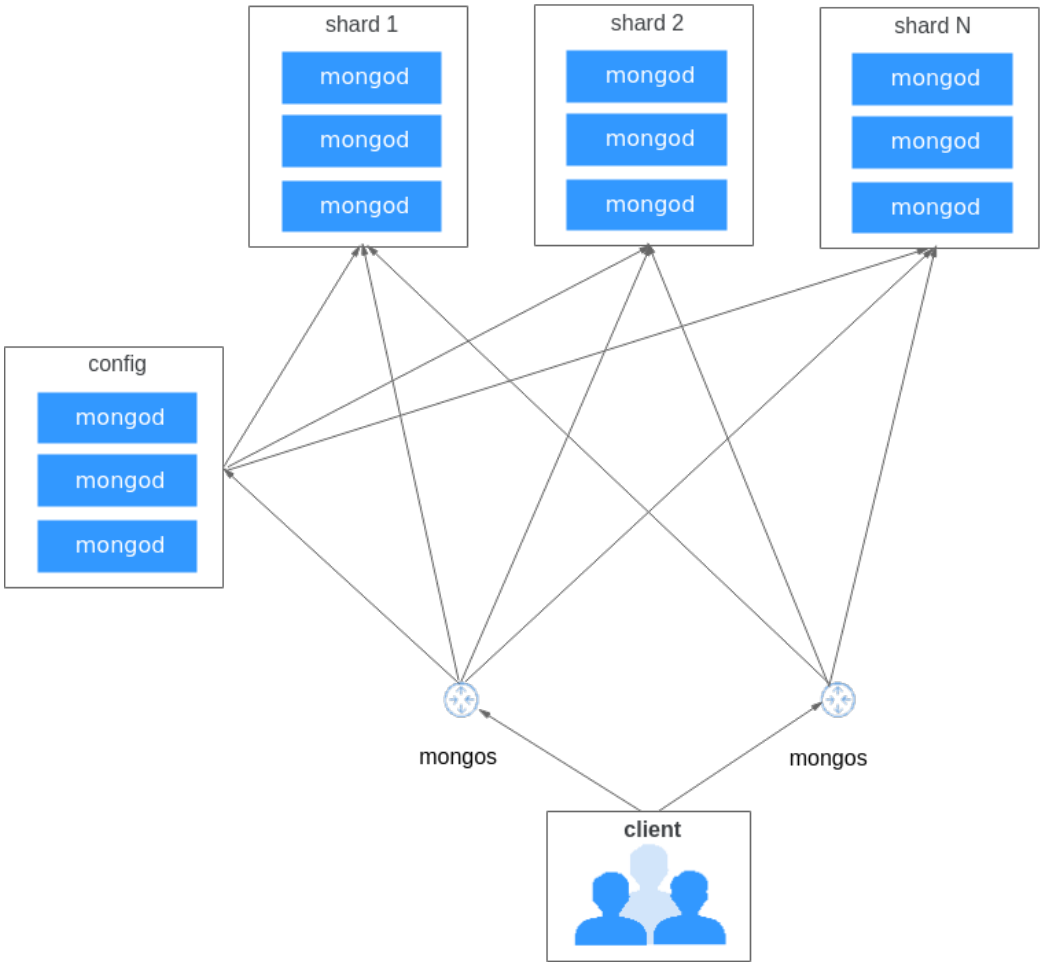
     聚合运算中，$match需前置于$group，减少$group操作符要处理的文档数量。

10 如何规避 dds mongos 路由缓存缺陷

背景信息

DDS作为基于分布式文件存储的数据库，以其可扩展性，高性能，开源，模式自由，面向文档等特点，逐步赢得了越来越多的使用者。下图为DDS集群架构示意图：

图 10-1 集群架构图



集群架构主要分为以下三部分：

- dds mongos：为单节点架构，负责提供对外供用户访问的接口，屏蔽分布式数据库内部的复杂性。一个DDS集群可以有2~12个dds mongos，您可以根据需要进行添加。
- config server：为一组副本集架构，负责存储整个集群的元数据信息，包括数据的路由信息和分片信息等内容。一个集群仅支持一个config server。
- shard server：为一组副本集架构，负责分片式存储用户数据。集群中可以添加多组副本集架构的shard server。

分片概念

分片是指将一个集合的数据，根据指定的shard key，相对均匀地分布保存在多个shard server上。这种指定了shard key的集合，称为分片集合。但是，如果并未对集合进行分片，则该集合的数据，只会全部存储在某一个shard server上。DDS集群模式允许分片集合和未分片集合在数据库中同时存在。

未分片的集合可以通过命令**sh.shardCollection**转为分片集合。对集合进行分片之前，需确保集合所属的数据库开启了分片功能，您可以通过命令**sh.enableSharding**开启分片功能。

dds mongos 路由缓存机制

用户数据存储在shard server中，元数据存储在config server中。路由信息属于元数据信息，即存储在config server中。当用户通过dds mongos对集群进行数据访问时，dds mongos会根据config server中的路由信息，将用户请求发送到对应的shard server上，进行数据访问。

但是，如果dds mongos在每次处理数据访问时，都从config server获取路由信息，很大程度上会影响性能。因此，在实现机制上，添加了缓存机制：将config server的路由信息缓存在dds mongos本地。该场景下，不但在config server中会存储路由信息，dds mongos的本地缓存中也可能会缓存路由信息。

dds mongos中并不是一定会存在缓存的路由信息，如果dds mongos上没有进行过任何数据操作，就没有缓存信息。并且，dds mongos上缓存的路由信息，也不一定是最新的config server的路由信息。因为dds mongos上缓存的路由信息，不是实时或者定时刷新的，而是lazy模式，是在特定的场景下被动触发的，包含但可能不限于如下几种触发场景：

- dds mongos启动时，从config server获取最新的路由信息，并缓存在本地。
- dds mongos第一次处理相关数据的请求：由于mongos本地没有缓存该相关数据的路由信息，将会触发更新相关的config server路由信息到dds mongos本地缓存的逻辑，在继续处理后续请求时，dds mongos已经缓存了相关数据的路由信息，会直接使用缓存中的路由信息来访问shard server。
- 在dds mongos上手工执行路由刷新命令。

说明

被动触发dds mongos的路由缓存刷新，只是刷新用户请求涉及到的元数据信息，而非刷新缓存中的全部内容。

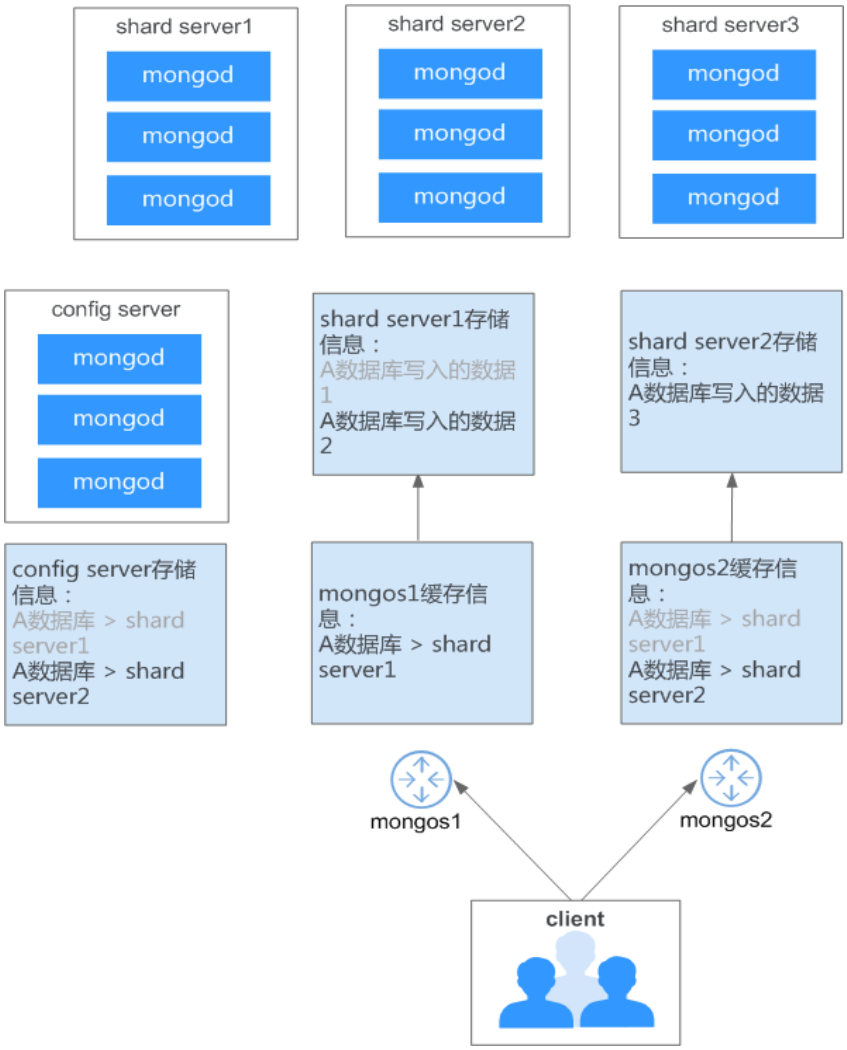
缓存刷新的范围以DB为单位。

使用场景

当未对数据进行分片时，若系统中存在多个dds mongos，通过不同的dds mongos进行数据访问时，可能出现不同dds mongos上本地缓存的路由信息不一致的情况。场景示例：

1. 通过mongos1创建A数据库，未开启分片。写入数据1后，数据1被全部分到shard server1上存储。然后，在mongos2上对数据进行查询。此时，mongos1和mongos2上，均存在缓存的A数据库的路由信息。
2. 通过mongos2执行了A数据库的删除操作。此时，config server和shard server1中的A数据库信息都被删掉。而mongos1无法识别数据1已经被删除。
3. 通过mongos1向A数据库中写入数据2时，由于存在缓存，所以无法识别A数据库已经被删除的场景。参照已经存在的路由信息，数据2被存储到shard server1上。然后，通过mongos2向A数据库中写入数据3时，由于能够识别出A数据库已经被删除，所以会在config server和shard server2中生成新的A数据库的信息。
4. 此时，mongos1和mongos2中缓存的路由信息不一致，关联不同的shard server，且仅能看到部分数据，导致数据异常。

图 10-2 mongos 缓存缺陷的场景



客户端通过不同mongos，所查询到的数据不同：

- mongos1：可以查到数据2，无法查询到数据3。
- mongos2：可以查询到数据3，无法查询到数据2。

规避建议

MongoDB官方建议：在每次删除数据库或集合后，在所有mongos节点上，通过命令**db.adminCommand("flushRouterConfig")**，刷新路由。

参考链接：

- **https://docs.mongodb.com/manual/reference/method/db.dropDatabase/index.html#replica-set-and-sharded-clusters**
- **https://jira.mongodb.org/browse/SERVER-17397**

其他规避建议：

- 对于集群模式，建议开启数据库的分片功能，再对其中的集合进行分片。
- 对于未开启分片功能的数据库。在删除数据库或集合之后，不建议创建同名的数据库或集合。

  如果因业务需求，需要创建同名的数据库或集合，请在删除数据库或集合之后，创建同名的数据库或集合之前，登录到所有的mongos节点上，执行刷新路由表的操作。

11 排查 DDS 实例 CPU 使用率高的问题

使用文档数据库服务时，如果您的CPU使用率很高或接近100%，会导致数据读写处理缓慢，从而影响业务正常运行。

本章节帮助您分析数据库正在执行的请求和数据库慢请求，经过分析优化后，使得数据库的查询相对合理，所有的请求都高效使用了索引，从而排查文档数据库服务CPU使用率高的问题。

分析 DDS 数据库正在执行的请求

1. 通过Mongo Shell连接DDS实例。

   开通公网访问的实例

   具体请参见：

   - **通过公网连接集群实例**
   - **通过公网连接副本集实例**
   - **通过公网连接单节点实例**

   未开通公网访问的实例

   具体请参见：

   - **通过内网连接集群实例**
   - **通过内网连接副本集实例**
   - **通过内网连接单节点实例**

2. 执行以下命令，查看数据库当前正在执行的操作。

   db.currentOp()

   回显如下：

```
{
	"raw" : {
		"shard0001" : {
			"inprog" : [
				{
					"desc" : "StatisticsCollector",
					"threadId" : "140323686905600",
					"active" : true,
					"opid" : 9037713,
					"op" : "none",
					"ns" : "",
					"query" : {

					},
```

```
                        "numYields" : 0,
                        "locks" : {

                        },
                        "waitingForLock" : false,
                        "lockStats" : {

                        }
                },
                {
                        "desc" : "conn2607",
                        "threadId" : "140323415066368",
                        "connectionId" : 2607,
                        "client" : "172.16.36.87:37804",
                        "appName" : "MongoDB Shell",
                        "active" : true,
                        "opid" : 9039588,
                        "secs_running" : 0,
                        "microsecs_running" : NumberLong(63),
                        "op" : "command",
                        "ns" : "admin.",
                        "query" : {
                                "currentOp" : 1
                    },
                        "numYields" : 0,
                        "locks" : {

                        },
                        "waitingForLock" : false,
                        "lockStats" : {

                        }
                }
        ],
        "ok" : 1
    },
  ...
}
```

说明

- client：发起请求的客户端。
- opid：操作的唯一标识符。
- secs_running：该操作已经执行的时间，单位：秒。如果该字段返回的值特别大，需要查看请求是否合理。
- microsecs_running：该操作已经执行的时间，单位：微秒。如果该字段返回的值特别大，需要查看请求是否合理。
- op：操作类型。通常是query、insert、update、delete、command中的一种。
- ns：操作目标集合。
- 其他参数详见**db.currentOp()**命令**官方文档**。

3. 根据命令执行结果，分析是否有异常耗时的请求正在执行。

   如果业务日常运行的CPU使用率不高，由于执行某一操作使得CPU使用率过高，导致业务运行缓慢，该场景下，您需要关注执行耗时久的请求。

   如果发现异常请求，您可以找到该请求对应的opid，执行**db.killOp(opid)**命令终止该请求。

分析 DDS 数据库的慢请求

文档数据库服务默认开启了慢请求Profiling ，系统自动将请求时间超过100ms的执行情况记录到对应数据库下的“system.profile”集合中。

1. 通过Mongo Shell连接DDS实例。

   开通公网访问的实例

   具体请参见：

   - **通过公网连接集群实例**
   - **通过公网连接副本集实例**
   - **通过公网连接单节点实例**

   未开通公网访问的实例

   具体请参见：

   - **通过内网连接集群实例**
   - **通过内网连接副本集实例**
   - **通过内网连接单节点实例**

2. 执行以下命令，进入指定数据库，以“test”为例。

   use test

3. 查看是否生成慢sql集合“system.profile”。

   show collections;

   - 回显中有“system.profile”，说明产生了慢SQL，继续执行下一步。

     ```
     mongos> show collections
     system.profile
     test
     ```

   - 回显中没有“system.profile”，说明未产生慢SQL，该数据库不涉及慢请求分析。

     ```
     mongos> show collections
     test
     ```

4. 查看数据下的慢请求日志。

   db.system.profile.find().pretty()

5. 分析慢请求日志，查找CPU使用率升高的原因。

   下面是某个慢请求日志示例，可查看到该请求进行了全表扫描，扫描了1561632个文档，没有通过索引进行查询。

   ```
   {
   	"op" : "query",
   	"ns" : "taiyiDatabase.taiyiTables$10002e",
   	"query" : {
   		"find" : "taiyiTables",
   		"filter" : {
   			"filed19" : NumberLong("852605039766")
   		},
   		"shardVersion" : [
   			Timestamp(1, 1048673),
   			ObjectId("5da43185267ad9c374a72fd5")
   		],
   		"chunkId" : "10002e"
   	},
   	"keysExamined" : 0,
   	"docsExamined" : 1561632,
   	"cursorExhausted" : true,
   	"numYield" : 12335,
   	"locks" : {
   		"Global" : {
   			"acquireCount" : {
   				"r" : NumberLong(24672)
   			}
   		},
   		"Database" : {
   			"acquireCount" : {
   ```

```
                        "r" : NumberLong(12336)
                    }
                },
                "Collection" : {
                    "acquireCount" : {
                        "r" : NumberLong(12336)
                    }
                }
            },
            "nreturned" : 0,
            "responseLength" : 157,
            "protocol" : "op_command",
            "millis" : 44480,
            "planSummary" : "COLLSCAN",
            "execStats" : {
                "stage" :
"SHARDING_FILTER",
                [3/1955]
                "nReturned" : 0,
                "executionTimeMillisEstimate" : 43701,
                "works" : 1561634,
                "advanced" : 0,
                "needTime" : 1561633,
                "needYield" : 0,
                "saveState" : 12335,
                "restoreState" : 12335,
                "isEOF" : 1,
                "invalidates" : 0,
                "chunkSkips" : 0,
                "inputStage" : {
                    "stage" : "COLLSCAN",
                    "filter" : {
                        "filed19" : {
                            "$eq" : NumberLong("852605039766")
                        }
                    },
                    "nReturned" : 0,
                    "executionTimeMillisEstimate" : 43590,
                    "works" : 1561634,
                    "advanced" : 0,
                    "needTime" : 1561633,
                    "needYield" : 0,
                    "saveState" : 12335,
                    "restoreState" : 12335,
                    "isEOF" : 1,
                    "invalidates" : 0,
                    "direction" : "forward",
                    "docsExamined" : 1561632
                }
            },
            "ts" : ISODate("2019-10-14T10:49:52.780Z"),
            "client" : "172.16.36.87",
            "appName" : "MongoDB Shell",
            "allUsers" : [
                {
                    "user" : "__system",
                    "db" : "local"
                }
            ],
           "user" : "__system@local"
}
```


在慢请求日志中，您需要重点关注以下关键字。

- 全集合（全表）扫描：COLLSCAN

  当一个操作请求（如query、update、delete）需要全表扫描时，将大量占用CPU资源。在查看慢请求日志时，发现COLLSCAN关键字，很可能是这些查询占用了CPU资源。

如果该类操作请求较为频繁，建议您对查询的字段建立索引进行优化。

- 全集合（全表）扫描：docsExamined

  通过查看参数“docsExamined”的值，可以查看一个查询扫描了多少文档。该值越大，请求的CPU使用率越高。

- 不合理的索引：IXSCAN、keysExamined

  说明

  索引不是越多越好，过多索引会影响写入和更新的性能。

  如果您的应用偏向于写操作，建立索引可能会降低写操作的性能。

  通过查看参数“keysExamined”的值，可以查看一个使用了索引的查询，扫描了多少条索引。该值越大，请求的CPU使用率越高。

  如果索引建立不太合理，或者匹配的结果很多。该场景下，即便使用了索引，请求的CPU使用率也不会降低很多，执行的速度也会很慢。

  示例：对于某个集合的数据，a字段的取值很少（只有1和2），而b字段的取值很多。

  ```
  { a: 1, b: 1 }
  { a: 1, b: 2 }
  { a: 1, b: 3 }
  ......
  { a: 1, b: 100000}
  { a: 2, b: 1 }
  { a: 2, b: 2 }
  { a: 2, b: 3 }
  ......
  { a: 1, y: 100000}
  ```

  如下所示，要实现 {a: 1, b: 2} 这样的查询。

  ```
  db.createIndex( {a: 1} )        效果不好，因为a相同取值太多
  db.createIndex( {a: 1, b: 1} )  效果不好，因为a相同取值太多
  db.createIndex( {b: 1 } )       效果好，因为b相同取值很少
  db.createIndex( {b: 1, a: 1 } ) 效果好，因为b相同取值少
  ```

  关于{a: 1}与{b: 1, a: 1}的区别，可参考**官方文档**。

- 大量数据排序：SORT、hasSortStage

  当查询请求中包含排序时，“system.profile”集合中的参数“hasSortStage”的值为“true”。如果排序无法通过索引实现，将在查询结果中进行排序。由于排序将占用大量CPU资源，该场景下，需要通过对经常排序的字段建立索引进行优化。

  当您在“system.profile”集合中发现SORT关键字时，可以考虑通过索引来优化排序。

其他操作如建立索引、Aggregation（遍历、查询、更新、排序等动作的组合）也可能占用大量CPU资源，但本质上也适用以上几种场景。更多Profiling的设置，请参见**官方文档**。

分析服务能力

经过前面数据库正在执行的请求和慢请求的分析和优化，所有的请求都使用了合理的索引，CPU的使用率相对趋于稳定。如果经过前面的分析排查，CPU使用率仍然居高不下，则可能是因为当前实例已达到性能瓶颈，不能满足业务需要，此时您可以通过如下方法解决。

1. 通过查看监控信息分析实例资源的使用情况，详情请参见**查看监控指标**。

2. 对DDS进行规格变更或者添加分片数量。具体操作请根据当前的实例类型参考如下文档。
   - **添加集群实例的节点数量**
   - **变更集群实例的CPU和内存**
   - **添加副本集实例的节点数量**
   - **变更副本集实例的CPU和内存**
   - **变更单节点实例的CPU和内存**

12 排查 DDS 实例内存占用较高的问题

使用文档数据库服务时，如果您的内存使用率很高或接近100%，不仅会导致业务请求响应缓慢，也会增大OOM风险，从而影响业务稳定运行。

本章节帮助您分析数据库连接数、慢请求和游标，经过分析优化后，使得数据库的查询相对合理，所有的请求都高效使用了索引，并规范使用游标，从而排查文档数据库服务内存使用率高的问题。如果经确认确实为业务增长导致的内存升高，建议及时进行**规格扩容**。

排查连接数

1. 查看**连接数占比**，总的连接数不宜超过当前实例能承受的最大连接数的80%。连接太多会导致内存和多线程上下文的开销增加，影响请求处理延时。
2. 建议配置连接池，一般情况建议连接池最大不要超过200，具体操作可参考**查询及限制连接数**。

分析慢日志

除了降低连接数以外，还需要注意单次请求的内存开销，尽量避免查询语句出现全表扫描、内存排序等。

1. 使用**慢日志功能**，查询当前实例产生的慢日志。
2. 分析慢日志，查找内存升高的原因：下面是某个慢请求日志示例，可查看到该请求进行了全表扫描，扫描了1561632个文档，没有通过索引进行查询。

```
{
    "op" : "query",
    "ns" : "taiyiDatabase.taiyiTables$10002e",
    "query" : {
        "find" : "taiyiTables",
        "filter" : {
            "filed19" : NumberLong("852605039766")
        },
        "shardVersion" : [
            Timestamp(1, 1048673),
            ObjectId("5da43185267ad9c374a72fd5")
        ],
        "chunkId" : "10002e"
    },
    "keysExamined" : 0,
    "docsExamined" : 1561632,
    "cursorExhausted" : true,
```

```
        "numYield" : 12335,
        "locks" : {
                "Global" : {
                        "acquireCount" : {
                                "r" : NumberLong(24672)
                        }
                },
                "Database" : {
                        "acquireCount" : {
                                "r" : NumberLong(12336)
                        }
                },
                "Collection" : {
                        "acquireCount" : {
                                "r" : NumberLong(12336)
                        }
                }
        },
        "nreturned" : 0,
        "responseLength" : 157,
        "protocol" : "op_command",
        "millis" : 44480,
        "planSummary" : "COLLSCAN",
        "execStats" : {
            "stage" :
"SHARDING_FILTER",
                                [3/1955]
                "nReturned" : 0,
                "executionTimeMillisEstimate" : 43701,
                "works" : 1561634,
                "advanced" : 0,
                "needTime" : 1561633,
                "needYield" : 0,
                "saveState" : 12335,
                "restoreState" : 12335,
                "isEOF" : 1,
                "invalidates" : 0,
                "chunkSkips" : 0,
                "inputStage" : {
                        "stage" : "COLLSCAN",
                        "filter" : {
                                "filed19" : {
                                        "$eq" : NumberLong("852605039766")
                                }
                        },
                        "nReturned" : 0,
                        "executionTimeMillisEstimate" : 43590,
                        "works" : 1561634,
                        "advanced" : 0,
                        "needTime" : 1561633,
                        "needYield" : 0,
                        "saveState" : 12335,
                        "restoreState" : 12335,
                        "isEOF" : 1,
                        "invalidates" : 0,
                        "direction" : "forward",
                        "docsExamined" : 1561632
                }
        },
        "ts" : ISODate("2019-10-14T10:49:52.780Z"),
        "client" : "172.16.36.87",
        "appName" : "MongoDB Shell",
```

```
    "allUsers" : [
        {
            "user" : "__system",
            "db" : "local"
        }
    ],
    "user" : "__system@local"
}
```

在慢请求日志中，您需要重点关注以下关键字。

- 全集合（全表）扫描：COLLSCAN
  - 当一个操作请求（如query、update、delete）需要全表扫描时，将大量占用内存资源。在查看慢请求日志时，发现COLLSCAN关键字，很可能是这些查询占用了内存资源。
  - 如果该类操作请求较为频繁，建议您对查询的字段建立索引进行优化。
- 全集合（全表）扫描：docsExamined
  - 通过查看参数“docsExamined”的值，可以查看一个查询扫描了多少文档。该值越大，请求的内存使用率越高。
- 不合理的索引：IXSCAN、keysExamined
- 大量数据排序：SORT、hasSortStage

当查询请求中包含排序时，参数“hasSortStage”的值为“true”。如果排序无法通过索引实现，将在查询结果中进行排序。由于排序将占用大量内存资源，该场景下，需要通过对经常排序的字段建立索引进行优化。

当您发现SORT关键字时，可以考虑通过索引来优化排序。

说明

索引不是越多越好，过多索引会影响写入和更新的性能。越建议参考ESR原则设计索引，以提高查询效率：

- 精确（Equal）匹配的字段放最前面。
- 排序（Sort）条件放中间。
- 范围（Range）匹配的字段放最后面。

检查游标

游标不规范的使用很容易造成内存升高并且长期不释放的情况，当客户端使用数据库的游标功能时，一定注意主动释放游标（游标的**官方说明**）。

1. 检查游标是否有被设置为不超时，默认情况下数据库会在10分钟后自动释放游标。Java driver给出的游标超时示例代码如下：
```
MongoCursor<Document> cursor = collection.find(query)
                .maxTime(10, TimeUnit.MINUTES)
                .iterator();
```
2. 在使用完游标后客户端是否有主动释放游标。Java driver中的释放示例如下：
```
cursor.close()
```
3. 如果已有noTimeout（不超时）游标，并且客户端已经无法释放，可**重启内存高的实例节点**以释放这些游标，但需要注意后续优化业务代码，尽量避免设置不超时游标，并在使用完游标后主动释放。

13 排查 DDS 实例连接数耗尽的问题

数据库连接数表示应用程序可以同时连接数据库的数量，与您应用程序或者网站能够支持的最大用户数没有关系。

- 对于集群实例，一般指客户端同mongos之间的连接数。
- 对于副本集实例，一般指客户端同Primary节点和Secondary节点之间的连接数。

问题现象

当DDS实例的连接数已满时，新发起的连接请求将无法被响应，从而导致实例连接失败。

- 使用Mongo Shell连接实例时，出现如下提示，表示当前连接池的连接数已满。

```
[root@623- ~]# mongo "mongodb://rwuser:@192.168.0.180:8635,192.168.0.50:8635/test?authSource=admin&replicaSet=replica"
MongoDB shell version v3.4.17
connecting to: mongodb://rwuser:623_Huawei@192.168.0.180:8635,192.168.0.50:8635/test?authSource=admin&replicaSet=replica
2019-10-16T17:43:45.203+0800 I NETWORK  [thread1] Starting new replica set monitor for replica/192.168.0.180:8635,192.168.0.50:8635
2019-10-16T17:43:45.205+0800 W NETWORK  [ReplicaSetMonitor-TaskExecutor-0] No primary detected for set replica
2019-10-16T17:43:45.205+0800 I NETWORK  [ReplicaSetMonitor-TaskExecutor-0] All nodes for set replica are down. This has happened for 1 checks in a row.
2019-10-16T17:43:45.708+0800 W NETWORK  [thread1] No primary detected for set replica
2019-10-16T17:43:45.708+0800 I NETWORK  [thread1] All nodes for set replica are down. This has happened for 2 checks in a row.
2019-10-16T17:43:46.210+0800 W NETWORK  [thread1] No primary detected for set replica
2019-10-16T17:43:46.210+0800 I NETWORK  [thread1] All nodes for set replica are down. This has happened for 3 checks in a row.
2019-10-16T17:43:46.712+0800 W NETWORK  [thread1] No primary detected for set replica
2019-10-16T17:43:46.712+0800 I NETWORK  [thread1] All nodes for set replica are down. This has happened for 4 checks in a row.
2019-10-16T17:43:47.215+0800 W NETWORK  [thread1] No primary detected for set replica
2019-10-16T17:43:47.215+0800 I NETWORK  [thread1] All nodes for set replica are down. This has happened for 5 checks in a row.
2019-10-16T17:43:47.717+0800 W NETWORK  [thread1] No primary detected for set replica
2019-10-16T17:43:47.717+0800 I NETWORK  [thread1] All nodes for set replica are down. This has happened for 6 checks in a row.
2019-10-16T17:43:48.218+0800 W NETWORK  [thread1] No primary detected for set replica
2019-10-16T17:43:48.218+0800 I NETWORK  [thread1] All nodes for set replica are down. This has happened for 7 checks in a row.
2019-10-16T17:43:48.721+0800 W NETWORK  [thread1] No primary detected for set replica
2019-10-16T17:43:48.721+0800 I NETWORK  [thread1] All nodes for set replica are down. This has happened for 8 checks in a row.
2019-10-16T17:43:49.222+0800 W NETWORK  [thread1] No primary detected for set replica
2019-10-16T17:43:49.222+0800 I NETWORK  [thread1] All nodes for set replica are down. This has happened for 9 checks in a row.
2019-10-16T17:43:49.724+0800 W NETWORK  [thread1] No primary detected for set replica
2019-10-16T17:43:49.724+0800 I NETWORK  [thread1] All nodes for set replica are down. This has happened for 10 checks in a row.
2019-10-16T17:43:50.226+0800 W NETWORK  [thread1] No primary detected for set replica
2019-10-16T17:43:50.226+0800 I NETWORK  [thread1] All nodes for set replica are down. This has happened for 11 checks in a row.
2019-10-16T17:43:50.727+0800 W NETWORK  [thread1] No primary detected for set replica
2019-10-16T17:43:51.230+0800 W NETWORK  [thread1] No primary detected for set replica
2019-10-16T17:43:51.731+0800 W NETWORK  [thread1] No primary detected for set replica
```

- 使用Python连接实例时，出现以下提示，表示当前连接池的连接数已满。

  pymongo.errors.ServerSelectionTimeoutError: connection closed, connection closed

- 查看**实例监控**，发现实例连接数确实被耗尽。

解决方法

经过上述步骤确认为连接数耗尽问题后，需要根据突发流量与长期业务两种情况分别处理。

- 突发流量造成的连接数满，考虑**重启实例或节点**释放当前连接。并同时排查客户端连接，是否存在大量重试请求，如果是大量重试请求引起的连接数耗尽，考虑修改客户端参数（修改重试逻辑，增大超时重试时长），从而避免连接数积压。

⚠ 注意

重启实例的操作会将实例的节点进行轮转重启，每个节点会有30秒左右的闪断，如果集合的数量较多（超过1万），闪断时间也会随之变长，重启前请做好业务安排并确保应用有重连机制。

- 针对长期业务导致的连接数耗尽，可以适当**上调最大连接数**（修改“**net.maxIncomingConnections**”的值，该参数修改后实时生效），保证每次调整幅度在20%以内，调整后**观察负载变化**。如果发现上调连接数后负载较高，说明实例负载已经达到瓶颈，请及时进行**规格扩容**。

14 创建用户并授权使用 DDS 只读权限

步骤 1：创建用户组并授权

用户组是用户的集合，IAM通过用户组功能实现用户的授权。您在IAM中创建的用户，需要加入特定用户组后，用户才具备用户组所拥有的权限。关于创建用户组并给用户组授权的方法，可以参考如下操作。

步骤1 使用注册的华为账号登录华为云，登录时请选择“华为账号登录”。

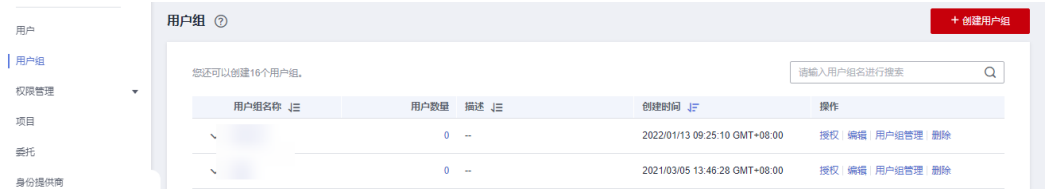
图 14-1 华为账号登录

步骤2 进入华为云控制台， 控制台页面中单击右上角的用户名，选择“统一身份认证”。

图 14-2 选择 IAM 服务

步骤3 在统一身份认证服务的左侧导航空格中，单击“用户组”>“创建用户组”。

图 14-3 用户组



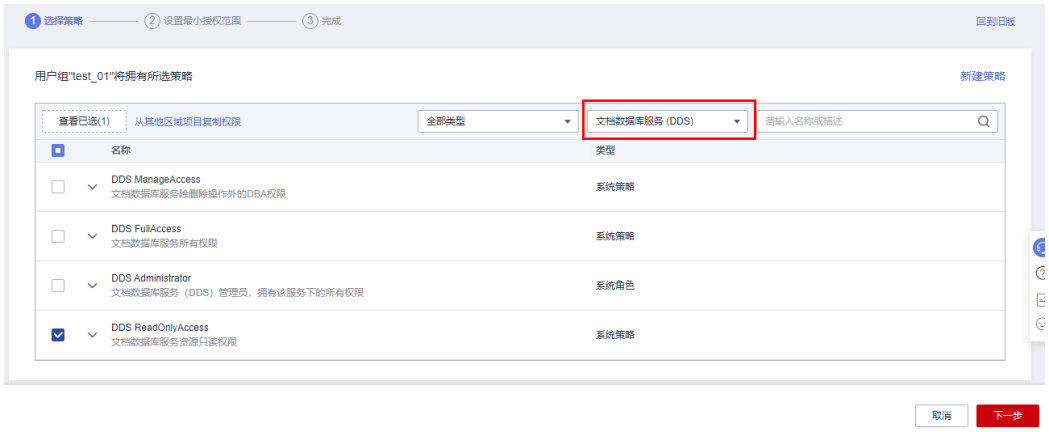
步骤4 在“创建用户组”界面，输入“用户组名称”，以“test_01”为例，并设置密码后，单击“确定”。

用户组创建完成，界面自动返回用户组列表，列表中显示新建的用户组。

步骤5 单击新建用户组右侧操作列的“授权”。

步骤6 筛选文档数据库服务(DDS)，选择“DDS ReadOnlyAccess”，单击“下一步”。

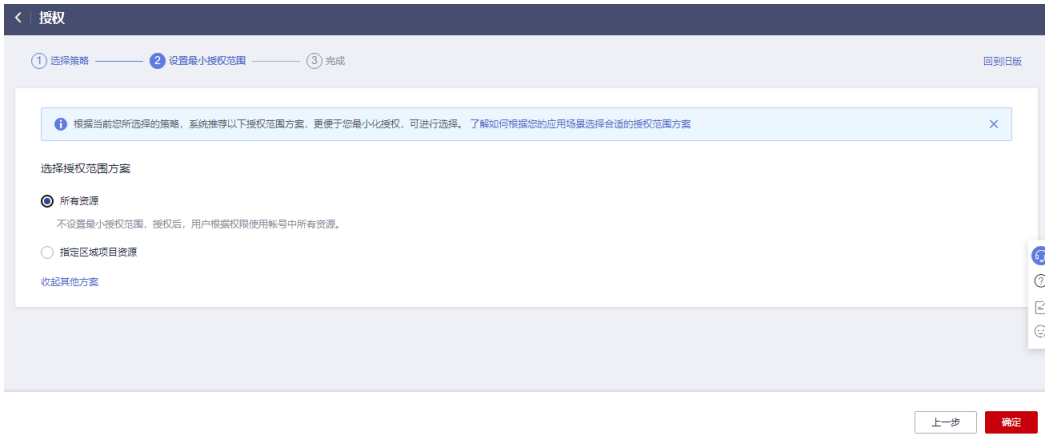
图 14-4 授权



步骤7 选择授权方案后，单击“确定”完成用户组授权。

- 所有资源。
- 指定区域项目资源。授权后，用户根据权限使用已选区域项目中的资源。

图 14-5 设置权限范围



----结束

步骤 2：创建 IAM 用户

IAM用户与企业中的实际员工或是应用程序相对应，有唯一的安全凭证，可以通过加入一个或多个用户组来获得用户组的权限。关于IAM用户的创建方式请参见如下步骤。

步骤1 在统一身份认证服务，左侧导航中，单击“用户 > 创建用户”。

步骤2 在“创建用户”页面填写“用户信息”。如需一次创建多个用户，可以单击“添加用户”进行批量创建，每次最多可创建10个用户。

图 14-6 创建用户

- 用户名：用户登录华为云的用户名，以“James”为例。
- 邮箱：IAM用户绑定的邮箱，仅“访问方式”选择“首次登录时设置”时必填，选择其他访问方式时选填。
- 手机号（选填）：IAM用户绑定的手机号。
- 描述（选填）：对用户的描述信息。

步骤3 在“创建用户”页面配置相关选项，完成后单击“下一步”。

图 14-7 配置

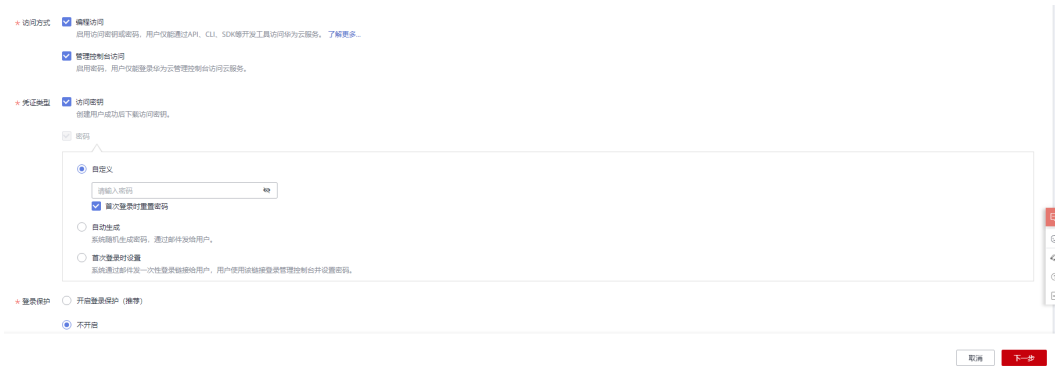
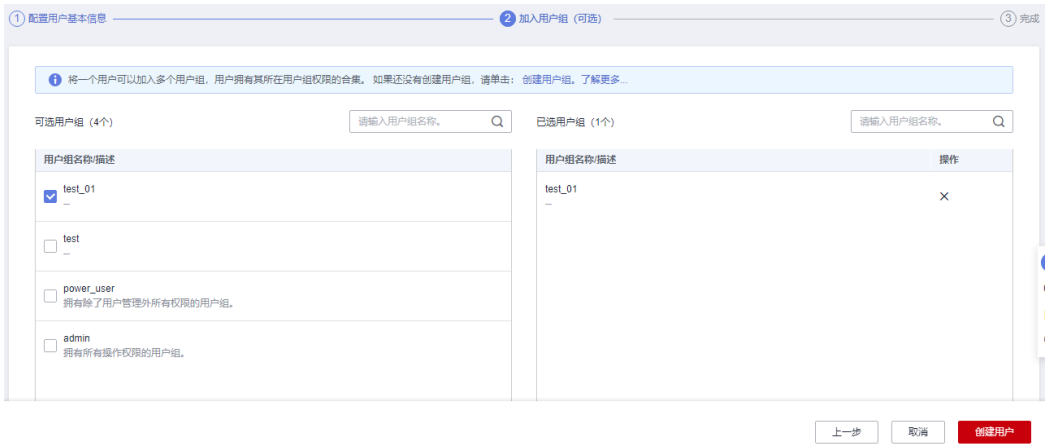


表 14-1 配置选项

配置项	说明
访问方式	• 编程访问：启用访问密钥或密码，用户仅能通过API、CLI、SDK等开发工具访问华为云服务。 • 管理控制台访问：启用密码，用户仅能登录华为云管理控制台访问云服务。
凭证类型	• 访问密钥：创建用户成功后下载访问密钥。 • 密码：当一次创建多个用户时，密码设置方式可选择“首次登录时设置”和“自定义”，不支持“自动生成”密码；当仅创建一个用户时，以上方式均可选择。
登录保护	为了您的账号安全，建议选择“开启登录保护”。后续如需开启或关闭登录保护，请参见：登录保护。

步骤4 将用户加入到用户组，选择步骤4创建的用户组，完成后单击“创建用户”。

图 14-8 创建用户



步骤5 IAM用户创建成功，用户列表中显示新创建的IAM用户。如果在凭证类型中勾选了“访问密钥”，可在此页面下载访问密钥，后续也可以在“我的凭证”中**管理访问密钥**。

图 14-9 查看创建结果

----结束

步骤 3：用户登录并验证权限

用户创建完成后，可以使用新用户的用户名及身份凭证登录华为云验证权限，即“DDS ReadOnlyAccess”权限。更多用户登录方法请参见**用户登录华为云方法**。

步骤1 在华为云登录页面，单击左下角的“IAM用户”。

图 14-10 IAM 用户登录

步骤2 在“IAM用户登录”页面，输入账号名、用户名及用户密码，使用新创建的用户登录。

- 账号名为该IAM用户所属华为账号的名称。
- 用户名和密码为账号在IAM创建用户时输入的用户名和密码。

如果登录失败，您可以联系您的账号主体，确认用户名及密码是否正确，或是重置用户名及密码，重置方法请参见**忘记IAM用户密码**。

步骤3 登录成功后，进入华为云控制台，请先切换至授权区域。

图 14-11 切换至授权区域

步骤4 在“服务列表”中选择文档数据库服务，进入DDS主界面，单击右上角“购买数据库实例”，若提示权限不足，表示“DDS ReadOnlyAccess”已生效。

步骤5 在“服务列表”中选择除文档数据库服务外的任一服务，如“弹性云服务器”，若提示权限不足，表示“DDS ReadOnlyAccess”已生效。

----结束

15 合理使用 DDL(Data Definition Languages)语句

数据库模式定义语言DDL(Data Definition Language)，是用于描述数据库中存储实体的语言，例如创建、修改和删除数据库、集合的结构。

DDS中常见的DDL包括：

- 创建集合（createCollection）：用于在指定的数据库中创建一个新的集合。
- 删除集合（drop）：用于删除指定的集合。
- 创建索引（createIndex）：用于在集合中创建一个或多个索引，以提高查询效率。
- 删除索引（dropIndex）：用于从集合中删除指定的索引。
- 修改集合的分片规则（shardCollection）：使用sh.shardCollection命令修改集合的分片规则，将集合分布在不同的分片中。
- 查看集合的元数据（collection.stats）：用于查看指定集合的元数据信息，如文档数量、存储大小等。
- 查看数据库的元数据（db.stats）：用于查看指定数据库的元数据信息，如集合数量、存储。

使用 DDL 的注意事项

- 创建索引、删除索引、删除集合等操作会消耗大量IO/计算资源，需要在业务的低峰期执行，避免影响业务。
- 避免多个DDL操作同时执行，可能会引起阻塞导致执行失败。例如：创建索引和删除集合不要同时执行。
- 创建索引时，只创建必要的索引，避免创建多余索引导致存储空间浪费。例如：不要基于复合索引的前缀字段再创建索引。
- 创建索引时，需要创建后台索引，即：db.<collection_name>.createIndex({ <field_name>: <index_type> }, { background: true })，注意，后台索引不会阻塞其他业务，但是仍然会消耗大量IO资源。
- 在删除索引时，需要先进行性能测试，确保删除的索引不会影响查询性能。
- 创建集合时，应避免创建过多集合，集合数量太多会产生大量元数据并消耗额外资源，且非常难以维护。例如：应避免以日期命名集合并每天创建一个新的集合，考虑将日期作为字段存储到同一集合中。

- 删除集合前，需要谨慎确认集合名称，因为集合删除后无法直接恢复数据，建议先备份重要数据。
- 遇到全表删除的场景，避免使用不带过滤条件的remove、delete命令进行删除，尽量使用db.<collection_name>.drop()删除集合来替代。remove、delete若有查询条件，也必须创建对应的索引。
- 对于DDS集群实例，如果某个集合的存储量很高，需要对该集合设置合理的数据分片，详情请参考**通过设置数据分片提升性能**。
- 对于数据库和集合的元数据信息（如文档数量、存储大小等），可以使用db.stats()和db.<collection_name>.stats()方法进行查看。这些信息对于性能优化和容量规划非常重要。

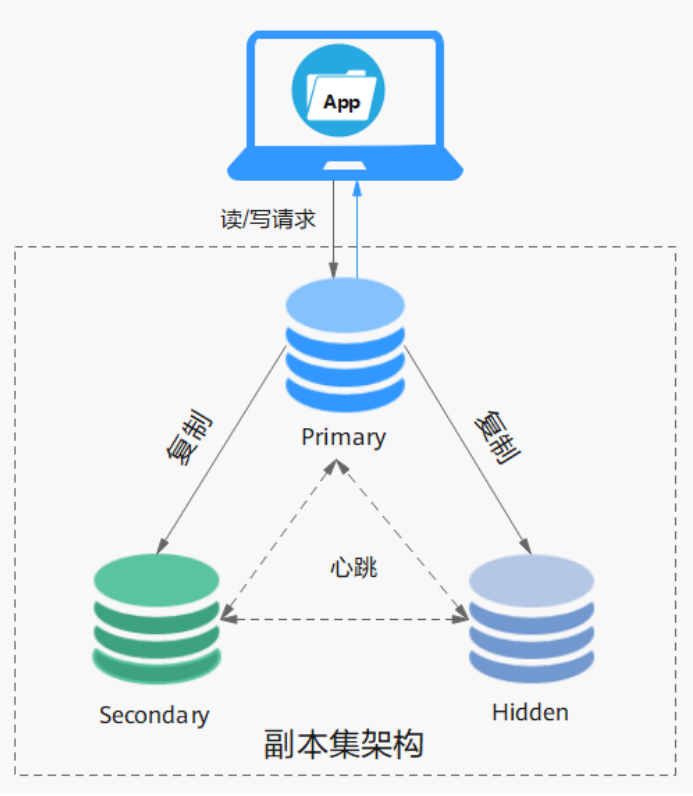
16 DDS 节点脱节原理和说明

副本集架构由主节点、备节点和隐藏节点组成，DDS自动搭建三节点的副本集供用户使用，节点之间数据自动同步，保证数据的高可靠性。对于需要保证高可用的中小型业务系统，推荐使用副本集。

- 主节点：即Primary节点，用于读写请求。
- 备节点：即Secondary节点，用于读请求。
- 隐藏节点：即Hidden节点，用于业务数据的备份。

用户可以直接操作主节点和备节点。若主节点故障，系统自动分配新的主节点。副本集架构如下图所示。

图 16-1 三节点副本集架构



DDS只能在主节点写入数据，主节点写入数据时，会同时生成oplog，Secondary和Hidden通过从主节点读取oplog进行回放，达到数据的最终一致。

oplog的存储量由oplogSize（默认磁盘容量的10%）决定。

- 主备时延是如何产生的?

  如果主节点写入速度太快，超过备节点oplog的读取回放速度，此时就产生了主备时延。

- 何时会脱节?

  因为oplog存储容量有限，如果到达容量上限，就会淘汰掉最早的oplog。备节点读取oplog，每次都会记录最后一条读取的oplog，如果主备时延达到一定程度，备节点发现上次回放的oplog点位已经被淘汰掉，此时备节点就无法继续读取oplog，说明备节点已经脱节。

- 如何有效防止备节点脱节?
  - 业务需要设置majority级别的writeConcern写入数据，表示将数据写入到大多数节点。
  - 适当增加oplog的存储空间上限，如需修改，可以通过控制台界面修改参数值oplogSizePercent，具体步骤请参见**修改DDS实例参数**。
  - 将创建索引等耗时较长的DDL操作，以及数据备份操作放在业务低峰期执行，同时尽可能避免突发大量增删改操作。

    说明

    如果没有以majority级别的writeConcern写入数据，发生主备倒换时，未同步到备节点的数据有丢失风险。

17 避免 hideIndex 导致游标失效

失效场景

在 DDS 中，某些索引操作可能导致游标失效，DDS这些行为和社区版MongoDB行为一致。以下是几个常见的例子及其解释：

hideIndex 导致游标失效

原因：

hideIndex操作改变了索引的元数据，从而使现有游标失效。

示例代码：

```
(function() {
    "use strict";

    let collName = "test_coll";
    let coll = db.getCollection(collName);
    coll.drop();

    // 创建索引，并插入数据，
    coll.createIndex({x: 1}, {background: false});
    for (var i = 0; i < 1000; ++i) {
        coll.insert({x: i});
    }

    // 获取游标
    let cursor = coll.find({x: {$gte: 1}});

    // 使用游标，但是游标并没有使用完
    for (var i = 0; i < 101; ++i) {
        let doc = cursor.next();
        print(tojson(doc));
    }

    // 在游标使用完之前隐藏索引
    coll.hideIndex("x_1");

    // 遍历游标
    cursor.forEach((doc) => {
        // 游标会报错：
        // "errmsg" : "definition of index 'x_1' changed"
        // "codeName" : "QueryPlanKilled"
```

```
        print(tojson(doc));
    });
})();
```

结果：

游标失效，可能抛出错误。

```
2024-12-24T16:26:42.445+0800 E QUERY    [js] Error: getMore command failed: {
    "ok" : 0,
    "errmsg" : "definition of index 'x_1' changed",
    "code" : 175,
    "codeName" : "QueryPlanKilled"
} :
```

dropIndex 导致游标失效

原因：

删除索引后，DDS需要重新计算查询计划，从而使现有游标失效。

示例代码：

```
(function() {
    "use strict";

    let collName = "test_coll";
    let coll = db.getCollection(collName);
    coll.drop();

    // 创建索引，并插入数据，
    coll.createIndex({x: 1}, {background: false});
    for (var i = 0; i < 1000; ++i) {
        coll.insert({x: i});
    }

    // 获取游标
    let cursor = coll.find({x: {$gte: 1}});

    // 使用游标，但是游标并没有使用完
    for (var i = 0; i < 101; ++i) {
        let doc = cursor.next();
        print(tojson(doc));
    }

    // 在游标使用完之前删除索引
    coll.dropIndex("x_1");

    // 遍历游标
    cursor.forEach((doc) => {
        // 游标会报错：
        // "errmsg" : "index 'x_1' dropped",
        // "codeName" : "QueryPlanKilled"
        print(tojson(doc));
    });
})();
```

结果：

游标失效，因为删除索引会导致查询计划需要重新计算。

```
2024-12-25T10:34:29.948+0800 E QUERY    [js] Error: getMore command failed: {
    "ok" : 0,
```

```
    "errmsg" : "index 'x_1' dropped",
    "code" : 175,
    "codeName" : "QueryPlanKilled"
}
```

renameCollection 导致游标失效

原因：

修改集合结构renameCollection也会导致游标失效。

示例代码：

```
(function() {
    "use strict";

    let collName = "test_coll";
    let coll = db.getCollection(collName);
    coll.drop();

    // 创建索引，并插入数据，
    coll.createIndex({x: 1}, {background: false});
    for (var i = 0; i < 1000; ++i) {
        coll.insert({x: i});
    }

    // 获取游标
    let cursor = coll.find({x: {$gte: 1}});

    // 使用游标，但是游标并没有使用完
    for (var i = 0; i < 101; ++i) {
        let doc = cursor.next();
        print(tojson(doc));
    }

    // 在游标使用完之前重命名集合
    coll.renameCollection("test_coll_reaname");

    // 遍历游标
    cursor.forEach((doc) => {
        // 游标会报错：
        // "errmsg" : "collection dropped between getMore calls",
        // "codeName" : "OperationFailed"
        print(tojson(doc));
    });
})();
```

结果：

游标失效，因为集合的名称改变，游标失去了上下文。

```
2024-12-25T10:39:28.624+0800 E QUERY    [js] Error: getMore command failed: {
    "ok" : 0,
    "errmsg" : "collection dropped between getMore calls",
    "code" : 96,
    "codeName" : "OperationFailed"
} :
```

dropCollection 导致游标失效

原因：

删除集合游标失去了上下文，会导致游标失效。

示例代码：

```
(function() {
    "use strict";

    let collName = "test_coll";
    let coll = db.getCollection(collName);
    coll.drop();

    // 创建索引，并插入数据，
    coll.createIndex({x: 1}, {background: false});
    for (var i = 0; i < 1000; ++i) {
        coll.insert({x: i});
    }

    // 获取游标
    let cursor = coll.find({x: {$gte: 1}});

    // 使用游标，但是游标并没有使用完
    for (var i = 0; i < 101; ++i) {
        let doc = cursor.next();
        print(tojson(doc));
    }

    // 在游标使用完之前删除集合
    coll.drop();

    // 遍历游标
    cursor.forEach((doc) => {
        // 游标会报错：
        // "errmsg" : "collection dropped between getMore calls",
        // "codeName" : "OperationFailed"
        print(tojson(doc));
    });
})();
```

结果：

游标失效，因为集合已经被删除，游标失去了上下文。

```
2024-12-25T10:40:33.737+0800 E QUERY    [js] Error: getMore command failed: {
    "ok" : 0,
    "errmsg" : "collection dropped between getMore calls",
    "code" : 96,
    "codeName" : "OperationFailed"
} :
```

解决方法

- 在索引操作之前处理游标： 确保在 hideIndex 或 dropIndex 之前使用完游标。
- 重新获取游标： 如果索引发生变化，重新执行查询以获取新的游标。
- 提前计划： 避免在长时间查询中执行索引操作。

示例代码：

```
(function() {
    "use strict";

    let collName = "test_coll";
```

```
    let coll = db.getCollection(collName);
    coll.drop();

    // 创建索引，并插入数据，
    coll.createIndex({x: 1}, {background: false});
    for (var i = 0; i < 1000; ++i) {
        coll.insert({x: i});
    }

    // 获取游标
    let cursor = coll.find({x: {$gte: 1}});

    // 使用游标，但是游标并没有使用完
    for (var i = 0; i < 101; ++i) {
        let doc = cursor.next();
        print(tojson(doc));
    }

    // 在游标使用完之前隐藏索引
    coll.hideIndex("x_1");

    // 遍历游标
    // cursor.forEach((doc) => {
        // 游标会报错：
        // "errmsg" : "definition of index 'x_1' changed"
        // "codeName" : "QueryPlanKilled"
        // print(tojson(doc));
    // });

    // 重新获取游标
    cursor = coll.find({x: {$gte: 1}});
    cursor.forEach((doc) => {
        print(tojson(doc));
    });
})();
```

18 使用 DDS 存储和分析日志数据

在现代应用的运维和分析中，日志数据扮演着至关重要的角色。不仅可以帮助监控应用的健康状况，还提供了宝贵的数据以方便洞察，用于优化用户体验和业务决策。然而，随着数据量的激增，传统的日志存储和分析方法逐渐显得力不从心。本文介绍如何使用DDS（Document Database Service）高效存储和分析日志数据，以挖掘数据的最大价值。DDS 4.2及其以上版本采用RocksDB作为底层存储引擎，对于日志存储与分析这种写多读少的场景有较好的性能表现。

通过合理设计日志存储结构、优化写入和查询策略、采用数据分片原理，DDS能够高效地存储和分析海量日志数据，为应用运维和业务分析提供强有力的支持。DDS不仅提供了强大的数据存储能力，还通过灵活的配置和优化策略，帮助用户在数据爆炸的时代中，轻松应对挑战，挖掘数据的无限价值。

日志数据的存储策略

在物联网（Internet of Things，简称IoT）领域，设备的日志数据扮演着至关重要的角色，不仅能帮助监控设备的运行状态，还提供了设备使用模式和故障预测的宝贵信息。例如，一个智能家庭安全系统的日志记录，可以包含设备ID、时间戳、事件类型（如“门锁开启”、“运动检测”）、设备状态、以及可能的错误代码等信息。

- 日志数据示例

  一个典型的IoT设备日志条目如下所示：

  ```
  DeviceID: 001, Timestamp: 2023-04-05T14:30:00Z, Event: DoorLockOpened, DeviceStatus: Active, Error: None
  ```

- 存储模式优化

  在数据存储到DDS时，将上述日志转换为结构化的文档，提取关键字段，例如：

  ```
  {   _id: ObjectId('5f442120eb03305789000000'),
      device_id: "001",
      timestamp: ISODate("2023-04-05T14:30:00Z"),
      event: "DoorLockOpened",
      device_status: "Active",
      error: "None"
  }
  ```

  在存储时，根据分析需求过滤无关字段，以节省存储空间。例如，如果分析不关心的设备的错误状态，可以省略error字段。

日志写入与性能优化

使用DDS副本集存储数据，可以提供数据冗余与高可靠性的存储能力。为了支持高并发的日志写入，可以定制writeConcern参数。例如，使用{w: 0}可达到最高写入吞吐

量，但同时牺牲了数据持久性。对于关键设备日志，推荐使用更安全的级别，如{w: 1}或{w: "majority"}。批量写入多条日志的同时也能显著提升效率。

```
replica:PRIMARY> log = {
...     "device_id": "001",
...     "timestamp": ISODate("2023-04-05T14:30:00Z"),
...     "event": "DoorLockOpened",
...     "device_status": "Active",
...     "error": "None" ... };
// 以{w: 0}写入
replica:PRIMARY> db.getSiblingDB("iot_logs").events.insert(log, {w: 0})
WriteResult({ "nInserted" : 1 })
// 以{w: 1}写入
replica:PRIMARY> db.getSiblingDB("iot_logs").events.insert(log, {w: 1})
WriteResult({ "nInserted" : 1 })
// 以{w: "majority"}写入
replica:PRIMARY> db.getSiblingDB("iot_logs").events.insert(log, {w: "majority"})
WriteResult({ "nInserted" : 1 })
// 单次批量写入：
replica:PRIMARY> logs = [
... { "device_id": "002", "timestamp": ISODate("2023-04-06T09:45:00Z"), "event": "MotionDetected",
"device_status": "Active", "error": "None"},
... { "device_id": "003", "timestamp": ISODate("2023-04-07T16:15:00Z"), "event": "TemperatureAlarm",
"device_status": "Active", "error": "None"},
... {"device_id": "004", "timestamp": ISODate("2023-04-08T20:30:00Z"), "event": "WindowOpened",
"device_status": "Active", "error": "None"}
... ]
replica:PRIMARY> db.getSiblingDB("iot_logs").events.insertMany(logs, {w: 0})
```

日志查询与分析

- 日志查询示例

  查询特定设备在特定时间范围内的所有事件。

  ```
  db.getSiblingDB("iot_logs").events.find({"device_id": "001", "timestamp": {"$gte":
  ISODate("2023-04-05T00:00:00Z"), "$lt": ISODate("2023-04-06T00:00:00Z")}})
  ```

- 建立索引以提高查询效率

  针对device_id和timestamp字段建立索引可以加速查询上述场景。

  ```
  replica:PRIMARY> db.getSiblingDB("iot_logs").events.createIndex({"device_id": 1, "timestamp": 1})
  ```

- 复杂分析

  利用DDS的聚合框架可以进行更复杂的查询分析。具体详情请参考**聚合框架**。

数据分片与扩展

随着日志数量的激增，日志数据量呈指数级增长，单一数据库节点往往难以应对海量数据的存储和查询需求。DDS（Document Database Service）通过数据分片（Sharding）技术，提供了水平扩展能力，能够有效分担数据存储和查询压力，确保系统的高可用性和高性能。

- 数据分片原理

  数据分片是将数据集分割成多个部分，分别存储在不同的数据库节点（Shard）上。每个Shard存储数据集的一部分，通过Shard Key来确定数据的分布。Shard Key的选择至关重要，它决定了数据的分布模式，影响着写入和查询的性能。

- Shard Key选择策略

  选择Shard Key时，应考虑以下几点：

  - **均匀分布**：Shard Key应确保数据在Shard之间均匀分布，避免热点问题。
  - **查询模式**：Shard Key应与常见的查询模式相匹配，以提高查询效率。
  - **数据访问模式**：如果数据访问模式是基于时间的，可以考虑使用时间戳或基于时间的字段作为Shard Key。

详细可参考文档**通过设置数据分片提升性能**。

- 基于设备ID的分片

  假设有大量的IoT设备，每个设备产生大量日志。可以选择device_id作为Shard Key，这样可以确保每个设备的日志数据存储在不同的Shard上，实现数据的均匀分布。具体示例如下：

```
db.getSiblingDB("iot_logs").events.createIndex({"device_id": 1})
sh.enableSharding("iot_logs")
sh.shardCollection("iot_logs.events", {"device_id": 1 })
```

  说明

  - 在创建分片集合时，确保Shard Key字段在数据中具有足够的唯一性，以实现数据的均匀分布。
  - Shard Key的选择应考虑到数据的访问模式，以优化查询性能。
  - 分片集群的管理，如添加或删除Shard，应根据数据增长和查询需求进行调整。

自动删除过期数据

- **TTL索引**：自动删除过期文档，如设置30天后自动删除。

```
db.eventlog.createIndex( { "timestamp": 1 }, { expireAfterSeconds: 30 * 24 * 60 * 60 } )
```

- **Capped集合**：限制集合大小，自动删除最旧文档。固定大小集合是大小固定的集合，根据插入顺序插入和检索文档。固定大小集合的工作方式与循环缓冲区类似：一旦一个集合填满了分配的空间，它就会通过覆盖集合中最旧的文档来为新文档腾出空间。

```
// 创建固定集合需要在创建集合的时候指定
db.getSiblingDB("iot_logs").dropDatabase()
db.getSiblingDB("iot_logs").createCollection( "events", { capped: true, size: 5242880 } )
```

- **定期归档**：按月归档日志数据，便于历史数据管理和查询。

```
db.getSiblingDB("iot_logs").events.renameCollection("events202301")
```

须知

单个实例中，数据库的总的个数不要超过200个，总的集合个数不要超过500个。集合数量过多会导致内存压力变高，并且集合数量多会导致重启以及主备倒换性能变差，影响紧急情况下的高可用性能。建议定期删除不需要的集合。

19 DDS 查询计划及查询重规划

DDS查询计划是决定查询如何执行的详细过程。DDS查询优化器会根据查询的条件、数据分布、索引信息等选择一个执行计划，以确保查询的执行效率。然而，在某些情况下，DDS会重新评估并生成新的查询计划，即查询重规划。用户理解何时发生查询重规划以及如何应对这一过程，这将有助于提升数据库性能并有效减少资源浪费。

以下将介绍DDS查询计划的工作原理，查询重规划的触发场景，以及处理和优化查询重规划的最佳实践。

查询计划的概述

查询计划是在执行查询时，DDS查询优化器选择的执行路径。这个执行路径定义了如何扫描数据，是否使用索引，以及如何处理查询的每个阶段。DDS通过以下几个步骤生成查询计划：

1. 解析查询：DDS解析查询条件，识别字段和操作符。
2. 选择索引：基于查询的条件，DDS会检查可用的索引，并选择最适合的索引来加速查询。
3. 评估执行路径：DDS会基于数据量、字段选择性、索引类型等评估可能的执行路径，最终选择最优的执行路径生成新的查询计划。
4. 执行查询：DDS按照选定的查询计划执行查询并返回结果。

查询重规划的概述

查询重规划（Query Re-Planning）是指DDS在执行查询过程中，基于数据变化或查询结构的变化，重新评估并生成一个新的查询计划。查询重规划发生时，DDS会放弃之前选择的查询计划，采用新的执行路径。

查询重规划的主要目的是在数据环境变化时，确保查询能够适应新的数据分布、索引变化和负载变化，从而维持查询性能的稳定。

查询重规划的触发场景

以下是一些常见的查询重规划场景：

- 索引变化

  新增或删除索引：当某个字段的索引发生变化（如新增索引或删除索引）时，DDS可能会重新规划查询执行计划。查询可能会重新评估现有索引的有效性，进而选择新的索引，或决定使用全表扫描。

- 数据分布变化

  如果数据分布发生了变化（例如某个字段的选择性发生显著变化），DDS可能会重新评估查询计划，以确保选择最佳的执行路径。例如，某个字段变得更加稀疏或密集，导致索引的效益变化。

- 查询条件变化

  当查询条件变化时，DDS可能需要重新生成查询计划。例如，某些查询添加了新的筛选条件，或者查询条件中使用了不同的操作符（如从$eq改为$in）。

- 聚合管道中的变化

  在聚合操作中，如果数据量、字段或管道顺序发生变化，DDS可能会重新评估聚合管道的执行计划。尤其是当管道阶段的数据过滤条件发生变化时，DDS可能会重新选择执行路径。

- 环境资源压力变化

  在实例负载发生变化（如节点资源耗尽、硬盘空间不足、或网络延迟增加）时，DDS可能会重新调整查询计划，以便在当前资源条件下优化查询性能。

- 查询缓存失效

  DDS会将之前执行的最优的查询计划缓存，DDS在某些情况下可能会使用查询缓存。随着查询缓存的失效，查询计划可能会被重新生成。

查询重规划的影响

发生重规划时，会在慢日志中看到关键字：replanned:1，这表示数据库无法针对当前查询模式的查询条件提供始终有效的计划。

频繁的查询重新规划（Query Re-Planning）可能带来以下影响：

- **CPU开销增加**：频繁的重新规划会消耗更多CPU资源，影响整体性能。
- **内存消耗增加**：频繁重新规划会增加内存使用，可能影响其他操作。
- **查询延迟增加**：每次重新规划都会增加查询执行时间，导致响应变慢。

查询重规划的处理建议

- 定期监控查询计划

  建议定期监控查询的执行计划，特别是在系统发生变动后，如添加索引、新增数据或调整查询结构。通过以下方式实现：

  - 使用explain()方法分析查询的执行计划。
  - 定期检查慢日志，了解可能的查询计划变化。

- 观察查询计划变化的原因

  当查询重规划发生时，首先要确定是哪些因素导致了查询计划的变化。根据以下几个方面进行排查：

  - 索引变化：使用db.collection.getIndexes()检查现有索引，确认是否有索引的新增或删除。
  - 数据分布：通过统计信息来检查数据是否发生了显著变化。例如使用db.collection.stats()查看集合的统计信息，如文档数、数据大小和索引大小。
  - 查询条件：检查查询条件是否发生了变化，是否增加了新的字段或者修改了查询操作符。

- 针对查询重规划进行优化

以下是一些优化查询重规划的建议：

- 【推荐】确保数据库环境有足够的资源（CPU、内存、磁盘等）来处理负载，减少因资源限制导致的查询重规划，升级实例规格以缓解数据库负载压力，具体操作请参考文档**变更实例的CPU和内存规格**。
- 【推荐】预估数据分布：在设计索引时，可以使用数据分布的预测来选择合适的字段进行索引，以避免数据分布变化时引发频繁的查询重规划。
- 【推荐】使用稳定的索引：确保为常用查询提供稳定且合适的索引。特别是在数据量较大时，索引可以有效减少查询重规划的发生。
- 【推荐】尽量避免频繁变化的查询结构：查询条件的频繁变动可能会导致DDS经常重新生成查询计划。建议根据业务需求，尽量确保查询条件的一致性。
- 【推荐】聚合优化：如果您的查询涉及复杂的聚合管道，建议定期分析管道的执行计划并进行优化。可以通过$match在聚合管道的早期阶段过滤数据，减少后续阶段的数据处理量。
- 对发生replan的查询条件使用hint()来指定索引，对查询调用此方法可覆盖DDS的默认索引选择和查询优化过程。示例：

  ```
  db.users.find({gender:"M"},{user_name:1,_id:0}).hint({gender:1,user_name:1})
  ```

- 在发生replan的查询条件使用索引过滤器来限制使用的索引，索引过滤器会覆盖查询计划器正常选择查询计划的行为，当一个查询同时存在Hint和索引过滤器时，索引过滤器会覆盖指定的Hint值，因此，您应当谨慎使用索引过滤器。以下示例为orders集合创建了一个索引筛选器。该筛选器适用于其谓词为item字段的等值匹配的查询，其中仅投影quantity字段，并指定按order_date升序排序。

  ```
  db.runCommand(
    {
      planCacheSetFilter: "orders",
      query: { item: "ABC" },
      projection: { quantity: 1, _id: 0 },
      sort: { order_date: 1 },
      indexes: [
        { item: 1, order_date: 1 , quantity: 1 }
      ]
    }
  )
  ```

  对于计划缓存查询结构，查询优化器将仅考虑使用索引{ item: 1, order_date: 1, quantity: 1 }的索引计划。

评估重规划并采取措施

查询重规划可能会导致执行计划的变化，进而影响查询的性能。在重规划发生时，需要评估查询性能的变化，检查是否有提升或下降，并相应地采取措施：

- 性能下降：如果发现查询性能因查询重规划而下降，可以通过添加或调整索引来改善性能。
- 性能提升：如果重规划后的查询执行更高效，可以继续保持当前的执行计划，监控是否有其他潜在的性能瓶颈。

20 DDS 事务与读写关注

事务

- 事务简介

  在DDS中，对单个文档的操作具有原子性。由于您可以使用嵌入式文档和数组在单个文档结构中捕获数据之间的关系，而不是在多个文档和集合中进行规范化，因此这种单文档原子性避免了许多实际使用案例中对多文档事务的需求。

  对于需要多个文档更新的原子性或多个文档读取之间的一致性的适用场景：从4.0版本开始，DDS提供了针对副本集执行多文档事务的能力。

  多文档事务可以跨多个操作、集合、数据库和文档使用。多文档事务提供了一个“全或无”的命题。当一个事务提交时，所有在事务中做的数据更改都会被保存。如果事务中的任何操作失败，事务将中止，并且在事务中进行的所有数据更改都将被丢弃，而不会变得可见。在事务提交之前，事务中的写操作在事务之外是不可见的。

  须知

  在大多数情况下，多文档事务比单文档写入要付出更大的性能代价，多文档事务的可用性不应该取代有效的模式设计。对于许多场景，非规范化数据模型（嵌入文档和数组）将继续适合您的数据和用例。也就是说，对于许多场景，适当地对数据进行建模将最大限度地减少对多文档事务的需求。

- 事务API

  下面的mongo shell代码示例展示了使用DDS事务的关键API：

```
// Runs the txnFunc and retries if TransientTransactionError encountered
function runTransactionWithRetry(txnFunc, session) {
    while (true) {
       try {
           txnFunc(session);  // performs transaction
           break;
       } catch (error) {
           // If transient error, retry the whole transaction
           if ( error.hasOwnProperty("errorLabels") &&
error.errorLabels.includes("TransientTransactionError")  ) {
               print("TransientTransactionError, retrying transaction ...");
               continue;
           } else {
               throw error;
           }
       }
```

```
    }
}

// Retries commit if UnknownTransactionCommitResult encountered
function commitWithRetry(session) {
    while (true) {
        try {
            session.commitTransaction(); // Uses write concern set at transaction start.
            print("Transaction committed.");
            break;
        } catch (error) {
            // Can retry commit
            if (error.hasOwnProperty("errorLabels") &&
error.errorLabels.includes("UnknownTransactionCommitResult") ) {
                print("UnknownTransactionCommitResult, retrying commit operation ...");
                continue;
            } else {
                print("Error during commit ...");
                throw error;
            }
        }
    }
}

// Updates two collections in a transactions
function updateEmployeeInfo(session) {
    employeesCollection = session.getDatabase("hr").employees;
    eventsCollection = session.getDatabase("reporting").events;

    session.startTransaction( { readConcern: { level: "snapshot" }, writeConcern: { w: "majority" } } );

    try{
        employeesCollection.updateOne( { employee: 3 }, { $set: { status: "Inactive" } } );
        eventsCollection.insertOne( { employee: 3, status: { new: "Inactive", old: "Active" } } );
        print(tojson(employeesCollection.find({ employee: 3 }).toArray()));
        print(tojson(eventsCollection.find({ employee: 3 }).toArray()));
        print("countDocuments = " + eventsCollection.countDocuments({}));
    } catch (error) {
        print("Caught exception during transaction, aborting.", error);
        session.abortTransaction();
        throw error;
    }

    commitWithRetry(session);
}

// insert data
employeesCollection = db.getSiblingDB("hr").employees; eventsCollection =
db.getSiblingDB("reporting").events;
employeesCollection.drop();
eventsCollection.drop();
for (var i = 0; i < 10; ++i) {
    employeesCollection.insertOne({employee: i});
    eventsCollection.insertOne({employee: i});
}

// Start a session.
session = db.getMongo().startSession( { readPreference: { mode: "primary" } } );
try{
   runTransactionWithRetry(updateEmployeeInfo, session);
} catch (error) {
   // Do something with error
} finally {
   session.endSession();
}
```

事务与会话相关联。当启动一个事务的时候，需要先启动一个会话。在任何给定的时间，一个会话只能有一个打开的事务。如果会话结束并且它有一个打开的事务，则该事务将中止。

须知

使用驱动程序时，必须将会话传递给事务中的每个操作。

- 事务和原子性

  多文档事务是原子性的：

  - 当事务提交时，在事务中进行的所有数据更改都将被保存并在事务外部可见。在事务提交之前，事务中所做的数据更改在事务外部是不可见的。
  - 当事务中止时，在事务中进行的所有数据更改都将被丢弃，而不会变得可见。例如，如果事务中的任何操作失败，事务将中止，并且在事务中进行的所有数据更改都将被丢弃，而不会变得可见。

- 事务的操作限制

  在事务中：

  - 您可以在已经存在的集合上指定读/写（CRUD）操作。集合可以在不同的数据库中。
  - 您无法读取/写入config、admin或local数据库中的集合。
  - 您无法写入system.*集合。
  - 不支持explain方法。
  - 对于在事务外部创建的游标，不能在事务内部调用getMore。
  - 对于在事务中创建的游标，不能在事务外部调用getMore。
  - 无法执行非CRUD的命令，包括listCollections/listIndexes/createUser/getParameter/count等。
  - 要在事务中执行计数操作，请使用$count聚合阶段或$group（带有$sum表达式）聚合阶段。从DDS4.0开始，mongo shell提供了db.collection.countDocuments()方法，该方法使用带有$sum表达式的$group来执行计数。
  - 不能包含会导致创建新集合的插入操作，即隐式创建集合的操作。不能包含影响数据库目录的操作，例如创建或删除集合或索引。

- 事务使用注意事项

  - 运行时间限制。

    默认情况下，事务的运行时间必须小于一分钟。您可以使用transactionLifetimeLimitSeconds参数修改此限制。超过此限制的事务将被视为已过期，并将由定期清理过程中止。

  - Oplog大小限制

    当事务提交时，如果事务包含任何写操作，则创建单个oplog（操作日志）条目。也就是说，事务中的各个操作没有对应的oplog条目。相反，单个oplog条目包含一个事务中的所有写操作。事务的oplog条目必须在16MB的BSON文档大小限制内。

  - 缓存

    要防止存储缓存压力对性能产生负面影响，请执行以下操作：

    - 当您放弃一个事务时，中止该事务。
    - 当您在事务中的单个操作过程中遇到错误时，请中止事务并重试事务。

    transactionLifetimeLimitSeconds参数可以确保定期中止过期的事务，以缓解存储缓存压力。

- 事务和锁

  默认情况下，事务最多等待5毫秒来获取事务中的操作所需的锁。如果事务无法在5毫秒内获得所需的锁，则事务中止。事务在中止或提交时会释放所有持有的锁。

  获取锁请求超时时间，您可以使用maxTransactionLockRequestTimeoutMillis参数来调整事务等待获取锁的时间。增大maxTransactionLockRequestTimeoutMillis可以让事务中的操作等待指定的时间来获取所需的锁。这有助于避免在瞬时并发锁获取（如快速运行的元数据操作）时发生事务中止。但是，这可能会延迟发生死锁的事务中止。

- 挂起的DDL操作和事务

  如果多文档事务正在进行，则影响相同数据库的新DDL操作会在事务之后等待。当这些挂起的DDL操作存在时，与挂起的DDL操作访问同一数据库的新事务无法获得所需的锁，并将在等待maxTransactionLockRequestTimeoutMillis后中止。此外，访问同一数据库的新的非事务性操作将阻塞，直到它们达到其maxTimeMS限制。

  为了解释说明，请比较以下两种情况：

  考虑这样一种情况：一个正在进行的事务对hr数据库中的employees集合执行各种CRUD操作。当该事务正在进行时，可以启动并完成访问hr数据库中foobar集合的单独事务。

  考虑另一种情况：正在进行的事务对hr数据库中的employees集合执行各种CRUD操作，并发出单独的DDL操作以在hr数据库中的employees 集合上创建索引。DDL操作需等待事务完成。

  当DDL操作挂起时，一个新的事务尝试访问hr数据库中的foobar集合。如果DDL操作在maxTransactionLockRequestTimeoutMillis之前一直挂起，则新事务将中止。

  - **正在进行的事务和写入冲突**

    如果多文档事务正在进行中，并且事务外部的写操作修改了该事务中的操作稍后将会尝试修改的文档，则事务会因为写冲突而中止。 如果一个多文档事务正在进行，并且已经获取了一个锁来修改文档，那么当事务外部的写操作尝试修改同一个文档时，该写操作会一直等待，直到事务结束。

  - **正在进行的事务和过时读取**

    事务内的读操作可以返回陈旧的数据。也就是说，事务内的读操作不能保证看到其他已提交事务或非事务性写操作执行的写操作。

    例如，以下序列：

    - 事务正在进行中。
    - 事务外部的写入删除文档。
    - 事务内部的读取操作能够读取现在删除的文档，因为该操作使用的是写入之前的快照。

    为了避免单个文档的事务内部的陈旧读取，可以使用db.collection.findOneAndUpdate()方法。例如：

```
// insert data
employeesCollection = db.getSiblingDB("hr").employees;
employeesCollection.drop();
employeesCollection.insertOne({ _id: 1, employee: 1, status: "Active" });

// Start a session.
session = db.getMongo().startSession( { readPreference: { mode: "primary" } } );
```

```
session.startTransaction( { readConcern: { level: "snapshot" }, writeConcern: { w:
"majority" } } );

employeesCollection = session.getDatabase("hr").employees;

employeeDoc = employeesCollection.findOneAndUpdate(
   { _id: 1, employee: 1, status: "Active" },
   { $set: { employee: 1 } },
   { returnNewDocument: true }
);
print(tojson(employeeDoc));
```

须知

- 如果employee文档在事务之外发生了更改，则事务中止。
- 如果employee文档没有更改，事务将返回文档并锁定文档。

- 事务使用最佳实践
  - 默认情况下，DDS将自动中止任何运行超过60秒的多文档事务。请注意，如果服务器的写入量很低，您可以灵活地调整事务以获得更长的执行时间。为了解决超时问题，应该将事务分成更小的部分，以便在配置的时间限制内执行。您还应该确保使用适当的索引覆盖来正确优化您的查询模式，以允许在事务中快速访问数据。
  - 对于一个事务中可以读取的文档数量没有硬性限制。作为最佳实践，在一个事务中修改的文档不应超过1000个。对于需要修改超过1000个文档的操作，开发人员应该将事务分解为单独的部分，这些部分可以批量处理文档。
  - 在DDS4.0中，事务在单个oplog条目中表示，因此必须在16MB文档大小限制内。更新操作只存储更新的增量（即已更改的内容），而插入操作将存储整个文档。因此，事务中所有语句的oplog描述的总和必须小于16MB。如果超过此限制，事务将被中止并完全回滚。因此，事务应该被分解成一个更小的操作集，这些操作集可以用16MB或更小的空间表示。
  - 当事务中止时，将异常返回给驱动程序，并将事务完全回滚。开发人员可以添加捕获并重试由于临时异常（例如暂时的网络故障或主副本选举）而中止的事务的应用逻辑。对于可重试写入，DDS驱动程序将自动重试事务的提交语句。
  - DDL操作（如创建索引或删除数据库）会阻塞命名空间上正在运行的事务。在DDL操作挂起期间尝试新访问命名空间的所有事务将无法获得锁，从而中止新事务。

事务和读关注

事务中的操作使用事务级别的读关注。也就是说，在事务内部，在集合和数据库级别设置的任何读关注都会被忽略。可以在事务开始时设置事务级别的读关注。

如果未设置事务级别的读关注，则事务级别的读关注默认为会话级别的读关注。

如果未设置事务级别和会话级别的读关注，则事务级别的读关注默认为客户端级别的读关注。默认情况下，对于针对主节点的读取，客户端级别的读关注为"local"

- 多文档事务支持以下读关注级别：
  - "local"

    在大多数情况下，建议使用"majority"级别的读关注。

  - "majority"

    如果事务以写关注"majority"提交，则读关注"majority"返回已被大多数副本集成员确认的数据（即不会被回滚的数据）。

    如果事务没有对提交使用写关注"majority"，则读关注"majority"不能保证读操作读取多数提交的数据。

  - "snapshot"

    如果事务以写关注"majority"提交，读关注"snapshot"将从多数已提交数据的快照中返回数据。

    如果事务没有使用写关注"majority"提交，则读关注"snapshot"不能保证读操作使用多数已提交数据的快照。

- 事务和读关注使用建议
  - 在大多数情况下，建议使用"majority"级别的读关注。

    这种设置能够保证数据的隔离性和一致性，只有当数据被复制到副本集中的大多数节点时，应用程序才能读取到该数据。这样，即使在选举新的主节点时，数据也不会发生回滚。

  - 在需要“读取自己所写”的场景中，需要直接从主节点读取，并使用"local"级别的读关注。

    这样可以在写操作完成后尽快读取到最新的修改。如果配套的写关注设置为"majority"，也可以使用"majority"级别的读关注。

事务和写关注

事务使用事务级别的写操作来提交写操作。事务内部的操作设置的写关注将被忽略。不要对事务内的单个写操作显式设置写关注。您可以在事务开始时设置事务级别的写关注：

如果未设置事务级别的写关注，则事务级别的写关注默认为提交的会话级别的写关注。

如果未设置事务级写关注和会话级写关注，则事务级写关注默认为客户端级写关注。默认情况下，客户端级别的写关注为w: 1。

- 多文档事务支持以下写关注w值：
  - w: 1

    写关注w: 1：在提交应用到主节点后返回确认。当使用w: 1提交时，如果发生故障，事务会被回滚。

    当使用写关注w: 1进行提交时，事务级别的读关注"majority"并不能保证事务中的读操作能读取到大多数节点已提交的数据。

    当您使用写关注w: 1进行提交时，事务级别的读关注"snapshot"无法保证事务中的读操作使用了大多数节点已提交数据的快照。

  - w: "majority"

    写关注w: "majority"在提交已应用于大多数（M个）投票成员之后返回确认；即，该提交已应用于主节点和（M-1个）有投票权的从节点。

    当使用写关注w: "majority"进行提交时，事务级别的读关注"majority"保证能读取到大多数节点已提交的数据。

    当使用写关注w: "majority"进行提交时，事务级别的读关注"snapshot"保证能读取来自大多数节点已提交数据的同步快照。

- 事务和写关注使用建议

- 在大多数情况下，推荐使用"majority"级别的写关注·。

  这种设置能够确保写入操作得到副本集中大多数节点的确认，从而即使在节点故障或异常切换的情况下，也能避免数据丢失或回滚的风险。

- 在需要高写入性能的场景中，可以考虑使用w: 1的写关注，并密切关注从节点的复制延迟。

  w: 1的写关注通常能提供更优的写入性能，适用于写入密集型的场景。同时，必须合理监控从节点的复制延迟，因为延迟过大可能导致主节点异常回滚。此外，如果复制延迟超过了oplog的保留时间，从节点可能会进入异常的RECOVERING状态且无法自动恢复，从而降低实例的可用性，因此应优先关注并处理此类问题。

- 根据不同的操作需求，设置最适合的写关注。

  业务侧可以根据实际需求灵活调整写关注，以满足多样化的业务场景。例如，金融交易数据可以使用带写关注的事务来保证原子性；游戏业务核心玩家数据可以使用w: "majority"级别的写关注来确保数据不会被回滚；日志数据则可以使用默认或w: 1的写关注获得更优的写入性能。

21 DDS 指标告警配置建议

操作场景

通过在云监控服务界面设置告警规则，用户可自定义监控目标与通知策略，及时了解实例的运行状况，从而起到预警作用。DDS支持为实例的监控指标设置阈值告警规则。当监控指标的值超出设置的阈值时就会触发告警，系统会通过SMN自动发送报警通知给云账号联系人，帮助您及时了解DDS实例的运行状况。

本章节介绍了设置DDS指标告警规则的配置建议。

创建告警规则

步骤1 登录管理控制台。

步骤2 选择“管理与监管 > 云监控服务”。

步骤3 在左侧导航树，选择“告警 > 告警规则”。

步骤4 在“告警规则”页面，单击“创建告警规则”。

步骤5 在“创建告警规则”界面，根据界面提示配置参数。

此处需要关注如下参数：

- 事件来源：选择文档数据库服务。
- 维度：DDS支持实例级别和节点级别的监控维度，不同的监控指标支持的监控维度不同，请参考文档数据库服务监控指标说明。

图 21-1 配置监控维度

- 其他参数，请参考《云监控服务用户指南》中“**设置告警规则**”章节进行配置。

----结束

表 21-1 DDS 指标告警配置建议

指标ID	指标名称	指标维度	最佳实践阈值	最佳实践告警级别	告警后的处理建议
mongo007_connections_usage	当前活跃连接数百分比	节点级	连续3个周期 原始值 > 80 %	重要	• 建议用户检查业务是否有合理使用连接池。 • 建议用户检查连接常用的超时等配置参数是否配置合理，详情请参考：**驱动侧通用参数配置**。 • 建议可提高最大连接数。 – 副本集最大连接数在16000以下可通过规格变更提高最大连接数。 – 集群可通过增加mongos数量提高最大连接数。
mongo031_cpu_usage	CPU使用率	节点级	连续3个周期 原始值 > 80 %	重要	• 建议排查CPU高的原因，详情请参考**排查DDS实例CPU使用率高的问题**。 • 建议在CPU持续高位的情况下，升配CPU规格，详情请参考**变更实例的CPU和内存规格**。 • 如果CPU持续高且用户业务不能完全停掉，规格变更可能会失败，此时可选择“**工单 > 新建工单**”，完成工单提交，要求工程师在后台通过一键限制连接数的能力限制业务连接，规格变更结束后再恢复连接数。

指标ID	指标名称	指标维度	最佳实践阈值	最佳实践告警级别	告警后的处理建议
mongo035_disk_usage	磁盘利用率	节点级	连续3个周期 原始值 > 80 %	重要	• 建议排查磁盘高的原因，详情请参考**磁盘使用率高问题排查**。 • 建议在磁盘使用率持续高位的情况下，进行磁盘扩容，详情请参考**扩容磁盘**。
mongo032_mem_usage	内存使用率	节点级	连续3个周期 原始值 > 90 %	重要	• 建议根据官网排查内存高的原因，详情请参考**内存使用率高问题排查**。 • 如果持续长期内存处于高位，请根据业务情况审视升配内存。详情请参考**变更实例的CPU和内存规格**。
mongo039_avg_disk_sec_per_read	硬盘读耗时	节点级	连续3个周期 阈值 >= 0.1 s	重要	• 建议检查实例是否存在CPU、内存、连接数等的性能瓶颈，如果有的话请参考相关指标建议解决性能瓶颈问题。 • 建议在业务无法优化的场景下，升配实例规格或变更到磁盘性能更优的规格。详情请参考：**变更实例的CPU和内存规格**和**实例规格**。

指标ID	指标名称	指标维度	最佳实践阈值	最佳实践告警级别	告警后的处理建议
mongo040_avg_disk_sec_per_write	硬盘写耗时	节点级	连续3个周期 阈值 >= 0.1 s	重要	• 建议检查实例是否存在CPU、内存、连接数等的性能瓶颈，如果有的话请参考相关指标建议解决性能瓶颈问题。 • 建议在业务无法优化的场景下，升配实例规格或变更到磁盘性能更优的规格。详情请参考：**变更实例的CPU和内存规格**和**实例规格**。 • 用户可在console尝试主备倒换。

22 使用索引

操作场景

在数据库管理中，使用索引可以显著提高查询效率。索引的使用不仅限于单个字段，还可以创建复合索引，以优化涉及多个字段的查询。

在使用索引时，应定期分析和优化索引性能。数据库管理系统通常提供工具和命令，帮助分析索引的使用情况和性能瓶颈。此外，索引的维护成本也不容忽视。插入、更新和删除操作会触发索引的更新，可能会影响性能。因此，应根据实际应用场景，合理选择索引策略，平衡查询性能和维护成本。

创建索引的关键

- 避免使用没有过滤条件的聚合/查询语句。务必确保在过滤条件（比如聚合中的$match，find，update，remove语句的过滤条件）上存在对应索引。避免造成全表聚合，全表扫描等。
- 尽量使用高选择性过滤条件缩小查询范围，同时在高选择性字段上保证索引的存在。建议将索引创建限制为其中的重复值数量低于集合中总文档数的1%的字段。例如，如果您的集合包含100,000个文档，则仅在相同值出现1000次或更少的字段上创建索引。
- 多字段索引中，能缩小查询范围的高选择性字段尽量放在前面，索引效果更好。
- 有大数据量排序的业务场景，需要在sort字段结合精准匹配的过滤条件创建索引。例如：过滤条件为{a:xx, b:xx, c:xx, f:xx } 并且 sort({e:1})的业务，需要创建全覆盖索引：{a:1,b:1,c:1,f:1,e:1} ，避免产生内存排序。
- 对于同时存在精确匹配，范围匹配，排序的语句，组合索引的最佳方式：ESR原则：精确（Equal）匹配的字段放最前面，排序（Sort）条件放中间，范围（Range）匹配的字段放最后面。
- 携带collation的查询，在对应查询字段的创建索引时，也需要携带collation选项。
- 如果需要删除索引，需要确认备机无正在创建的索引（不确定可以通过使用currentOp命令进行查看当前执行中的命令）。若备机有索引正在创建，禁止立即删除索引，该操作会导致备机的删除索引操作陷入锁等待状态。

创建索引的影响

创建索引请求下发后，会扫描当前数据的原始文档，构建索引字段与磁盘位置的映射关系数据，因此会产生IO与计算资源开销。因此：

- 当需要创建的索引涉及的数据量较多时，务必需要在业务低谷期进行创建，避免影响到生产业务。
- 当同一个集合需要创建多个索引时，使用createIndexes代替createIndex，可以在一次扫描中构建多个索引，减少创建索引带来的计算资源和IO开销。
- 核心业务表在早期需要提前规划好可能用到的索引。

索引对数据写入的影响

虽然索引能够通过避免扫描集合中的每个文档来提高查询性能，但这种提升是有代价的。对于集合中的每个索引，每次插入、更新或删除文档时，数据库都需要更新索引并将字段写入每个索引。例如，如果一个集合有九个索引，数据库必须执行十次写入操作，才能将操作结果通知给客户端。因此，每增加一个索引都会增加写入延迟、开销和总存储空间的占用。为了获得最佳性能，建议定期审查并减少集合中不必要的索引，仅添加必要的索引来优化常用查询的性能。建议将每个集合的索引数量控制在十个以内。