阿里云
云数据库 MongoDB 版

最佳实践

文档版本：20190121

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。	 **注意:** 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。
斜体	表示参数、变量。	`bae log list --instanceid` *`Instance_ID`*
[]或者[a|b]	表示可选项，至多选择一个。	`ipconfig` *`[-all|-t]`*
{}或者{a|b}	表示必选项，至多选择一个。	`swich` *`{stand | slave}`*

目录

1 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。

> **注意:**
> 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 注意: 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。

注意:

- 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。

2 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

迁移类型	全量数据迁移
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。

   注意:

   - 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*。

   注意:

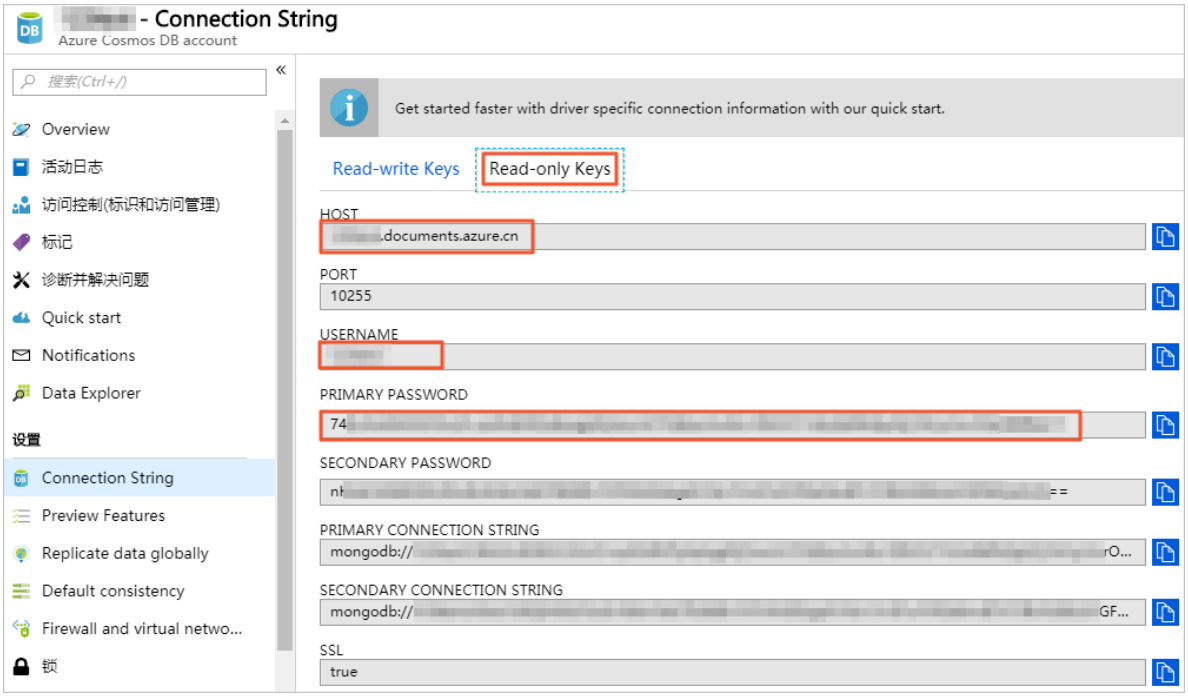
   - 请安装MongoDB3.0以上版本。
   - 该服务器仅作为数据备份与恢复的临时中转平台，迁移操作完成后不再需要。
   - 备份目录所在分区的可用磁盘空间要大于Azure Cosmos DB。

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 **2-1: Azure**连接信息



注意:

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。

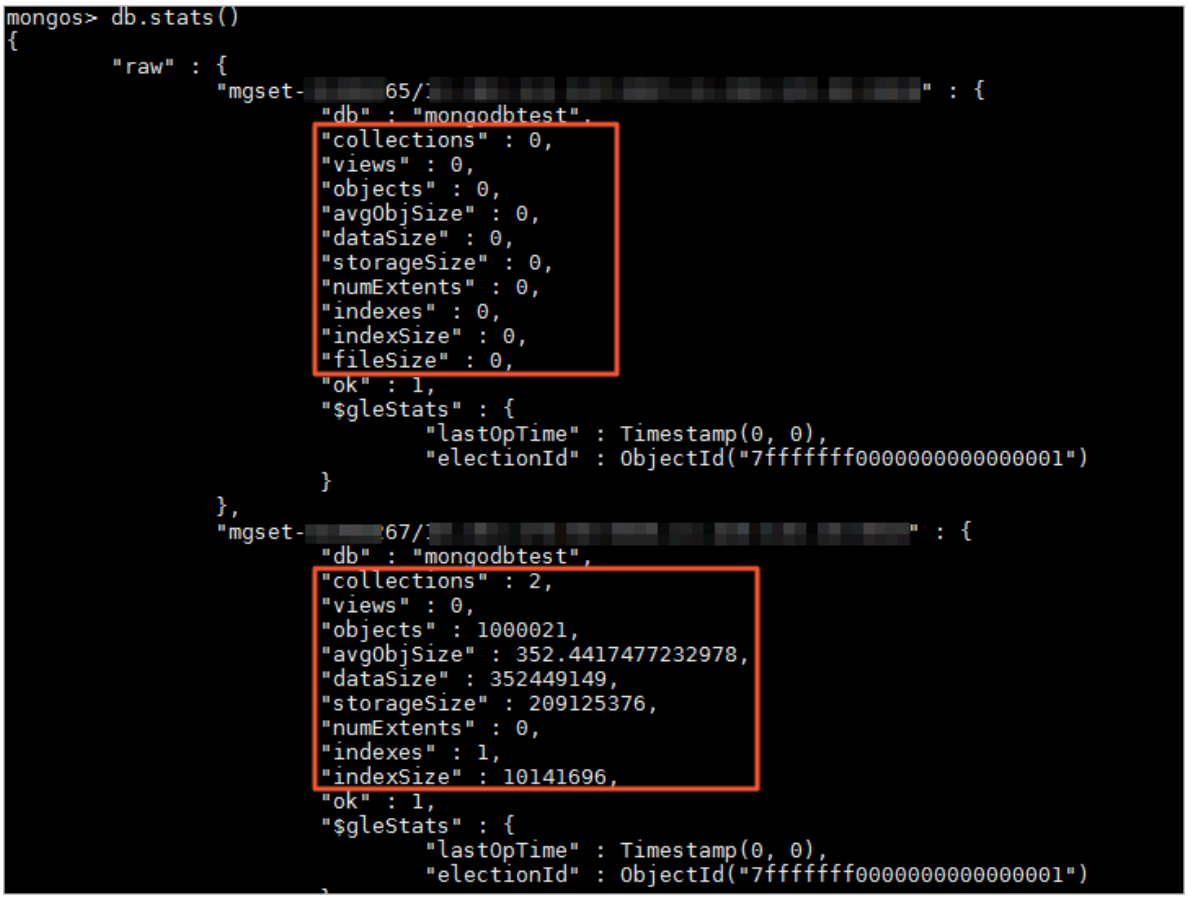
等待数据恢复完成，Azure Cosmos DB API for MongoDB数据库即迁移至阿里云MongoDB数据库中。

3 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数据库的集合设置数据分片，以充分利用各 Shard 节点的存储空间和计算性能。

注意事项

- 该操作仅适用于分片集群实例。
- 进行数据分片操作后，均衡器会对满足条件的现有数据进行分片，这将占用实例的性能，请在业务低峰期操作。
- 分片的片键一经设置后不可修改。
- 分片的片键选取将影响分片集群实例性能，片键的选取可参考如何选择*Shard Key*。
- 若未进行数据分片，数据写入将被集中在 PrimaryShard 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。



操作示例

本文进行操作演示的示例中，数据库为mongodbtest，集合为customer。

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。

   操作示例：

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

   注意:

   您可以通过`sh.status()`查看分片状态。
3. 对集合的某个字段建立索引。

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

   说明：

   - <collection>：集合名。
   - <keys>：包含用于建立索引的字段和排序方式。

     排序方式设置为`1`表示按升序来创建索引，设置为`-1`表示按照降序来创建索引。
   - <options>：表示接收可选参数，详情请参考*db.collection.createIndex()*，本操作示例中暂未使用到该字段。

   操作示例：

   ```
   db.customer.createIndex({"name":1})
   ```
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将根据该值进行数据分片。
   - <value>
     - 1：表示索引升序，通常能很好的支持基于 Shard Key 的范围查询。
     - -1：表示索引降序，通常能很好的支持基于 Shard Key 的范围查询。

- hashed：表示使用Hash分片，通常能将写入均衡分布到各个 Shard 节点中。

。

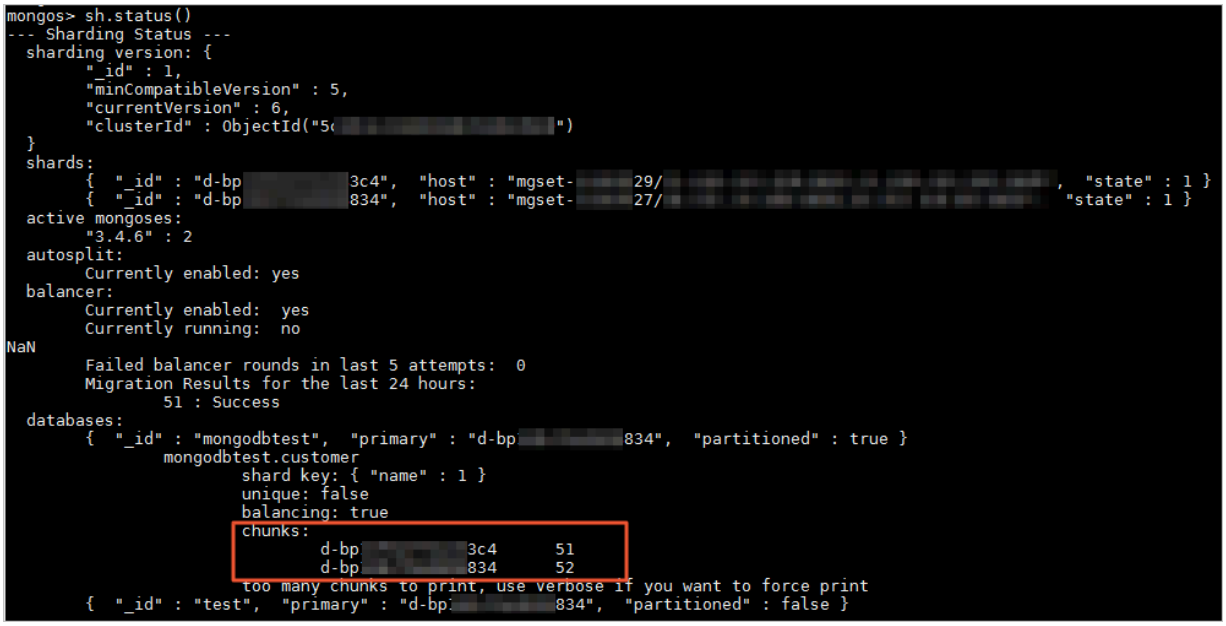
操作示例：

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

如数据库中该集合拥有实际数据，等待后台的均衡器自动执行即可，该过程对用户透明。

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()`，查看数据分片信息和Shard上的块存储信息。



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")
```

4 整理数据库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，将产生很多碎片。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。

注意事项

- 执行该操作前，建议对数据库进行备份。
- 正在进行碎片整理的数据库会被锁定，读写操作将被阻塞。请在业务低峰期操作。
- 该操作不宜频繁执行。

单节点实例/副本集实例操作示例

1. 通过mongo shell连接云数据库的Primary节点，详情请参考*mongoshell*连接实例。
2. 切换至集合所在的数据库。

   ```
   use <database_name>
   ```

   命令说明：

   <database_name>：数据库名。
3. 执行命令来对某个集合进行碎片整理。

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

   命令说明：

   <collection_name>：集合名。

   注意:

   force为可选参数。

   - 值为true时，compact命令才可以在副本集中的主节点上运行 。
   - 如果为false， compact命令在主节点上运行时将返回错误。
4. 等待执行，返回`{ "ok" : 1 }`代表执行完成。

   注意:

   compact操作不会传递给Secondary节点，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。

分片集群实例操作示例

1. 通过mongo shell连接到分片集群实例中的任一mongos节点，详情请参考*mongo shell*连接分片集群实例。
2. 执行命令对Shard节点中的Primary节点合进行集合的碎片整理。

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

   命令说明：

   <Shard ID>：Shard节点ID。

   <collection_name>：集合名。
3. 执行命令对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
   '}}})
   ```

   命令说明：

   <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。

5 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。

关闭 Balancer 功能

云数据库MongoDB的 Balancer 功能默认是开启状态。特殊业务场景下需要关闭，请参考下述操作步骤。

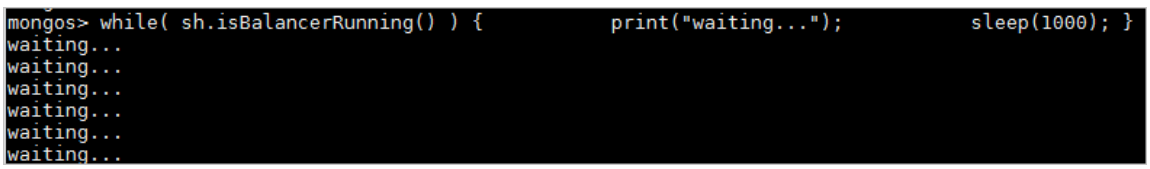
1. 通过 *Mongo Shell* 登录数据库。
2. 在 mongos 节点命令窗口中，切换至 config 数据库。

```
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()
```

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)
```

设置 Balancer 的活动窗口

为避免 Balancer 执行块迁移操作影响您的业务，您可以通过设置 Balancer 的活动窗口，让 Blancer 在指定的时间段工作。

> 注意:
> 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
```

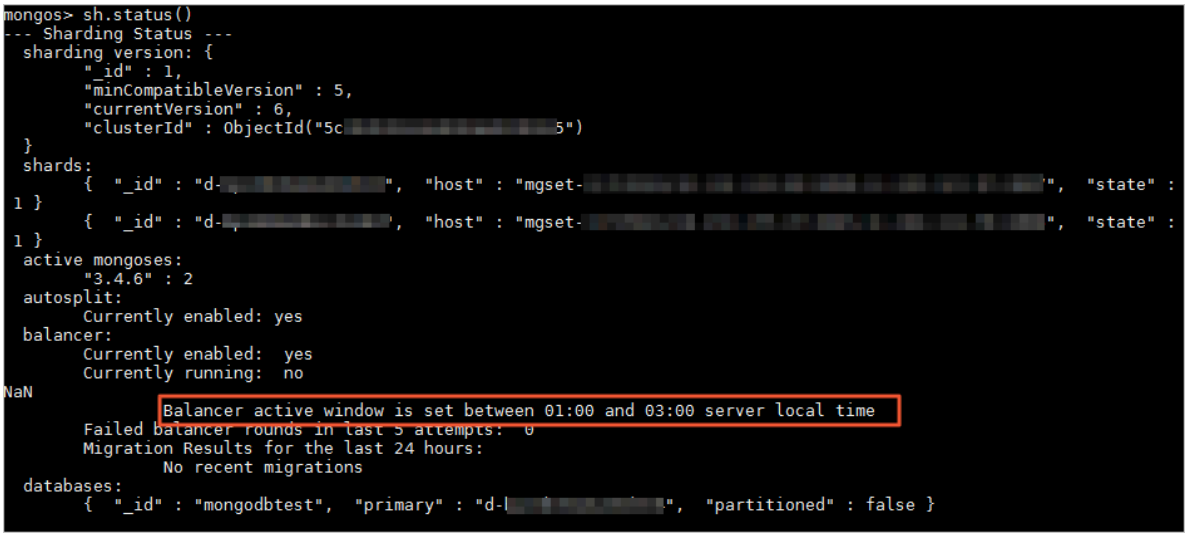
3. 执行如下命令设置 Balancer 的活动窗口。

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

> 注意:
>
> - <start-time>：开始时间，时间格式为 HH:MM，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，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 } })
```

6 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。

注意:

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'}}})
```

7 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。

注意:

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。

> **注意:**
>
> 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：

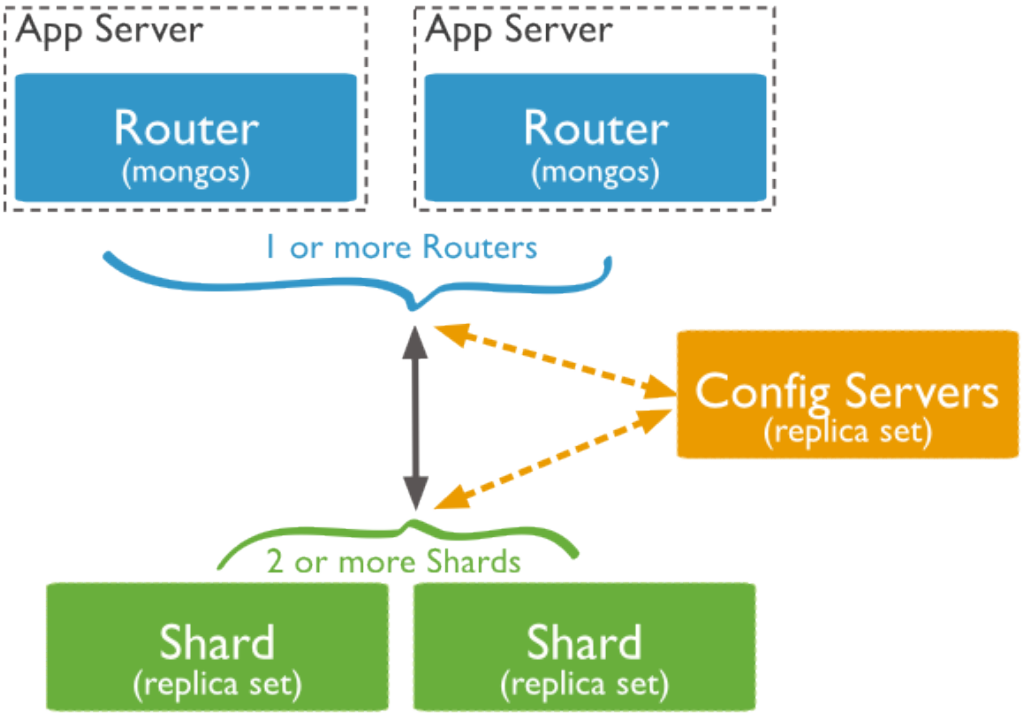
```
mongodb://root:xxxxxxxx@dds-xxxxxxxxxxxx:3717,xxxxxxxxxxxx:3717/
admin?replicaSet=mgset-xxxxxx&readPreference=secondaryPreferred
```

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。

8 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]]
```

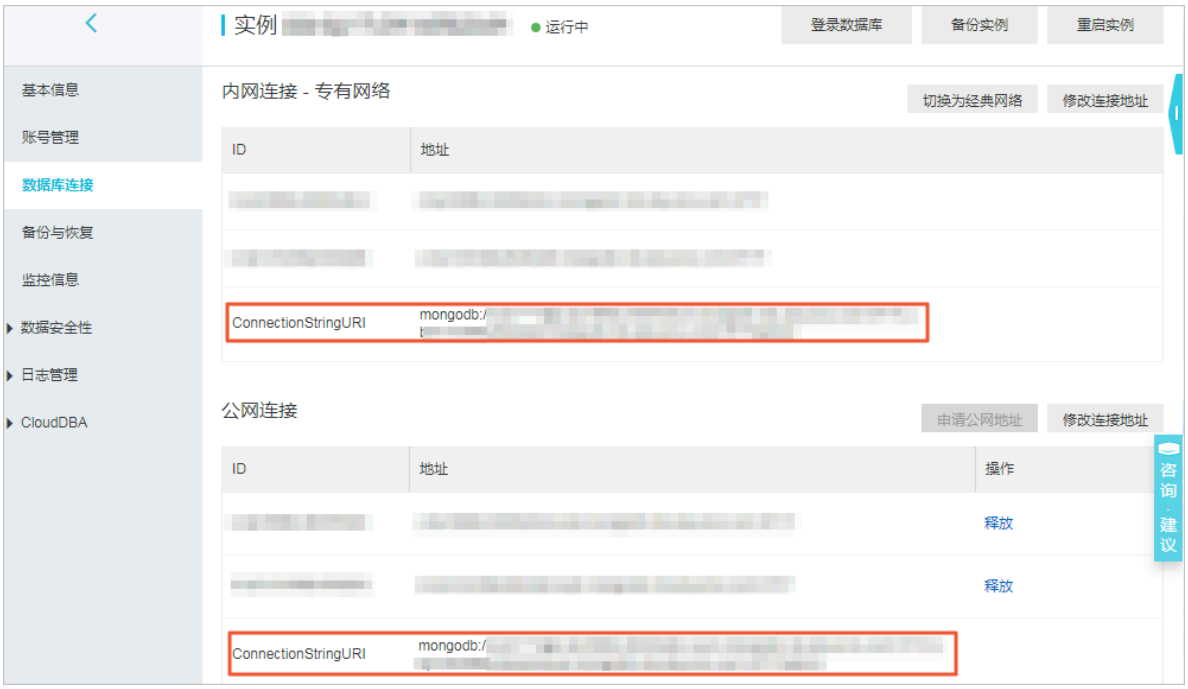
注意:

- `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-m5e80a9241323604.mongodb.rds.aliyuncs.com:3717,s-
m5e053215007f404.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");
```

 **注意:**

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。

 **注意:**

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。

9 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使用率很高的问题， 本文主要帮助您从应用的角度排查该问题。

分析数据库正在执行的请求

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. 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 opid 。 **注意:** 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	请求运行的时间，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	请求运行的时间，单位为毫秒。如果该字段返回的值特别大，需要查看请求是否合理。
query	正在执行的操作。
ns	该操作目标集合。
locks	跟锁相关的参数，请参考官方文档，本文不做详细介绍。 **注意:** db.currentOp 文档请参见：*db.currentOp*。

说明：

通过`db.currentOp()`查看正在执行的操作，可以查看是否有不正常的耗时请求正在执行。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，导致您的业务响应很慢，此时需要重点关注执行时间非常耗时的操作。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， 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。

分析数据库慢请求

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

操作步骤

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()
```

常见分析场景

- 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。

    注意:

    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 ）

  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。

  注意:

  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。

注意:

更多 profiling 的设置请参考*profiling*官方文档。

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。