阿里云
云数据库 MongoDB 版

最佳实践

文档版本：20190815

法律声明

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

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

通用约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **警告:** 重启操作将导致业务中断，恢复业务所需时间约10分钟。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 **说明:** 您也可以通过按Ctrl + A选中全部文件。
>	多级菜单递进。	设置 > 网络 > 设置网络类型
粗体	表示按键、菜单、页面名称等UI元素。	单击 确定。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
*## *	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[]或者[a|b]	表示可选项，至多选择一个。	`ipconfig` *[-all|-t]*
{}或者{a|b}	表示必选项，至多选择一个。	`swich` *{stand | slave}*

目录

1 云数据库MongoDB版数据安全最佳实践

针对用户重点关注的数据安全，云数据库MongoDB版提供了全面的安全保障。您可以通过同城容灾、RAM授权、审计日志、网络隔离、白名单、密码认证等多手段保障数据库数据安全。

同城容灾

为进一步满足业务场景中高可靠性和数据安全需求，云数据库MongoDB版提供了同城容灾解决方案。该方案将副本集中的节点或分片集群实例中的组件分别部署在同一地域下三个不同的可用区，当三个可用区中的任一可用区因电力、网络等不可抗因素失去通信时，高可用系统将自动触发切换操作，确保整个实例的持续可用和数据安全。

您可以在创建实例时选择多可用区，详情请参见创建多可用区副本集实例或创建多可用区分片集群实例；您也可以将现有的副本集实例从单可用区迁移至多可用区，详情请参见迁移可用区。

权限控制

- 授权RAM用户管理MongoDB实例

  使用访问控制RAM（Resource Access Management），您可以创建、管理子账号，控制子账号对您名下资源的操作权限。当您的企业存在多用户协同操作资源时，使用RAM可以按需为用户分配最小权限，避免与其他用户共享云账号密钥，降低企业的信息安全风险。

  具体操作方法请参见云数据库MongoDB版如何配置RAM授权。
- 创建数据库用户并授权

  请勿在生产环境中直接使用root用户连接数据库，您可以根据业务需求，创建数据库用户并分配权限。

  具体操作方法请参见使用DMS管理MongoDB数据库用户。

网络隔离

- 使用专有网络

  云数据库MongoDB版支持多种网络类型，推荐使用专有网络。

  专有网络是一种隔离的网络环境，安全性和性能均高于传统的经典网络。专有网络需要事先创建，详情请参见创建专有网络。

  当MongoDB实例为经典网络时，您可以将实例的网络类型切换至专有网络，详情请参见切换实例网络类型。如果您的MongoDB实例已经是专有网络，则无需配置。

   说明:

云数据库MongoDB版支持在专有网络环境下开启免密访问，在保障高安全性的前提下提供更便捷的数据库连接方式，详情请参见开启/关闭内网免密访问。

- 合理设置白名单

  MongoDB实例创建完毕后，默认情况下实例的白名单中IP地址为`127.0.0.1`，您必须手动设置白名单地址后才可以连接MongoDB数据库。

  具体操作方法请参见设置白名单。

  说明:

  - 请勿将白名单地址配置为`0.0.0.0/0`，允许所有IP地址访问。
  - 建议按需设置并定期维护白名单，及时删除不再需要的IP地址。

日志审计

云数据库MongoDB版审计日志记录了您对数据库执行的所有操作。通过审计日志，您可以对数据库进行故障分析、行为分析、安全审计等操作，有效帮助您获取数据的执行情况。

具体操作方法请参见审计日志。

数据加密

在通过公网连接数据库时，您可以启用SSL（Secure Sockets Layer）加密功能提高数据链路的安全性。通过SSL加密功能可以在传输层对网络连接进行加密，在提升通信数据安全性的同时，保障数据的完整性。

具体操作方法请参见使用Mongo Shell通过SSL加密连接数据库。

2 设置常用的MongoDB监控报警规则

云数据库MongoDB提供实例状态监控及报警功能。本文将介绍设置磁盘空间使用率、IOPS使用率、连接数使用率、CPU使用率等常用的监控项目。

背景信息

- 随着数据量及业务的发展，MongoDB实例的性能资源使用率可能会逐步提升，直至被消耗殆尽。
- 某些场景下MongoDB实例的性能资源可能被大量地异常消耗。如大量的慢查询引起的CPU使用率上升，大量数据写入导致磁盘空间被急剧消耗等情况。

> 说明:
> 当磁盘容量不足将导致实例被锁定。如遇到实例被锁定您可以提交工单。实例解锁后您可以通过变更配置来增加磁盘空间。

通过对实例的关键性能指标设置监控报警规则，让您在第一时间得知指标数据发生异常，帮助您迅速定位并处理故障。

操作步骤

1. 登录MongoDB管理控制台。
2. 在页面左上角，选择实例所在的地域。
3. 找到目标实例，单击实例ID。
4. 在左侧导航栏中，单击报警规则。
5. 单击设置报警规则，跳转至云监控控制台页面。
6. 在云监控控制台页面，单击页面右上角的创建报警规则。

7. 在创建报警规则页面，设置关联资源。

设置项目	说明
产品	下拉选择实例类型。 · 云数据库MongoDB版-副本集 · 云数据库MongoDB版-分片集群 · 云数据库MongoDB版-单节点实例 说明: 当选择云数据库MongoDB版-分片集群时，请选择需要监控的Mongos节点和Shard节点。
资源范围	· 资源范围选择全部实例，则产品下任何实例满足报警规则描述时，都会发送报警通知。 · 选择指定的实例，则选中的实例满足报警规则描述时，才会发送报警通知。
地域	选择实例所属地域。
实例	选择实例ID，可选择多个实例。

8. 设置报警规则，此处先设置磁盘空间使用率，设置完成后单击添加报警规则。

说明:

- 例如规则描述为磁盘使用率5分钟平均值>=80%，则报警服务会5分钟检查一次5分钟内的数据是否满足平均值>=80%。您可以根据您的业务场景微调相关数值。
- 角色选择为任意角色即代表监控实例的 Primary 节点和 Secondary 节点。

9. 参考上一步骤设置IOPS使用率、连接数使用率、CPU使用率的监控报警规则。

10. 设置报警规则的其他项目。

设置项目	说明
通道沉默时间	指报警发生后如果未恢复正常，间隔多久重复发送一次报警通知。
连续几次超过阈值后报警	即连续几次报警的探测结果符合您设置的规则描述，才会触发报警，建议设置为3次。 例如规则描述为"CPU使用率 5分钟内平均值>80%,连续3次超过阈值后报警"，则连续出现3次 CPU使用率 5分钟内平均值>80%的情况，才会触发报警。
生效时间	设置报警规则生效的时间。

11.设置通知方式。

设置项目	说明
通知对象	发送报警的联系人或联系组，详情请参考报警联系人和报警联系组。
报警级别	分为Critical 、Warning、Info三个等级，不同等级对应不同的通知方式。 · Critical：电话语音+手机短信+邮件+钉钉机器人 · Warning：手机短信+邮件+钉钉机器人 · Info：邮件+钉钉机器人
邮件主题	自定义报警邮件的主题，默认为产品名称+监控项名称+实例ID。
邮件备注	自定义报警邮件补充信息。填写邮件备注后，发送报警的邮件通知中会附带您的备注。
报警回调	详情请参考使用报警回调。

12.设置完成后，单击确认。报警规则将自动生效。

3 Azure Cosmos DB API for MongoDB 迁移到阿里云

使用MongoDB数据库自带的备份还原工具，您可以将Azure Cosmos DB API for MongoDB迁移至阿里云。

注意事项

- 该操作为全量迁移，为避免迁移前后数据不一致，迁移开始前请停止数据库写入。
- 如果您之前使用mongodump命令对数据库进行过备份操作，请将备份在dump文件夹下的文件移动至其他目录。确保默认的dump备份文件夹为空，否则将会覆盖该文件夹下之前备份的文件。
- 请在安装有MongoDB服务的服务器上执行mongodump和mongorestore命令，并非在mongo shell环境下执行。

数据库账号权限要求

实例类型	账号权限
Azure Cosmos DB	read
目的MongoDB实例	readWrite

环境准备

1. 创建云数据库MongoDB实例，详情请参考创建实例。

   说明:

   - 实例的存储空间要大于Azure Cosmos DB。
   - 实例的数据库版本选用3.4。

2. 设置阿里云MongoDB数据库的数据库密码，详情请参考设置密码。
3. 在某个服务器上安装MongoDB程序，详情请参考安装MongoDB。

   说明:

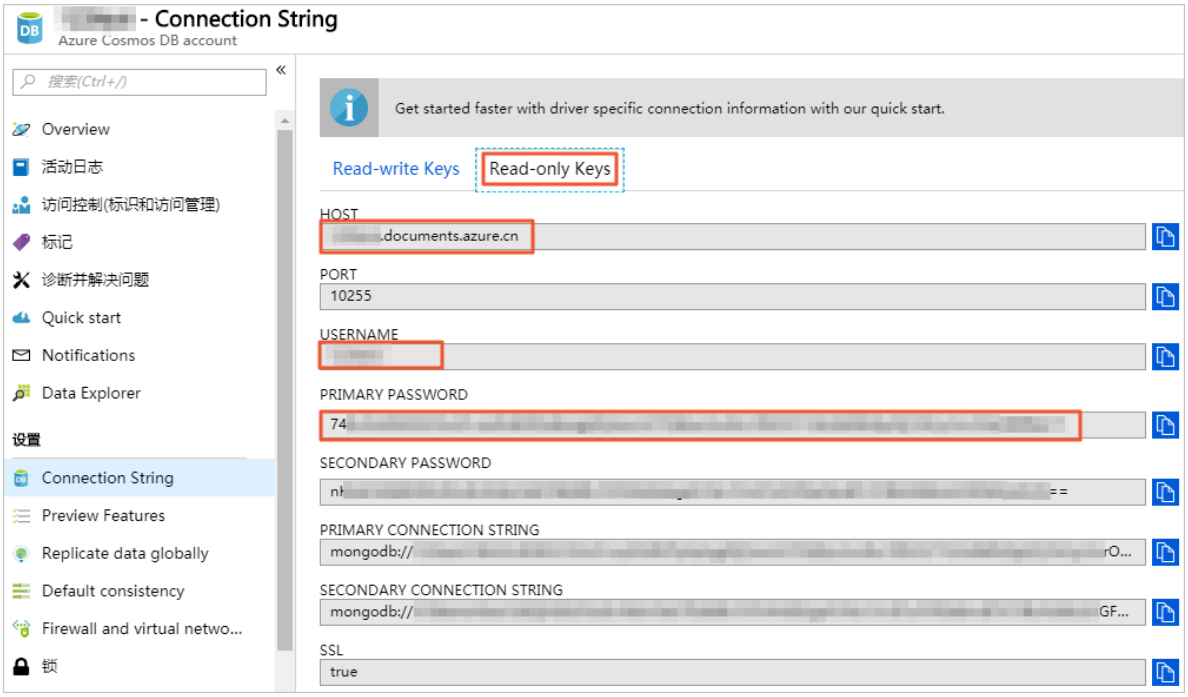
   - 请安装MongoDB3.0以上版本。
   - 该服务器仅作为数据备份与恢复的临时中转平台，迁移操作完成后不再需要。
   - 备份目录所在分区的可用磁盘空间要大于Azure Cosmos DB。

   本案例将MongoDB服务安装在Linux服务器上进行演示。

迁移步骤

1. 登录Azure门户。
2. 在左侧导航栏单击Azure Cosmos DB。
3. 在Azure Cosmos DB页面，单击需要迁移的Cosmos DB 账户名称。
4. 在账户详情页，单击Connection String。
5. 单击Read-only Keys页签，查看连接该数据库所需的信息。

   图 3-1: Azure连接信息



> **说明:**
> 迁移数据时使用只读权限的账号密码信息即可。

6. 在安装有MongoDB服务的Linux服务器上执行以下命令进行数据备份，将数据备份至该服务器上。

```
mongodump --host <HOST>:10255 --authenticationDatabase admin -u <
USERNAME> -p <PRIMARY PASSWORD> --ssl --sslAllowInvalidCertificates
```

   说明：将<HOST>、<USERNAME>、<PRIMARY PASSWORD>更换为Azure连接信息图中对应选项的值。

   等待备份完成，Azure Cosmos DB的数据库将备份至当前目录下dump文件夹中。

7. 获取阿里云MongoDB数据库的Primary节点连接地址，详情请参考实例连接说明。

8. 在安装有MongoDB服务的Linux服务器上执行以下语句将数据库数据全部导入至阿里云MongoDB数据库。

```
 mongorestore --host <mongodb_host>:3717 --authenticationDatabase
admin -u <username> -p <password> dump
```

说明：

- <mongodb_host>：MongoDB实例的Primary节点连接地址。
- <username>：登录MongoDB实例的数据库用户名。
- <password>：登录MongoDB实例的数据库密码。

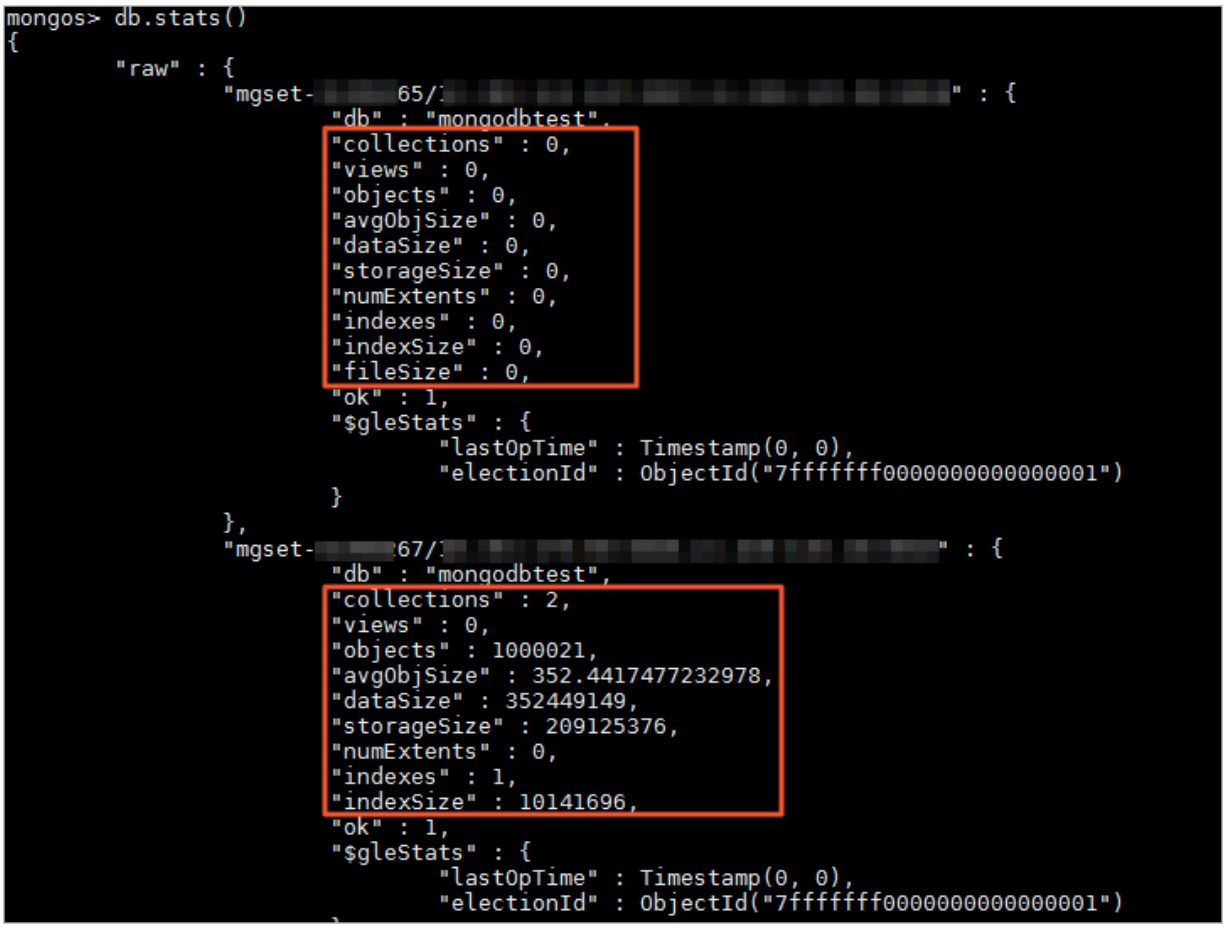
等待数据恢复完成，Azure Cosmos DB API for MongoDB数据库即迁移至阿里云MongoDB数据库中。

4 设置数据分片以充分利用Shard性能

您可以对分片集群实例中的集合设置数据分片，以充分利用Shard节点的存储空间和计算性能。

背景信息

如果没有对集合设置数据分片，数据将被集中存放在一个Shard节点中，这将导致其他Shard节点的存储空间和计算性能无法被充分利用。



前提条件

实例类型为分片集群实例。

注意事项

- 片键一经设置，不可修改，不可删除。
- 执行了数据分片操作后，均衡器会对满足条件的数据进行拆分，这将占用实例的资源，请在业务低峰期操作。

> **说明:**

您可以在设置数据分片之前，调整均衡器的活动窗口，指定它在业务低峰期执行均衡操作。详情请参见设置Balancer的活动窗口。

- 片键的选取将影响分片集群实例性能，关于片键选取的案例介绍，请参见如何选择Shard Key。

分片策略介绍

分片策略	说明	适用场景
基于范围的分片	MongoDB按照片键的值的范围将数据拆分为不同的块（chunk），每个块包含了一段范围内的数据。 · 优点： mongos可以快速定位请求需要的数据，并将请求转发到相应的Shard节点中。 · 缺点： 可能导致数据在Shard节点上分布不均衡，容易造成读写热点，且不具备写分散性。	片键的值不是单调递增或单调递减、片键的值基数大且重复的频率低、需要范围查询等业务场景。
基于Hash值的分片	MongoDB计算单个字段的哈希值作为索引值，并以哈希值的范围将数据拆分为不同的块。 · 优点：可以将数据更加均衡地分布在各Shard节点中，具备写分散性。 · 缺点：不适合进行范围查询，进行范围查询时，需要将读请求分发到所有的Shard节点。	片键的值存在单调递增或递减、片键的值基数大且重复的频率低、需要写入的数据随机分发、数据读取随机性较大等业务场景。

除了上述两种分片策略，您还可以配置复合片键，例如由一个低基数的键和一个单调递增的键组成，详情请参见Shard Key选择案例。

操作步骤

本文以mongodbtest数据库，customer集合为例介绍操作流程。

1. 通过Mongo Shell登录分片集群实例。
2. 对集合所在的数据库启用分片功能。

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

说明:

<database>：数据库名。

示例：

```
sh.enableSharding("mongodbtest")
```

说明：

您可以通过sh.status()查看分片状态。

3. 对片键的字段建立索引。

```
db.<collection>.createIndex(<keyPatterns>,<options>)
```

说明：

- <collection>：集合名。
- <keyPatterns>：包含用于建立索引的字段和索引类型。

  常见的索引类型如下：

  - 1：创建升序索引
  - -1：创建降序索引
  - "hashed"：创建哈希索引
- <options>：表示接收可选参数，详情请参见db.collection.createIndex()，本操作示例中暂未使用到该字段。

创建升序索引示例：

```
db.customer.createIndex({name:1})
```

创建哈希索引示例：

```
db.customer.createIndex({id:"hashed"})
```

4. 对集合设置数据分片。

```
sh.shardCollection("<database>.<collection>",{ "<key>":<value> } )
```

说明：

- <database>：数据库名。
- <collection>：集合名。
- <key>：分片的键，MongoDB将根据片键的值进行数据分片。

- <value>
  - 1：表示基于范围分片，通常能很好地支持基于片键的范围查询。
  - “hashed”：表示基于哈希分片，通常能将写入均衡分布到各Shard节点中。

基于范围分片的配置示例：

```
sh.shardCollection("mongodbtest.customer",{"name":1})
```

基于哈希分片的配置示例：

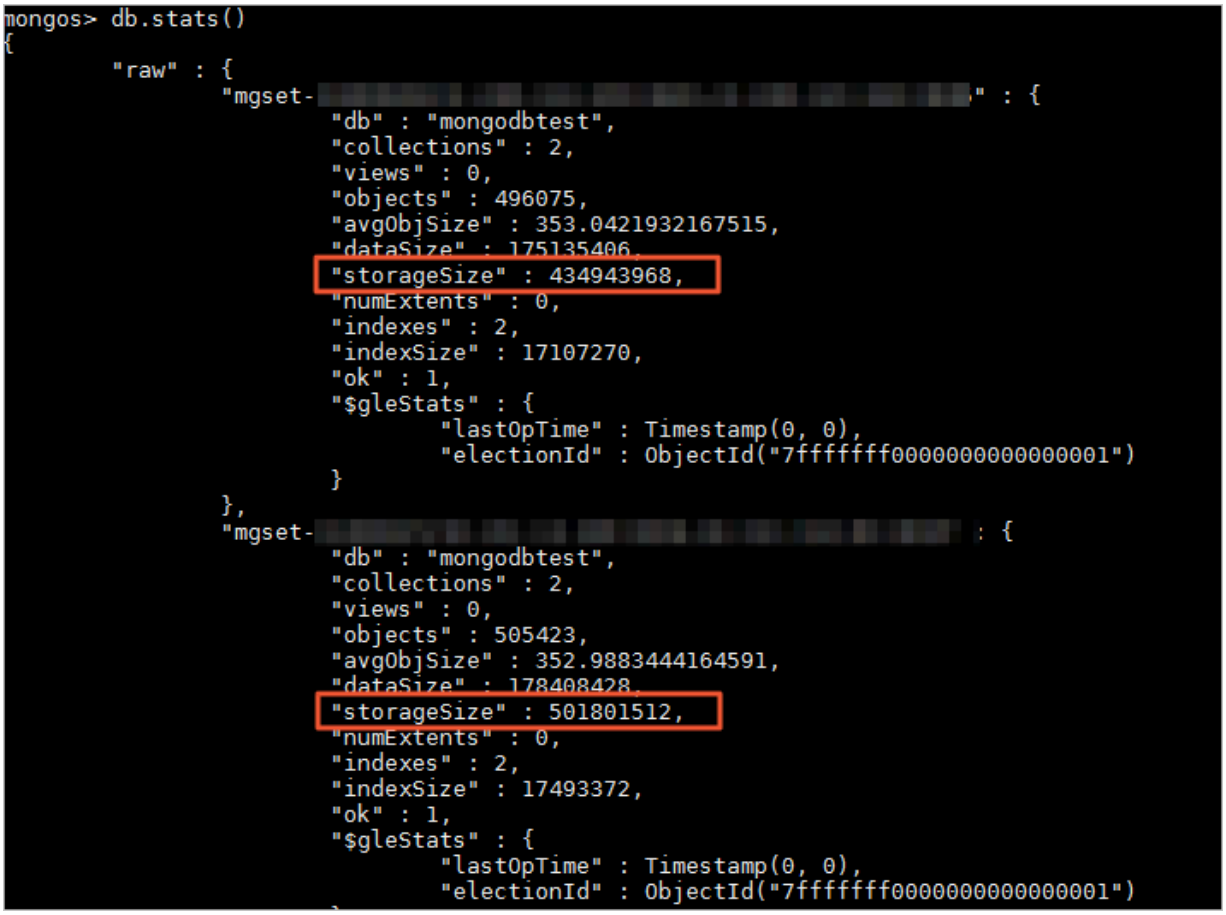
```
sh.shardCollection("mongodbtest.customer",{"id":"hashed"})
```

后续操作

经过一段时间的运行或数据写入后，您可以在Mongo Shell中执行`sh.status()`，查看数据在各Shard节点中的分布信息。

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("5[redacted]")
  }
  shards:
        {  "_id" : "d-bp[redacted]3c4",  "host" : "mgset-[redacted]29/[redacted]",  "state" : 1 }
        {  "_id" : "d-bp[redacted]834",  "host" : "mgset-[redacted]27/[redacted]"  "state" : 1 }
  active mongoses:
        "3.4.6" : 2
  autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
NaN
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                51 : Success
  databases:
        {  "_id" : "mongodbtest",  "primary" : "d-bp[redacted]834",  "partitioned" : true }
                mongodbtest.customer
                        shard key: { "name" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                d-bp[redacted]3c4       51
                                d-bp[redacted]834       52
                        too many chunks to print, use verbose if you want to force print
        {  "_id" : "test",  "primary" : "d-bp[redacted]834",  "partitioned" : false }
```

您也可以通过执行`db.stats()`查看该数据库在各Shard节点的数据存储情况。


mongos> db.stats()
{
"raw" : {
"mgset-
" : {
"db" : "mongodbtest",
"collections" : 2,
"views" : 0,
"objects" : 496075,
"avgObjSize" : 353.0421932167515,
"dataSize" : 175135406,
"storageSize" : 434943968,
"numExtents" : 0,
"indexes" : 2,
"indexSize" : 17107270,
"ok" : 1,
"$gleStats" : {
"lastOpTime" : Timestamp(0, 0),
"electionId" : ObjectId("7fffffff0000000000000001")
}
},
"mgset-
: {
"db" : "mongodbtest",
"collections" : 2,
"views" : 0,
"objects" : 505423,
"avgObjSize" : 352.9883444164591,
"dataSize" : 178408428,
"storageSize" : 501801512,
"numExtents" : 0,
"indexes" : 2,
"indexSize" : 17493372,
"ok" : 1,
"$gleStats" : {
"lastOpTime" : Timestamp(0, 0),
"electionId" : ObjectId("7fffffff0000000000000001")

5 整理物理空间碎片以提升磁盘利用率

MongoDB数据库在长期频繁地删除/写入数据或批量删除了大量数据，将产生很多物理空间碎片。这些碎片将占用磁盘空间，降低磁盘利用率。您可以对集合中的所有数据和索引进行重写和碎片整理，释放未使用的空间，提升磁盘利用率和查询性能。

前提条件

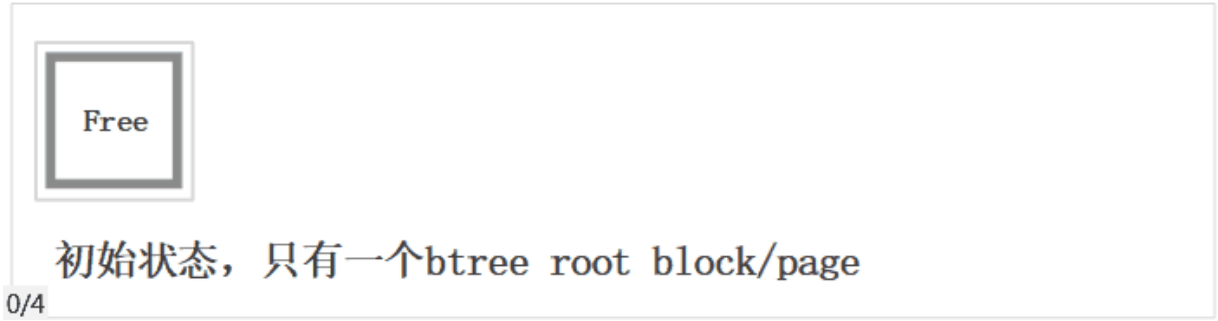
实例的存储引擎为WiredTiger。

注意事项

- 执行该操作前，建议对数据库进行备份，详情请参见手动备份MongoDB数据。
- 执行该操作会导致集合所属的数据库被锁定，且该数据库的读写操作将被阻塞，请在业务低峰期操作。

> 说明:
> 执行物理空间回收命令（compact）所需的时间与集合数据量、系统负载等因素有关。

背景信息



通常情况下，使用`db.collection.remove({}, {multi: true})`命令删除文档，文档将从btree中删除，但是文档占用的物理空间不会被回收。如果执行remove命令删除了大量的文档，且后续的数据写入操作很少，这将降低磁盘利用率，此时可执行compact命令回收空闲的物理空间。

> 说明:
> - 新写入的数据将会使用未被回收的物理空间，所以在数据持续写入的场景中，不需要频繁执行compact命令整理物理空间碎片。
> - 如您使用`db.collection.drop()`命令删除集合，那么集合的物理文件会被删除，且物理空间会被回收。

预估回收的物理空间

1. 通过mongo shell连接MongoDB实例，详情请参见：
   - Mongo Shell连接单节点实例
   - Mongo Shell连接副本集实例
   - Mongo Shell连接分片集群实例
2. 切换至集合所在的数据库。

   ```
   use <database_name>
   ```

   说明：
   <database_name>：数据库名。
3. 执行下列命令查询。

   ```
   db.<collection_name>.stats().wiredTiger["block-manager"]["file bytes
    available for reuse"]
   ```

   说明：
   <collection_name>：集合名。

   查询示例：

   ```
   db.customer.stats().wiredTiger["block-manager"]["file bytes
   available for reuse"]
   ```

   执行结果示例：

   ```
   207806464
   ```

整理单节点实例/副本集实例的碎片

1. 通过mongo shell连接MongoDB实例的Primary节点，详情请参见mongoshell连接实例。
2. 切换至集合所在的数据库。

   ```
   use <database_name>
   ```

   说明：
   <database_name>：数据库名。
3. 执行`db.stats()`命令查看碎片整理前数据库占用的磁盘空间。

4. 执行以下命令，对某个集合进行碎片整理。

```
db.runCommand({compact:"<collection_name>",force:true})
```

> 说明:
>
> - <collection_name>：集合名。
> - force为可选项，如您需要在副本集实例的Primary节点执行该命令，需要设置force的值为true。

5. 等待执行，返回{ "ok" : 1 }代表执行完成。

> 说明:
> compact操作不会传递给Secondary节点，当实例为副本集实例时，请重复上述步骤通过mongo shell连接至Secondary节点，执行碎片整理命令。

碎片整理完毕后，可通过db.stats()命令查看碎片整理后数据库占用的磁盘空间。例如，本案例碎片整理前后的对比：

整理分片集群实例的碎片

1. 通过mongo shell连接分片集群实例中的任一mongos节点，详情请参见mongo shell连接分片集群实例。
2. 执行以下命令，对Shard节点中的Primary节点进行集合的碎片整理。

```
db.runCommand({runCommandOnShard:"<Shard ID>","command":{compact:"<
collection_name>",force:true}})
```

> 说明:

- <Shard ID>：Shard节点ID。
- <collection_name>：集合名。

3. 执行`db.stats()`命令查看碎片整理前数据库占用的磁盘空间。
4. 执行以下命令，对Shard节点中的Secondary节点进行集合的碎片整理。

```
db.runCommand({runCommandOnShard:"<Shard ID>","command":{compact:"<
collection_name>"},queryOptions: {$readPreference: {mode: 'secondary
'}}})
```

说明:

- <Shard ID>：Shard节点ID。
- <collection_name>：集合名。

碎片整理完毕后，可通过`db.runCommand({dbstats:1})`命令查看碎片整理后数据库占用的磁盘空间。

6 管理MongoDB均衡器Balancer

云数据库支持均衡器 Balancer 管理操作。在一些特殊的业务场景下，您可以启用或者关闭 Balancer 功能，设置活动窗口等操作。

注意事项

- Balancer 属于分片集群架构中的功能，该操作仅适用于分片集群实例。
- Balancer 的相关操作可能会占用实例的资源，请在业务低峰期操作。

设置Balancer的活动窗口

均衡器在执行块迁移操作时将占用实例中节点的资源，可能造成节点的资源使用率不均衡，影响业务使用。为避免块迁移给您的业务带来影响，您可以通过设置均衡器的活动窗口，让其在指定的时间段工作。

> 说明:
> 执行该操作须确保 Balancer 功能处于开启状态。如未开启，请参考开启 Balancer 功能。

1. 通过 Mongo Shell 登录MongoDB数据库。
2. 在 mongos 节点命令窗口中，切换至 config 数据库。

```
use config
```

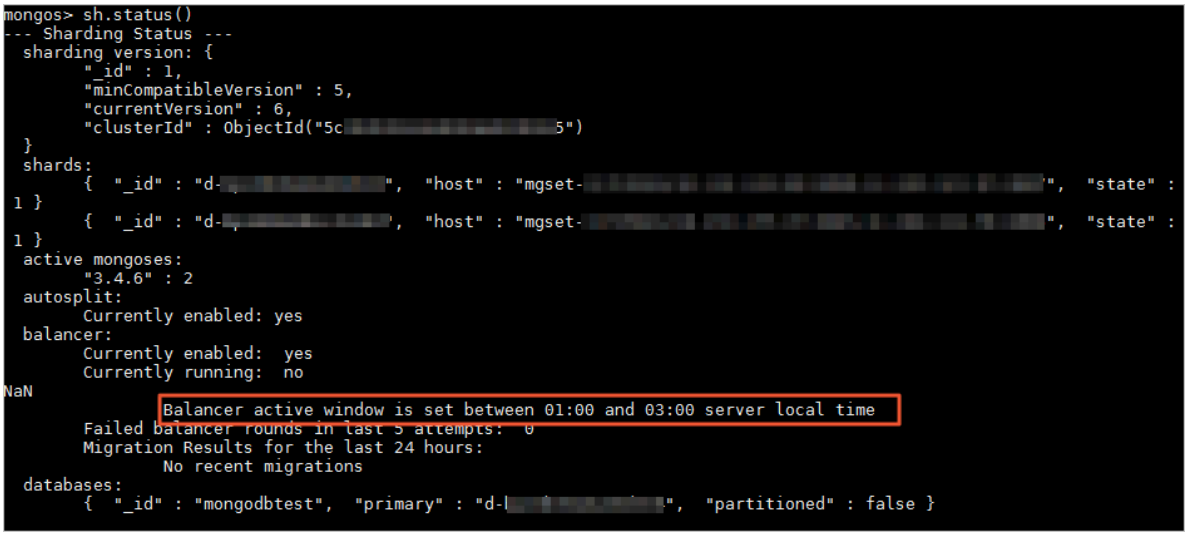
3. 执行如下命令设置 Balancer 的活动窗口。

```
db.settings.update(
   { _id: "balancer" },
   { $set: { activeWindow : { start : "<start-time>", stop : "<stop-time>" } } },
   { upsert: true }
)
```

> 说明:
> - <start-time>：开始时间，时间格式为*HH:MM*（北京时间），HH取值范围为00 - 23，MM取值范围为00 - 59。

- <stop-time>：结束时间，时间格式为*HH:MM*（北京时间），HH取值范围为00 - 23，MM取值范围为00 - 59。

您可以通过执行`sh.status()`命令查看 Balancer 的活动窗口。如下示例中，活动窗口被设置为01:00- 03:00。



相关操作：如您需要 balancer 始终处于运行状态，您可以使用如下命令去除活动窗口的设置。

```
db.settings.update({ _id : "balancer" }, { $unset : { activeWindow :
true } })
```

开启 Balancer 功能

如果您设置了数据分片，开启 Balancer 功能后可能会立即触发均衡任务。这将占用实例的资源，请在业务低峰期执行该操作。

1. 通过 Mongo Shell 登录MongoDB数据库。
2. 在 mongos 节点命令窗口中，切换至 config 数据库。

```
use config
```

3. 执行如下命令开启 Balancer功能。

```
sh.setBalancerState(true)
```

关闭 Balancer 功能

云数据库MongoDB的 Balancer 功能默认是开启状态。如果在某些特殊业务场景下需要临时关闭，请参考下述步骤进行操作。

1. 通过 Mongo Shell 登录MongoDB数据库。

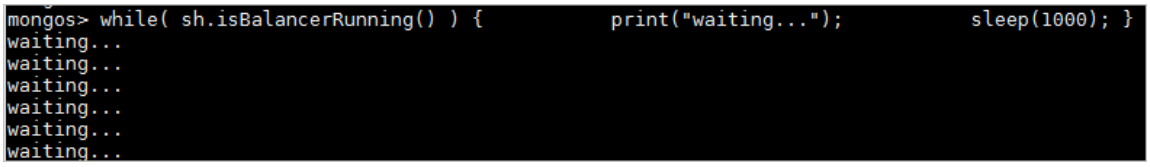
2. 在 mongos 节点命令窗口中，切换至 config 数据库。

```
use config
```

3. 执行如下命令查看 Balancer 运行状态，如返回值为空。

```
while( sh.isBalancerRunning() ) {
          print("waiting...");
          sleep(1000);
}
```

- 返回值为空，表示 Balancer 没有处于执行任务的状态，此时可执行下一步的操作，关闭 Balancer 。
- 返回值 waiting 表示 Balancer 正在执行块迁移，此时不能执行关闭 Balancer 的命令，否则可能引起数据不一致。



4. 确认执行第3步的命令后返回的值为空，可执行关闭 Balancer 命令。

```
sh.stopBalancer()
```

7 使用数据镜像保护尚未写入完整的数据

云数据库MongoDB提供数据镜像能力，您可以对副本集实例或分片集群实例创建一个只读数据镜像。其中副本集最高支持3TB数据，集群版本最高支持96TB数据。

使用场景

创建数据镜像，可确保在数据大批量写入更新期间，所有读请求从数据镜像获取数据。从而确保数据在完整写入前不会被应用程序读取到。数据镜像的读取性能与先前非镜像数据的读取性能完全保持一致。

说明:

数据更新完成后，可将数据正式同步生效，供应用正常连接读取最新数据。通过阿里云提供的数据镜像操作命令，可实现数据自动同步生效（秒级别同步）。数据同步期间不影响正常数据读取操作。

副本集实例操作方法

1. 通过mongo shell连接到需要操作的节点（Primary节点或Secondary节点），连接方法请参考mongo shell 连接副本集实例。
2. 创建数据镜像。

```
db.runCommand({checkpoint:"create"})
```

3. 数据镜像功能使用完毕，删除数据镜像。

```
db.runCommand({checkpoint:"drop"})
```

分片集群实例操作方法

1. 通过mongo shell连接到分片集群实例中的任意一个mongos，连接方法请参考mongo shell 连接分片集群实例。

2. 创建数据镜像。

   - 在所有Shard的Primary节点上创建数据镜像。

     ```
     db.runCommand({runCommandOnShard: "all", "command": {checkpoint:"
     create"}})
     ```

   - 在所有Shard的Secondary节点上创建数据镜像。

     ```
     db.runCommand({runCommandOnShard: "all", "command": {checkpoint:"
     create"}, $queryOptions: {$readPreference: {mode: 'secondary'}}})
     ```

3. 数据镜像功能使用完毕，删除数据镜像。

   - 在所有Shard的Primary节点上删除数据镜像。

     ```
     db.runCommand({runCommandOnShard: "all", "command": {checkpoint:"
     drop"}})
     ```

   - 在所有Shard的Secondary节点上删除数据镜像。

     ```
     db.runCommand({runCommandOnShard: "all", "command": {checkpoint:"
     drop"}, $queryOptions: {$readPreference: {mode: 'secondary'}}})
     ```

8 如何连接副本集实例实现读写分离和高可用

MongoDB副本集实例通过多个数据副本来保证数据的高可靠，通过自动的主备切换机制来保证服务的高可用。需要注意的是，您需要使用正确的方法连接副本集实例来保障高可用，您也可以通过设置来实现读写分离。

使用前须知

- 副本集实例的Primary节点不是固定的。当遇到副本集轮转升级、Primary节点宕机、网络分区等场景时可能会触发主备切换，副本集可能会选举一个新的Primary节点，原先的Primary节点会降级为Secondary节点。
- 若使用Primary节点的地址直接连接Primary节点，所有的读写操作均在Primary节点完成，造成该节点压力较大，且一旦副本集发生主备切换，您连接的Primary会降级为Secondary，您将无法继续执行写操作，将严重影响到您的业务使用。

Connection String 连接说明

要正确连接副本集实例，您需要先了解下MongoDB的Connection String URI，所有官方的driver都支持以Connection String的方式来连接MongoDB。

```
mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN
[:portN]]][/[database][?options]]
```

说明：

- mongodb:// ：前缀，代表这是一个Connection String。
- `username:password@`：登录数据库的用户和密码信息，如果启用了鉴权，需要指定密码。
- `hostX:portX`：副本集成员的IP地址:端口信息，多个成员以逗号分割。
- `/database`：鉴权时，用户帐号所属的数据库。
- `?options`：指定额外的连接选项。

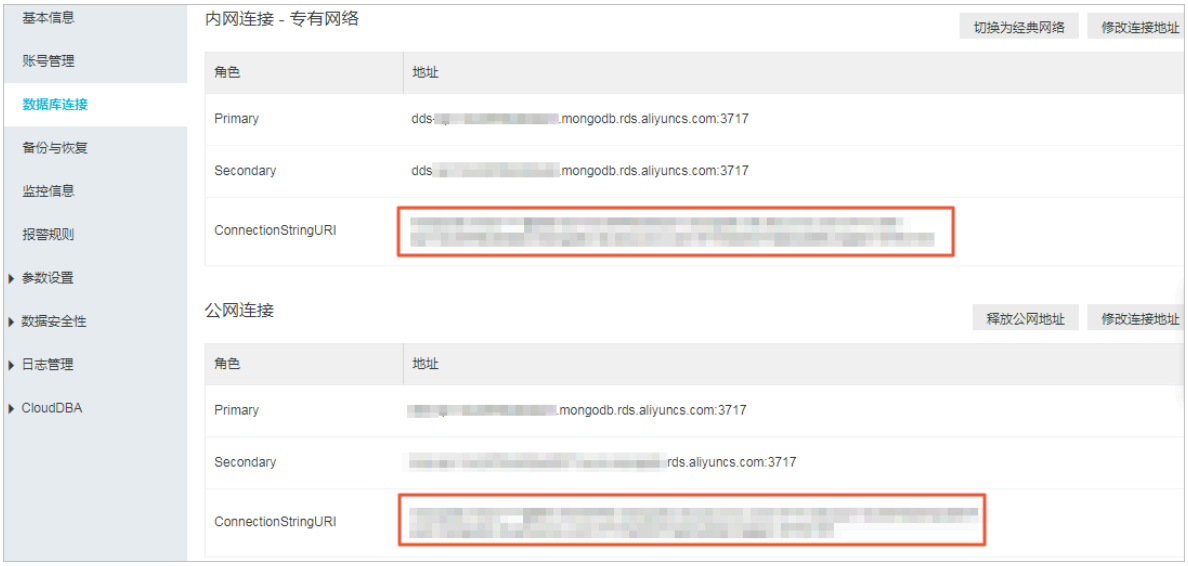
说明：

更多关于 Connection String 请参考MongoDB文档Connection String URI。

副本集实例 Connection String URI 连接示例

云数据库MongoDB提供了 Connection String URI 连接方式。

1. 获取副本集实例的 Connection String URI 连接信息，详情请参考副本集实例连接说明。



2. 应用程序设置使用 Connection String URI 来连接实例，详情请参考程序代码连接实例。

> **说明：**
>
> 要实现读写分离，需要在 Connection String URI 的options里添加`readPreference=secondaryPreferred`，设置读请求为Secondary节点优先。
>
> 更多读选项请参考Read preferences。

示例：

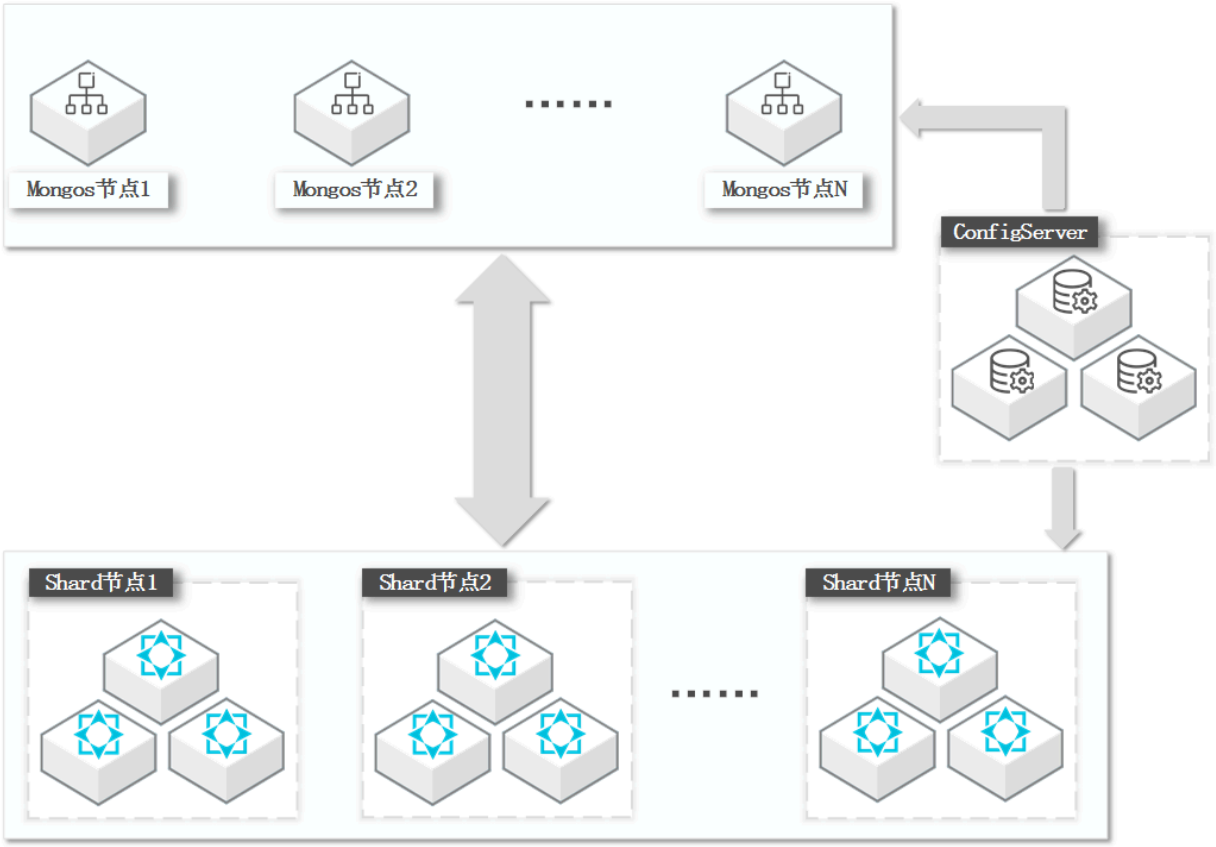
```
mongodb://root:xxxxxxxx@dds-xxxxxxxxxxxx:3717,xxxxxxxxxxxx:3717/
admin?replicaSet=mgset-xxxxxx&readPreference=secondaryPreferred
```

通过上述 Connection String 来连接MongoDB副本集实例，读请求将优先发给Secondary节点实现读写分离。同时客户端会自动检测节点的主备关系，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。

9 使用 Connection String URI 连接分片集群实例

MongoDB分片集群实例提供各个 mongos 节点的连接地址，通过连接 mongos 节点，您可以连接至分片集群实例的数据库。需要注意的是，您需要使用正确的方法连接分片集群实例来实现负载均衡及高可用。

背景信息



MongoDB分片集群（Sharded Cluster）通过将数据分散存储到多个分片（Shard）中，以实现高可扩展性。实现分片集群时，MongoDB 引入 Config Server 来存储集群的元数据，引入 mongos 作为应用访问的入口，mongos 从 Config Server 读取路由信息，并将请求路由到后端对应的 Shard 上。

- 用户访问 mongos 跟访问单个 mongod 类似。
- 所有 mongos 是对等关系，用户访问分片集群可通过任意一个或多个 mongos 。
- mongos 本身是无状态的，可任意扩展，集群的服务能力为“Shard服务能力之和”与“mongos服务能力之和”的最小值。
- 访问分片集群时，最好将应用负载均匀地分散到多个 mongos 上。

Connection String URI 连接说明

要正确连接分片集群实例，您需要先了解下MongoDB的Connection String URI，所有官方的driver都支持以 Connection String URI 的方式来连接MongoDB数据库。

Connection String URI 示例：

```
mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN
[:portN]]][/[database][?options]]
```

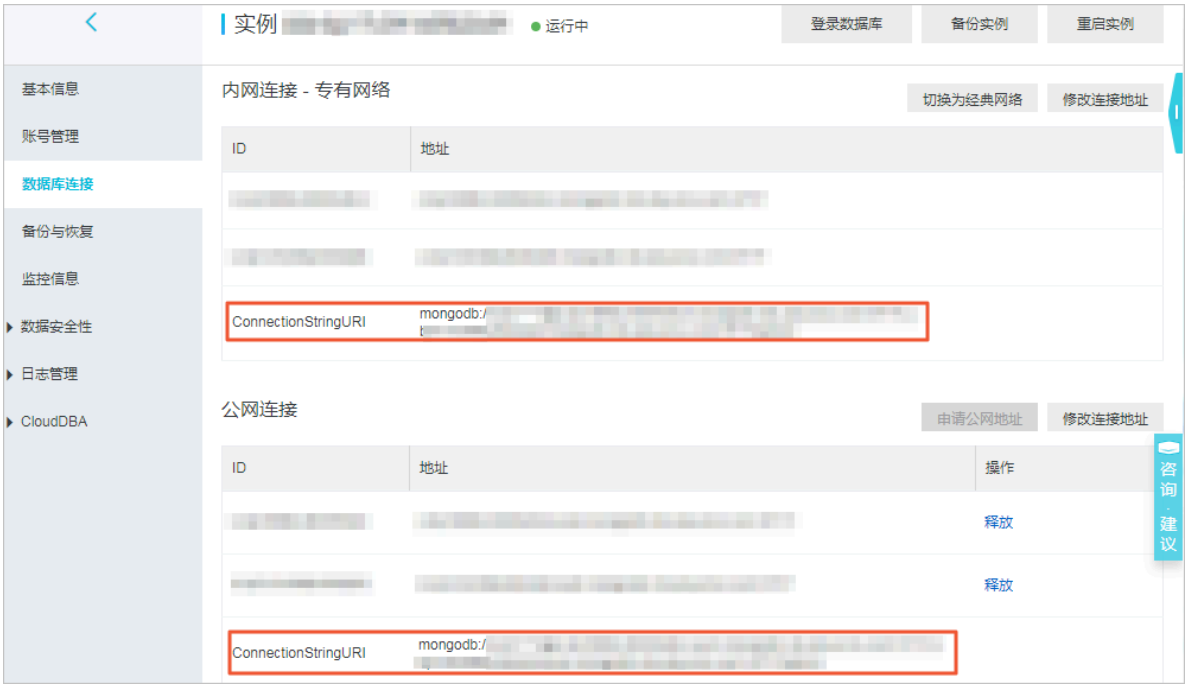
说明：

- `mongodb://` 前缀，代表这是一个Connection String URI 。
- `username:password@` 登录数据库的用户和密码信息。
- `hostX:portX`多个 mongos 的地址列表。
- `/database`鉴权时，用户帐号所属的数据库。
- `?options` 指定额外的连接选项。

如何正确地连接分片集群实例

云数据库MongoDB提供了 Connection String URI 连接方式。使用 Connection String URI 连接方式进行连接，可实现负载均衡及高可用。

1. 获取分片集群实例的 Connection String URI 连接信息，详情请参考分片集群实例连接说明。



2. 应用程序中设置使用 Connection String URI 来连接实例，详情请参考程序代码连接。

   通过 java 来连接的示例代码如下所示。

```
MongoClientURI connectionString = new MongoClientURI("mongodb
://:****@s-xxxxxxxx.mongodb.rds.aliyuncs.com:3717,s-xxxxxxxx.mongodb
.rds.aliyuncs.com:3717/admin"); // ****替换为root密码
MongoClient client = new MongoClient(connectionString);
MongoDatabase database = client.getDatabase("mydb");
MongoCollection<Document> collection = database.getCollection("
mycoll");
```

    说明:

   通过上述方式连接分片集群时，客户端会自动将请求分散到多个 mongos 上，以实现负载均衡。同时，当 URI 里 mongos 数量在2个及以上时，当有 mongos 故障时，客户端能自动进行切换，将请求都分散到状态正常的 mongos 上。

当 mongos 数量很多时，您可以按应用将 mongos 进行分组。例如有2个应用 A、B，实例有4个 mongos，可以让应用 A 访问 mongos 1-2（URI 里只指定 mongos 1-2 的地址），应用 B 来访问 mongos 3-4（URI 里只指定 mongos 3-4 的地址）。根据这种方法来实现应用间的访问隔离。

 说明:

应用访问的 mongos 彼此隔离，但后端 Shard 仍然是共享的。

常用连接参数

- 如何实现读写分离

  在 Connection String URI 的options里添加`readPreference=secondaryPreferred`，设置读请求为Secondary节点优先。

  示例

```
mongodb://root:xxxxxxxx@dds-xxxxxxxxxxxx:3717,xxxxxxxxxxxx:3717/
admin?replicaSet=mgset-xxxxxx&readPreference=secondaryPreferred
```

- 如何限制连接数

  在 Connection String URI 的options里添加 `maxPoolSize=xx`，即可将客户端连接池中的连接数限制在xx以内。

- 如何保证数据写入到大多数节点后才返回

  在 Connection String URI 的options里添加 `w= majority`，即可保证写请求成功写入大多数节点才向客户端确认。

10 排查MongoDB CPU使用率高的问题

在使用云数据库MongoDB的时候您可能会遇到MongoDB CPU使用率很高或者CPU使用率接近100%的问题，从而导致数据读写处理异常缓慢，影响正常业务。本文主要帮助您从应用的角度排查MongoDB CPU使用率高的问题。

分析MongoDB数据库正在执行的请求

1. 通过Mongo Shell连接实例。
   - Mongo Shell连接单节点实例
   - Mongo Shell连接副本集实例
   - Mongo Shell连接分片集群实例
2. 执行`db.currentOp()`命令，查看数据库当前正在执行的操作。

   该命令的输出示例如下。

```
{
        "desc" : "conn632530",
        "threadId" : "140298196924160",
        "connectionId" : 632530,
        "client" : "11.192.159.236:57052",
        "active" : true,
        "opid" : 1008837885,
        "secs_running" : 0,
        "microsecs_running" : NumberLong(70),
        "op" : "update",
        "ns" : "mygame.players",
        "query" : {
            "uid" : NumberLong(31577677)
        },
        "numYields" : 0,
        "locks" : {
            "Global" : "w",
            "Database" : "w",
            "Collection" : "w"
        },
        ....
    },
```

您需要重点关注以下几个字段。

字段	返回值说明
client	该请求是由哪个客户端发起的。
opid	操作的唯一标识符。 说明: 如果有需要，可以通过`db.killOp(opid)`直接终止该操作。

字段	返回值说明
secs_running	表示该操作已经执行的时间，单位为秒。如果该字段返回的值特别大，需要查看请求是否合理。
microsecs_running	表示该操作已经执行的时间，单位为微秒。如果该字段返回的值特别大，需要查看请求是否合理。
ns	该操作目标集合。
op	表示操作的类型。通常是查询、插入、更新、删除中的一种。
locks	跟锁相关的参数，请参考官方文档，本文不做详细介绍。 说明: db.currentOp 文档请参见db.currentOp。

通过`db.currentOp()`查看正在执行的操作，分析是否有不正常耗时的请求正在执行。比如您的业务平时 CPU 使用率不高，运维管理人员连到MongoDB数据库执行了一些需要全表扫描的操作导致 CPU 使用率非常高，业务响应缓慢，此时需要重点关注执行时间非常耗时的操作。

说明:
如果发现有异常的请求，您可以找到该请求对应的opid，执行`db.killOp(opid)`终止该请求。

如果您的应用刚刚上线，MongoDB实例的CPU使用率马上处于持续很高的状态，执行`db.currentOp()`，在输出结果中未发现异常请求，您可参考下述小节分析数据库慢请求。

分析MongoDB数据库的慢请求

云数据库MongoDB默认开启了慢请求Profiling，系统自动地将请求时间超过100ms的执行情况记录到对应数据库下的system.profile集合里。

1. 通过 Mongo Shell 连接实例。

   详情请参考Mongo Shell连接单节点实例、Mongo Shell连接副本集实例、Mongo Shell连接分片集群实例。

2. 通过`use <database>`命令进入指定数据库。

```
use mongodbtest
```

3. 执行如下命令，查看该数据下的慢请求日志。

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

4. 分析慢请求日志，查找引起MongoDB CPU使用率升高的原因。

   以下为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11000000个文档，没有通过索引进行查询。

```
{
        "op" : "query",
        "ns" : "123.testCollection",
        "command" : {
                "find" : "testCollection",
                "filter" : {
                        "name" : "zhangsan"
                },
                "$db" : "123"
        },
        "keysExamined" : 0,
        "docsExamined" : 11000000,
        "cursorExhausted" : true,
        "numYield" : 85977,
        "nreturned" : 0,
        "locks" : {
                "Global" : {
                        "acquireCount" : {
                                "r" : NumberLong(85978)
                        }
                },
                "Database" : {
                        "acquireCount" : {
                                "r" : NumberLong(85978)
                        }
                },
                "Collection" : {
                        "acquireCount" : {
                                "r" : NumberLong(85978)
                        }
                }
        },
        "responseLength" : 232,
        "protocol" : "op_command",
        "millis" : 19428,
        "planSummary" : "COLLSCAN",
        "execStats" : {
                "stage" : "COLLSCAN",
                "filter" : {
                        "name" : {
                                "$eq" : "zhangsan"
                        }
                },
                "nReturned" : 0,
                "executionTimeMillisEstimate" : 18233,
                "works" : 11000002,
                "advanced" : 0,
```

```
                "needTime" : 11000001,
                "needYield" : 0,
                "saveState" : 85977,
                "restoreState" : 85977,
                "isEOF" : 1,
                "invalidates" : 0,
                "direction" : "forward",
....in"
                }
        ],
        "user" : "root@admin"
}
```

通常在慢请求日志中，您需要重点关注以下几点。

- 全表扫描（关键字： COLLSCAN、 docsExamined ）
  - 全集合（表）扫描COLLSCAN 。

    当一个操作请求（如查询、更新、删除等）需要全表扫描时，将非常占用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你的CPU资源。

    说明:

    如果这种请求比较频繁，建议对查询的字段建立索引的方式来优化。
  - 通过查看docsExamined的值，可以查看到一个查询扫描了多少文档。该值越大，请求所占用的CPU开销越大。
- 不合理的索引（关键字： IXSCAN、keysExamined ）

  说明:

  - 索引不是越多越好，索引过多会影响写入、更新的性能。
  - 如果您的应用偏向于写操作，索引可能会影响性能。

  通过查看keysExamined字段，可以查看到一个使用了索引的查询，扫描了多少条索引。该值越大，CPU开销越大。

  如果索引建立的不太合理，或者是匹配的结果很多。这样即使使用索引，请求开销也不会优化很多，执行的速度也会很慢。

  如下所示，假设某个集合的数据，x字段的取值很少（假设只有1、2），而y字段的取值很丰富。

```
{ x: 1, y: 1 }
{ x: 1, y: 2 }
{ x: 1, y: 3 }
......
{ x: 1, y: 100000}
{ x: 2, y: 1 }
{ x: 2, y: 2 }
{ x: 2, y: 3 }
```

```
......
{ x: 1, y: 100000}
```

要实现 {x: 1, y: 2} 这样的查询。

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

关于{y: 1} 与 {y: 1, x: 1} 的区别，可参考MongoDB索引原理及复合索引官方文档。

- 大量数据排序（关键字： SORT、hasSortStage ）

  当查询请求里包含排序的时候， system.profile 集合里的 hasSortStage 字段会为 true 。如果排序无法通过索引满足， MongoDB 会在查询结果中进行排序。而排序这个动作将非常消耗CPU资源，这种情况需要对经常排序的字段建立索引的方式进行优化。

  > 说明:
  > 当您在system.profile集合里发现SORT关键字时，可以考虑通过索引来优化排序。

其他还有诸如建立索引、aggregation（遍历、查询、更新、排序等动作的组合） 等操作也可能非常耗CPU资源，但本质上也是上述几种场景。更多 profiling 的设置请参考profiling官方文档。

您也可以将MongoDB实例接入至混合云数据库管理HDM（Hybrid Cloud Database Management）。在HDM控制台中，您可以对MongoDB实例的实时性能、实时会话、慢日志、磁盘空间等信息进行监控和管理，详情请参见HDM操作流程。

服务能力评估

经过上述分析数据库正在执行的请求和分析数据库慢请求两轮优化之后，整个数据库的查询相对合理，所有的请求都高效地使用了索引。

此时在业务环境使用中还经常遇到CPU资源被占满，那么可能是实例的服务能力已经达到上限了。这种情况下您应当查看监控信息以分析实例资源使用状态；同时对MongoDB数据库进行测试，以便了解在您的业务场景下，当前实例是否满足所需要的设备性能和服务能力。

如您需要升级实例，可以参考变更配置或变更副本集实例节点数进行操作。