ALIBABA CLOUD

Alibaba Cloud

云数据库 MongoDB 版

产品简介

文档版本：20220704

阿里云

法律声明

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

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

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

目录

1.什么是云数据库MongoDB版

云数据库MongoDB版（ApsaraDB for MongoDB）完全兼容MongoDB协议，基于飞天分布式系统和高可靠存储引擎，提供多节点高可用架构、弹性扩容、容灾、备份恢复、性能优化等功能。

MongoDB的数据结构

MongoDB是面向文档的NoSQL（非关系型）数据库，它的数据结构由字段（Field）和值（Value）组成，类似于JSON对象，示例如下：

```
{
    name:"张三",
    sex:"男性",
    age:30
}
```

MongoDB的存储结构

MongoDB的存储结构区别于传统的关系型数据库，由如下三个单元组成：

- 文档（Document）：MongoDB中最基本的单元，由BSON键值对（key-value）组成。相当于关系型数据库中的行（Row）。
- 集合（Collection）：一个集合可以包含多个文档，相当于关系型数据库中的表格（Table）。
- 数据库（Database）：等同于关系型数据库中的数据库概念，一个数据库中可以包含多个集合。您可以在MongoDB中创建多个数据库。

为什么选择云数据库MongoDB版

详情请参见产品优势和应用场景。

产品定价

详情请参见收费项目及价格说明。

专属集群

当前专属集群MyBase已支持MongoDB引擎，您可以以MyBase的形态购买MongoDB实例。详情请参见什么是云数据库专属集群MyBase。

部署建议

您可以从以下维度考虑如何创建并使用MongoDB实例：

- 地域和可用区

  地域指阿里云的数据中心，可用区是指在同一地域内，电力和网络互相独立的物理区域。地域和可用区决定了MongoDB实例所在的物理位置，一旦成功创建MongoDB实例后将无法更换地域。更多详情，请参见地域和可用区。

  您可以从用户地理位置、阿里云产品发布情况、应用可用性以及是否需要内网通信等因素选择地域和可用区。例如，您的应用部署在云服务器ECS（Elastic Compute Service）上，需要使用MongoDB实例作为该应用的数据库，那么在创建MongoDB实例时，应当选择与ECS实例相同的地域和可用区。

> **说明**
>
> 同一可用区内的ECS实例和MongoDB实例通过内网连接时，网络延时最小。

- 网络规划

  阿里云推荐您使用专有网络VPC，您可自行规划私网IP地址段。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，您可以使用默认的专有网络，也可以自行事先创建，详情请参见新建实例场景下配置专有网络。

- 安全方案

  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、透明数据加密TDE等多手段保障数据库数据安全。详情请参见云数据库MongoDB版数据安全最佳实践。

如何使用云数据库MongoDB版

您可以通过以下方式管理MongoDB实例，进行实例创建、网络设置、数据库创建、账号创建等操作：

- 控制台：提供图形化的Web界面，操作方便。
- API：控制台上所有的操作都可以通过API实现。

创建MongoDB实例后，您可以通过以下方式访问MongoDB实例：

- Mongo Shell：MongoDB官方命令行工具，您可以通过Mongo Shell连接MongoDB副本集实例，对数据库进行管理操作。
- 客户端：云数据库MongoDB版完全兼容MongoDB协议，您可以使用通用的数据库客户端工具访问MongoDB实例。例如Robo 3T、Studio 3T等。

相关服务

- ECS：云服务器ECS（Elastic Compute Service）通过内网访问同一地域的MongoDB实例时，可实现最佳性能。ECS搭配MongoDB实例是典型的业务访问架构。
- DTS：您可以使用数据传输服务DTS（Data Transmission Service）将本地MongoDB数据库迁移上云。
- OSS：对象存储服务OSS（Object Storage Service）是阿里云提供的海量、安全、低成本、高可靠的云存储服务。
- DLA：云原生数据湖分析DLA（Data Lake Analytics）是新一代大数据解决方案，采取计算与存储完全分离的架构，提供弹性的Serverless SQL与Serverless Spark服务，满足在线交互式查询、流处理、批处理、机器学习等诉求。相关链接：
  - Serverless SQL对接MongoDB快速入门
  - Serverless Spark对接MongoDB快速入门

2.功能特性

云数据库MongoDB版（ApsaraDB for MongoDB）完全兼容MongoDB协议，基于飞天分布式系统和高可靠存储引擎，提供多节点高可用架构、弹性扩容、容灾、备份回滚、性能优化等解决方案。本文介绍云数据库MongoDB版的功能特性。

架构灵活

云数据库MongoDB版支持灵活的部署架构，提供的实例架构包括单节点架构、副本集架构以及分片集群架构，满足不同的业务场景。

弹性扩容

根据业务需求，您可以变更实例配置，即变更实例规格、存储空间、节点数量。同时也支持您设置变更配置的生效时间，尽量将生效时间设置在业务低峰期，避免在变更配置过程中对业务造成影响。详情请参见变更配置方案概览。

数据安全

安全技术	说明
DDoS防护	在网络入口实时监测，当发现超大流量攻击时，对源IP进行清洗，清洗无效情况下可以触发黑洞机制。
IP访问白名单	提供对实例进行IP访问过滤功能，实现高等级的访问安全保护，IP白名单最多可配置1000条，详情请参见设置白名单。
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。
数据容灾	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。 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。 **说明** 迁移可用区仅支持MongoDB 4.2及以下版本且未开启TDE功能的副本集实例。
SSL加密	在传输层对网络连接进行加密，提升通信数据安全性的同时，保证数据的完整性，详情请参见设置SSL加密。
透明数据加密TDE	对数据文件执行实时I/O加密和解密，数据在写入磁盘之前进行加密，从磁盘读入内存时进行解密。TDE不会增加数据文件的大小，您无需更改任何应用程序，即可使用TDE功能，详情请参见设置透明数据加密TDE。
自动备份	支持设置备份策略，您可以根据业务低峰时段灵活配置备份时间。
临时备份	支持手动备份MongoDB数据，支持物理备份和逻辑备份。

安全技术	说明
数据恢复	通过备份文件，您可以进行数据恢复。具体请参见： • 按备份点将备份数据恢复至新建实例 • 按时间点将备份数据恢复至新建实例 • 恢复备份数据至当前实例
备份文件下载	您可以在备份保留天数期限内下在备份文件至本地，具体请参见下载备份文件。

全面监控

云数据库MongoDB提供多达20种系统性能监控项，包括磁盘容量、IOPS、连接数、CPU使用率、网络流量、TPS、QPS、缓存命中率等，详情请参见基本监控。

专业工具支持

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

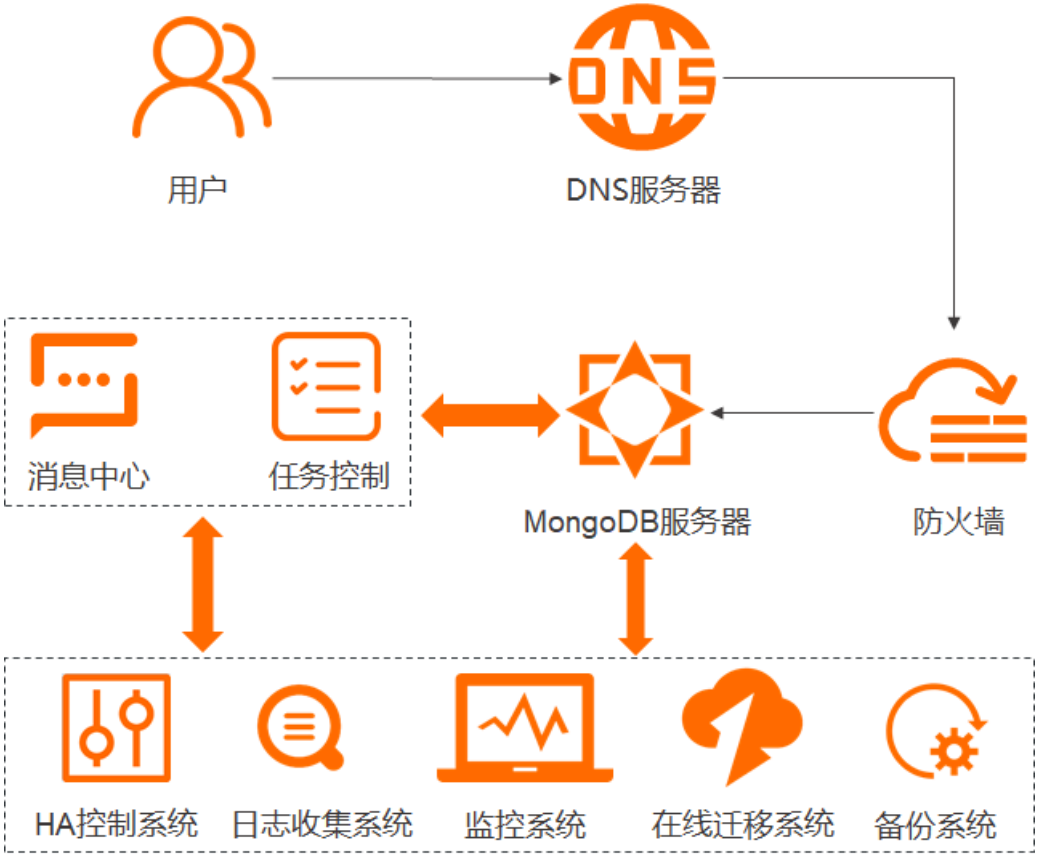
数据传输服务DTS（Data Transmission Service）是阿里云提供的一种支持关系型数据库、NoSQL、OLAP等多种数据源之间数据交互的数据服务。它提供了数据迁移、实时数据订阅及数据实时同步等多种数据传输能力。通过数据传输可实现不停服数据迁移、数据异地灾备、跨境数据同步、缓存更新策略等多种业务应用场景，助您构建安全、可扩展、高可用的数据架构。

3.产品架构

3.1. 云数据库MongoDB产品架构

本文详细介绍云数据库MongoDB的产品架构和组件。

产品架构图



主要组件说明

- 任务控制

  MongoDB实例支持多种管理控制任务，如创建实例、变更配置以及备份实例等。任务系统会根据您下发的操作指令，进行灵活控制并进行任务跟踪及出错管理。

- HA控制系统

  高可用探测模块，用于探测MongoDB实例的运行状况。如果判断主节点实例不可用，则切换主备节点操作并通知用户，您也可以手动切换主备节点，保障MongoDB实例的高可用。详情请参见切换节点角色。

- 日志收集系统

  收集MongoDB运行情况的日志信息，包括实例慢操作记录日志、审计日志等。详情请参见日志管理概览和开通审计日志。

- 监控系统

  收集MongoDB实例的性能监控信息，包括基础指标、磁盘容量、网络请求以及操作次数等核心信息。详情请参见基本监控。

- 备份系统

  针对MongoDB实例进行备份处理，将生成的备份文件存储至OSS（Object Storage Service）中。目前MongoDB备份系统支持用户自定义备份策略的自动备份和手动备份，保存7天内的备份文件。详情请参见自动备份MongoDB数据和手动备份MongoDB数据。

- 在线迁移系统

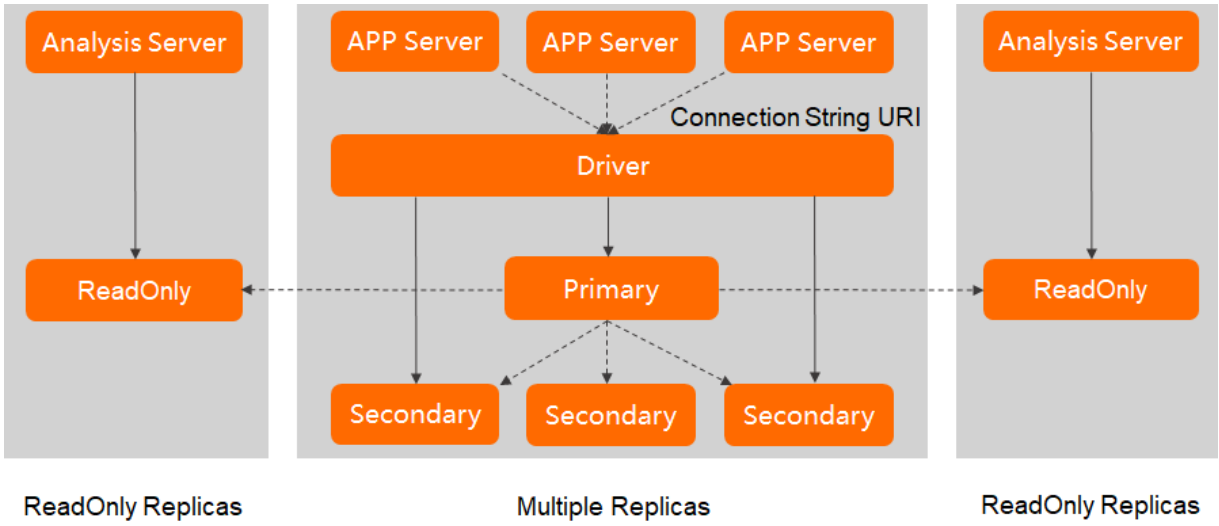
  当实例所运行的物理机出现故障，在线迁移系统会根据备份系统中的备份文件重新搭建实例，保障业务不受影响。详情请参见数据迁移和同步方案概览。

3.2. MongoDB只读节点简介

为了扩展主从节点的读请求能力，云数据库MongoDB提供具备独立连接地址的只读节点，适合独立系统直连访问，以减轻大量读请求给主从节点造成的压力。

在有大量读请求的应用场景下，数据库的主从节点可能难以承受读取压力，甚至对业务造成影响。为了分担主从节点的读取压力，您可以根据业务需求创建一个或多个只读节点，来满足大量的数据读取需求，增加应用的吞吐量。

只读节点架构



只读节点（ReadOnly）的特点如下：

- 只读节点（ReadOnly）通过操作日志（oplog）从延迟最低的主节点（Primary）或从节点（Secondary）同步数据，应用于有大量读请求的场景，以减轻主节点（Primary）和从节点（Secondary）的访问压力。
- 只读节点（ReadOnly）具有独立的连接地址，适合独立系统（如Analysis Server）直连访问，与已有主从节点的连接互不干扰。
- 两个或以上只读节点（ReadOnly）可以使用ReadOnly Connection String URI实现读请求负载均衡。

只读节点与从节点的区别

节点	说明	适用场景

节点	说明	适用场景
只读节点（Read Only）	• 只读节点具有高可用保障，即某个只读节点故障时，系统会自动将其与隐藏节点切换，若未自动切换，您可以自行切换，只读节点的连接地址保持不变。 **说明** 如何切换只读节点和隐藏节点，请参见切换节点角色。 触发节点的角色切换后，会产生1次30秒内的连接闪断，建议您在业务低峰期操作或确保应用具备重连机制。 • 只读节点具有独立的连接地址，适合独立系统及应用直连访问，与已有主从节点的连接互不干扰。 • 只读节点不在“主节点的备用列表”中，不会被选举为主节点，也不会参与投票选举主节点。	两个或以上只读节点可以使用ReadOnly Connection String URI实现读请求负载均衡，适用于从现有实例中读取大量数据的业务场景，如BI分析、大数据分析等。
从节点（Secondary）	• 从节点具有高可用保障，即某个从节点故障时，系统会自动将其与隐藏节点切换，若未自动切换，您可以自行切换，从节点的连接地址保持不变。 **说明** 如何切换从节点和隐藏节点，请参见切换节点角色。 触发节点的角色切换后，会产生1次30秒内的连接闪断，建议您在业务低峰期操作或确保应用具备重连机制。 • 从节点故障时，不会被选举为主节点。 • 主节点故障时，每个从节点都有可能被选举为新的主节点，来执行和响应数据的读写请求。	主从节点可以使用Connection String URI实现读写分离，适用于读多写少的并发场景，从而在性能扩展的同时，避免节点故障对业务带来的影响。

功能优势

- 您可以根据业务需求随时更改只读节点个数，节省业务成本。
- 只读节点具有独立的连接地址，适合独立系统及应用直连访问，与已有主从节点的连接互不干扰。
- 只读节点与主从节点采用一致规格，自动从延迟最低的主节点或从节点同步数据，免去维护的烦恼。
- 独立的只读节点提供只读服务，不占用主节点的资源。增减只读节点的操作，不会对主从节点的业务构成干扰，也不会中断主从节点的连接访问。
- 云数据库MongoDB提供统一的只读地址连接所有只读节点，您只需添加只读节点的个数即可扩展数据库的处理能力，应用程序无需做任何修改。

功能限制

- 目前仅MongoDB副本集实例和分片集群实例支持只读节点。
- MongoDB实例版本需为3.4、4.0及4.2版本。

> **说明** 实例小版本需升级至最新版本，如何升级小版本，请参见升级数据库小版本。

- 只读节点仅供读请求访问，不参与主从节点选举。
- 一个副本集实例可以添加最多5个只读节点。
- 一个分片集群实例中的每个Shard可以添加最多5个只读节点。
- 只读节点与主节点或从节点之间的数据复制方式为异步复制，正常情况下存在毫秒级的延迟，在主节点写入压力大的时候可能出现秒级的延迟。

价格

单个只读节点的价格等同于副本集实例或者分片集群实例Shard中单个节点的价格。

3.3. 单节点架构

云数据库MongoDB的单节点架构是阿里云为用户提供的一种高性价比部署架构，适用于存储企业非核心数据的场景，例如开发、测试、学习、培训等。

注意事项

单节点架构的故障恢复时间较长，无SLA保障。

使用限制

- 云数据库MongoDB仅支持在以下地域的可用区创建单节点实例。
  - 华东1（杭州）：杭州可用区G、H和I。
  - 华东2（上海）：上海可用区B和G。
  - 华北1（青岛）：青岛可用区C。
  - 华北2（北京）：北京可用区F和H。
  - 华南1（深圳）：深圳可用区E。
  - 新加坡：新加坡可用区A、B和C。
- 仅MongoDB 4.0和MongoDB 3.4版本支持单节点架构，您可以根据业务需求创建对应版本的单节点实例。如何创建单节点实例，请参见创建单节点实例。

单节点架构

- 单节点架构仅提供一个StandAlone节点，用于读写数据。
- 单节点架构以较低的价格享受云数据库MongoDB提供的运维支持和内核优化服务，您可以根据各类业务场景的差异选择对应的规格配置，降低成本支出。更多规格信息，请参见单节点实例规格表。

常见问题

- 问：单节点架构是否提供高可用？

  答：不提供。单节点架构只有一个副本，极端情况下如发生故障会造成30分钟左右服务不可用状态，建议您在生产环境中使用副本集架构或分片集群架构。

- 问：单节点实例是否支持增量数据迁移与同步、按时间点创建实例恢复数据功能？

  答：不支持。云数据库MongoDB单节点实例默认不支持操作日志（Oplog），所以不支持增量数据迁移与同步、按时间点创建实例恢复数据。

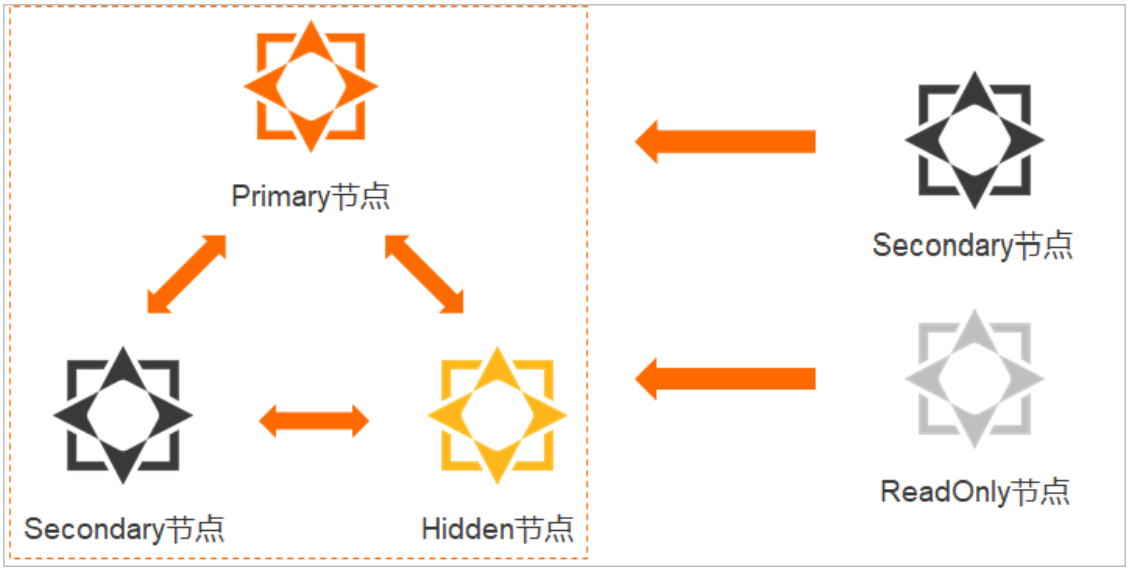
相关文档

- 副本集架构
- 分片集群架构
-

3.4. 副本集架构

云数据库MongoDB版自动搭建好副本集，您可以直接操作副本集中的主节点和从节点。容灾切换、故障迁移等高级功能为您整体打包好，实例使用过程中对您完全透明。

副本集架构



MongoDB副本集架构通过部署多种节点来达到高可用和读写分离的效果，每个副本集实例包含一个主节点（Primary节点）、一个或多个从节点（Secondary节点）、一个隐藏节点（Hidden节点）和可选的一个或多个只读节点（ReadOnly节点）。其中主节点、从节点和隐藏节点合起来统称为“主备节点”。各节点的说明如下：

节点	功能	说明
主节点（Primary节点）	负责执行和响应数据读写请求。	每个副本集实例中只能有一个主节点。

节点	功能	说明
从节点（Secondary节点）	通过操作日志（oplog）同步主节点的数据，可在主节点故障时通过选举成为新的主节点，保障高可用。	• 通过从节点的连接地址进行连接时，只能读取数据不能写入数据。 • 从节点具有高可用保障，即某个从节点故障时，系统会自动将其与隐藏节点切换，若未自动切换，您可以自行切换，从节点的连接地址保持不变。 **说明** 如何切换从节点和隐藏节点，请参见切换节点角色。 触发节点的角色切换后，会产生1次30秒内的连接闪断，建议您在业务低峰期操作或确保应用具备重连机制。
隐藏节点（Hidden节点）	通过操作日志（oplog）同步主节点的数据，可在从节点故障时接替该故障节点成为新的从节点，也可在只读节点故障时接替该故障节点成为新的只读节点，保障高可用。	• 隐藏节点仅用作高可用，对客户端不可见。 • 隐藏节点不在“主节点的备用列表”中，不会被选举为主节点，但会参与投票选举主节点。 • 每个副本集实例中只能有一个隐藏节点。
只读节点（ReadOnly节点）	通过操作日志（oplog）从延迟最低的主节点或从节点同步数据，应用于有大量读请求的场景，以减轻主节点和从节点的访问压力。两个或以上只读节点可以使用ReadOnly Connection String URI连接实现读请求负载均衡。 **说明** 更多信息，请参见MongoDB只读节点简介。	• 只读节点具有高可用保障，即某个只读节点故障时，系统会自动将其与隐藏节点切换，若未自动切换，您可以自行切换，只读节点的连接地址保持不变。 **说明** 如何切换只读节点和隐藏节点，请参见切换节点角色。 触发节点的角色切换后，会产生1次30秒内的连接闪断，建议您在业务低峰期操作或确保应用具备重连机制。 • 只读节点具有独立的连接地址，适合独立系统直连访问，与已有主从节点的连接互不干扰。 • 只读节点不在“主节点的备用列表”中，不会被选举为主节点，也不会参与投票选举主节点。

扩展副本集节点

云数据库MongoDB提供扩展节点功能，您可以按照业务需求增加从节点或只读节点的数量，详情请参见变更副本集实例配置。

说明 每个副本集实例中仅包含一个隐藏节点，扩展节点时仅增加从节点或只读节点，不会增加隐藏节点。

例如：某个业务场景下对数据库有更高读取性能需求，如阅读类网站、订单查询系统等读多写少场景或有临时活动等突发业务需求，按需增加从节点或只读节点来弹性调整实例的读取性能。

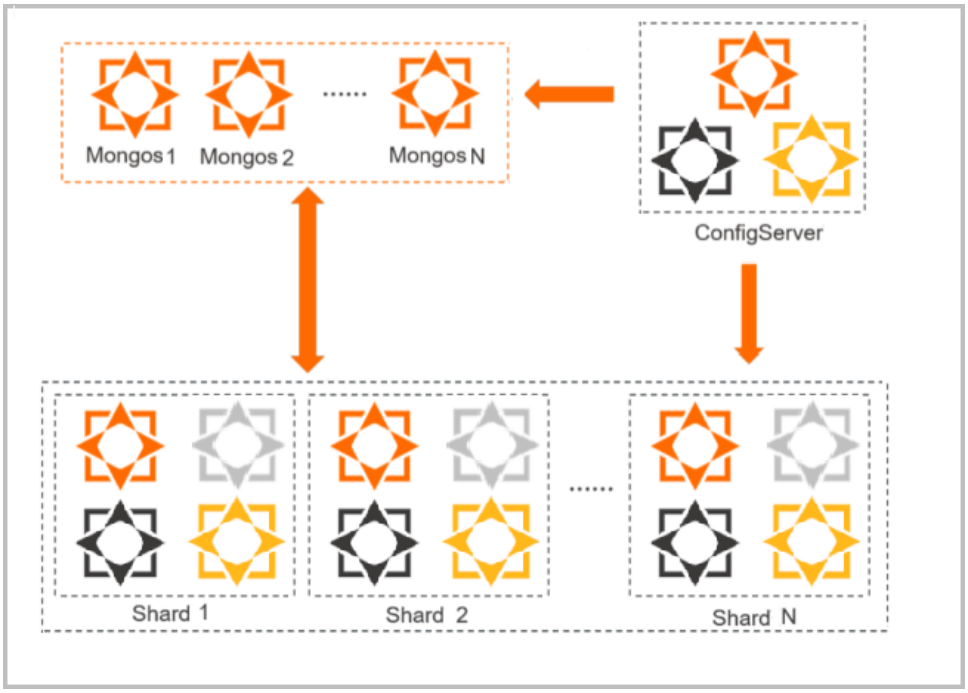
相关文档

- 单节点架构
- 分片集群架构

3.5. 分片集群架构

分片集群架构提供Mongos、Shard和ConfigServer三种组件。您可以自由选择Mongos和Shard的个数和配置，创建具备不同服务性能的MongoDB分片集群实例。

分片集群架构



分片集群架构的各组件说明如下：

组件名称	架构	说明
Mongos	单节点架构	负责将读写操作路由到对应Shard中。 您可以通过购买多个Mongos实现负载均衡及故障转移。单个分片集群实例默认支持最多32个Mongos，如果业务需要32个以上的Mongos，您可以提交工单申请。
Shard	副本集架构（主备三节点（主节点、从节点和隐藏节点）+只读节点）	负责存储数据库数据。 您可以通过购买多个Shard来横向扩展实例的数据存储和读写并发能力。单个分片集群实例默认支持最多32个Shard，如果业务需要32个以上的Shard，您可以提交工单申请。

组件名称	架构	说明
ConfigServer	副本集架构（三节点）	负责存储Shard的元数据，即各Shard中包含哪些数据。 ConfigServer为固定配置，暂不支持修改。配置如下： • ConfigServer规格：**1核2GB（通用型）**。 • ConfigServer存储空间：20 GB。

相关文档

- 单节点架构
- 副本集架构

4.产品规格

4.1. 实例规格族

本文介绍云数据库MongoDB版的通用型、独享型和专属型实例规格族。

规格族

规格族	说明	适用场景
通用型	• 独享被分配的内存和存储空间，共享CPU和I/O，存在资源争抢的风险。 • 通过资源复用换取利用率最大化，性价比较高，享受规模红利。	• 对价格敏感的客户。 • 对性能稳定性要求较低的业务场景。
独享型	• 本地盘版：独享被分配的内存、存储空间和CPU，共享I/O，仍有低概率存在I/O争抢情况。 • 云盘版：独享被分配的内存、存储空间、CPU和I/O，不存在资源争抢情况。与本地盘版相比，云盘版不会受物理机上其他实例的影响，性能长期稳定。 • 独占物理机：完全独占一台物理机的所有资源，不存在资源争抢情况。	以数据库为核心系统的业务场景，例如金融、电商、政务、大中型互联网业务等。

通用型、独享型（本地盘版）和独享型（云盘版）的区别如下图所示。


通用型
MongoDB 1
内存
存储空间
MongoDB 2
内存
存储空间
MongoDB 3
内存
存储空间
共享CPU和I/O
独享型（本地盘版）
MongoDB 1
内存
CPU
存储空间
MongoDB 2
内存
CPU
存储空间
MongoDB 3
内存
CPU
存储空间
共享I/O
独享型（云盘版）
MongoDB 1
内存
CPU
I/O
存储空间
MongoDB 2
内存
CPU
I/O
存储空间
MongoDB 3
内存
CPU
I/O
存储空间

实例规格表

各种实例规格族对应提供了一种或多种实例规格（包含CPU核数、内存、存储空间、最大连接数和IOPS），详细信息请参见实例规格表实例规格概述。

价格

关于各实例规格的价格，详细信息请参见收费项目及价格说明。

变更规格

您可以按需变更实例规格，操作方法请参见变更配置方案概览。

4.2. 实例规格表

4.2.1. 实例规格表

本文介绍云数据库MongoDB支持的实例规格的详细信息。本文介绍云数据库MongoDB支持的实例规格的详细信息。

关于云盘IOPS

本地SSD盘的IOPS与实例规格绑定。ESSD云盘的IOPS与存储空间大小成正比，同时ECS的实例规格也会制约IOPS上限，详情请参见本文规格表。

ESSD云盘实例的公式说明：

```
min{1800+50*存储空间, 50000, ECS限制IOPS}
```

上述公式中各值的说明如下：

- `1800+50*存储空间`：ESSD云盘IOPS的基础计算公式。
- `50000`：ESSD云盘性能级别（Performance Level）PL1限定的单盘最大IOPS。更多信息，请参见ESSD云盘。
- `ECS限定IOPS`：ECS实例规格限定的最大IOPS。每个规格对应的最大IOPS请参见本文规格表。

实例的IOPS取上述公式三个值中的最小值。例如：

- 某实例的存储空间为20GB，套用上面的公式1800+50*20，得出结果为2800，则该实例的IOPS为2800。
- 某实例的规格为`mdb.shard.2x.xlarge.d`，限定最大IOPS为21000。当实例的存储空间为6000GB时，套用计算公式的结果为301800，超出了ECS实例与PL1限定的最大值。因此该实例的IOPS为21000。

> **说明** 吞吐量如果达到上限，也会影响IOPS。更多信息，请参见全新一代企业级实例规格族存储I/O性能表。

规格类型说明

根据实例规格类型的不同，实际使用过程中可能无法达到现行实例规格表中给出的最大连接数和最大IOPS数。各规格类型对应的说明如下：

规格类型	说明	是否承诺最大连接数	是否承诺最大IOPS
独享型云盘版	完全独享CPU、内存、存储介质以及I/O资源。	是	是
独占物理机	完全独享CPU、内存、存储介质以及I/O资源。	是	是
独享型本地盘版	独享CPU和内存，与同一物理机中的其他用户共享I/O资源。	是	否
通用型	独享内存，与同一物理机中的其他用户共享CPU和I/O资源。	是	否

说明 更多信息，请参见实例规格族。

现行实例规格表

说明

- 实例规格定义的内存包括MongoDB相关管理服务、数据库服务和底层操作系统占用的内存（如BIOS预留内存、内核和Hypervisor运行内存等），因此，您查看的可用内存会小于实例规格定义的内存。
- 因物理硬件资源的迭代演进，2017年7月10日起，新购及变更实例规格后，实例将采用以下表格中的新规格。

MongoDB 5.0和MongoDB 4.4版本副本集实例规格表

实例类型	规格类型	规格信息	规格代码	最大连接数	最大IOPS	存储空间
		4核8GB	mdb.shard.2x.xlarge.d	3000	min{1800+50*存储空间, 21000}	
		8核16GB	mdb.shard.2x.2xlarge.d	5000	min{1800+50*存储空间, 26250}	
		16核32GB	mdb.shard.2x.4xlarge.d	8000	min{1800+50*存储空间, 42000}	
		32核64GB	mdb.shard.2x.8xlarge.d	16000	min{1800+50*存储空间, 50000}	
		2核8GB	mdb.shard.4x.large.d	3000	min{1800+50*存储空间, 10500}	
		4核16GB	mdb.shard.4x.xlarge.d	5000	min{1800+50*存储空间, 21000}	

实例类型	规格类型	规格信息	规格代码	最大连接数	最大IOPS	存储空间
副本集实例	独享型云盘版	8核32GB	mdb.shard.4x.2xlarge.d	8000	min{1800+50*存储空间, 26250}	20~16000GB
		16核64GB	mdb.shard.4x.4xlarge.d	16000	min{1800+50*存储空间, 42000}	
		32核128GB	mdb.shard.4x.8xlarge.d	16000	min{1800+50*存储空间, 50000}	
		2核16GB	mdb.shard.8x.large.d	5000	min{1800+50*存储空间, 10500}	
		4核32GB	mdb.shard.8x.xlarge.d	8000	min{1800+50*存储空间, 21000}	
		8核64GB	mdb.shard.8x.2xlarge.d	16000	min{1800+50*存储空间, 26250}	
		16核128GB	mdb.shard.8x.4xlarge.d	16000	min{1800+50*存储空间, 42000}	
		32核256GB	mdb.shard.8x.8xlarge.d	16000	min{1800+50*存储空间, 50000}	

单节点、副本集实例规格表

实例类型	规格类型	规格信息	规格代码	最大连接数	最大IOPS	存储空间
副本集实例	通用型	1核2GB	dds.mongo.mid	500	8000	10~2000GB
		2核4GB	dds.mongo.standard	1000	8000	
		4核8GB	dds.mongo.large	3000	8000	
		8核16GB	dds.mongo.xlarge	5000	8000	
		8核32GB	dds.mongo.2xlarge	8000	14000	
		16核64GB	dds.mongo.4xlarge	16000	16000	
		2核16GB	mongo.x8.medium	2500	4500	250~3000GB
		4核32GB	mongo.x8.large	5000	9000	500~3000GB

实例类型	规格类型	规格信息	规格代码	最大连接数	最大IOPS	存储空间
	独享型本地盘版	8核64GB	mongo.x8.xlarge	10000	18000	1000~3000GB
		16核128GB	mongo.x8.2xlarge	20000	36000	• 包年包月：2000~3000GB • 按量付费：2000GB
		32核256GB	mongo.x8.4xlarge	40000	72000	2000~3000GB
	独占物理机	60核440GB	dds.mongo.2xmonopolize	100000	100000	2000~6000GB
		90核660GB	dds.mongo.3xmonopolize	100000	100000	2000~6000GB
单节点实例	通用型	4核8GB	dds.sn2.large.1	6000	min{30 * 存储空间，20000}	20~2000GB
		4核16GB	dds.sn4.xlarge.1	8000		
		8核16GB	dds.sn2.xlarge.1	8000		
		8核32GB	dds.sn4.2xlarge.1	10000		

分片集群实例规格表

节点类型	规格类型	规格信息	规格代码	最大连接数	最大IOPS	存储空间
Mongos	通用型	1核2GB	dds.mongos.mid	1000	无	无
		2核4GB	dds.mongos.standard	2000		
		4核8GB	dds.mongos.large	4000		
		8核16GB	dds.mongos.xlarge	8000		
		8核32GB	dds.mongos.2xlarge	16000		
		16核64GB	dds.mongos.4xlarge	16000		
		1核2GB	dds.shard.mid		1000	
		2核4GB	dds.shard.standard		2000	

节点类型	规格类型	规格信息	规格代码	最大连接数	最大IOPS	存储空间
Shard	通用型	4核8GB	dds.shard.large	无	4000	10~2000 GB
		8核16GB	dds.shard.xlarge		8000	
		8核32GB	dds.shard.2xlarge		14000	
		16核64GB	dds.shard.4xlarge		16000	
	独享型本地盘版	2核16GB	dds.shard.sn8.xlarge.3		4500	10~250GB
		4核32GB	dds.shard.sn8.2xlarge.3		9000	10~500GB
		8核64GB	dds.shard.sn8.4xlarge.3		18000	10~1000 GB
		16核128GB	dds.shard.sn8.8xlarge.3		36000	10~2000 GB
		32核256GB	dds.shard.sn8.16xlarge.3		72000	10~3000 GB
Configserver	通用型	1核2GB	dds.cs.mid		1000	20GB

历史实例规格表

2017年07月10日前的实例，且之后未进行过实例规格变更，将继续采用以下实例规格。

单节点实例规格表

规格类型	规格信息	规格代码	最大连接数	最大IOPS
通用型	1核2GB	dds.n2.small.1	2000	min{30 * 存储空间，20000}
	2核4GB	dds.sn2.medium.1	4000	
	2核8GB	dds.sn4.large.1	6000	

副本集实例规格表

规格类型	规格信息	规格代码	最大连接数	最大IOPS
通用规格	1核2GB	dds.mongo.mid	200	800
	2核4GB	dds.mongo.standard	400	1600
	4核8GB	dds.mongo.large	1000	3200
	8核16GB	dds.mongo.xlarge	2000	6400

规格类型	规格信息	规格代码	最大连接数	最大IOPS
	8核32GB	dds.mongo.2xlarge	4000	12800
	16核64GB	dds.mongo.4xlarge	8000	12800
独享规格	2核16GB	mongo.x8.medium	2000	4500
	4核32GB	mongo.x8.large	4000	9000
	8核64GB	mongo.x8.xlarge	8000	18000
	16核128GB	mongo.x8.2xlarge	16000	36000
	32核256GB	mongo.x8.4xlarge	32000	72000
独占物理机	60核440GB	dds.mongo.2xmonopolize	36000	40000

分片集群实例规格表

节点类型	规格类型	规格信息	规格代码	最大连接数	最大IOPS
Mongos	通用规格	1核2GB	dds.mongos.mid	200	无
		2核4GB	dds.mongos.standard	400	
		4核8GB	dds.mongos.large	1000	
		8核16GB	dds.mongos.xlarge	2000	
		8核32GB	dds.mongos.2xlarge	4000	
		16核64GB	dds.mongos.4xlarge	8000	
Shard	通用规格	1核2GB	dds.shard.mid	无	800
		2核4GB	dds.shard.standard		1600
		4核8GB	dds.shard.large		3200
		8核16GB	dds.shard.xlarge		6400
		8核32GB	dds.shard.2xlarge		12800
		16核64GB	dds.shard.4xlarge		12800
Configserver	通用规格	1核2GB	dds.cs.mid		800

4.2.2. 单节点实例规格表

本文介绍云数据库MongoDB单节点实例支持规格的详细信息。

> **说明** 实例规格定义的内存包括MongoDB相关管理服务、数据库服务和底层操作系统占用的内存（例如BIOS预留内存、内核和Hypervisor运行内存等），因此，您查看的可用内存会小于实例规格定义的内存。

数据库版本	规格类型	规格信息	规格代码	最大连接数	最大IOPS	存储空间
• MongoDB 4.0 • MongoDB 3.4	通用型云盘版	4核8GB	dds.sn2.large.1	6000	min{30 * 存储空间，20000}	20~2000 GB，步长为 10 GB。
		4核16GB	dds.sn4.xlarge.1	8000		
		8核16GB	dds.sn2.xlarge.1	8000		
		8核32GB	dds.sn4.2xlarge.1	10000		

4.2.3. 副本集实例规格表

本文介绍云数据库MongoDB副本集实例支持规格的详细信息。

> **说明** 实例规格定义的内存包括MongoDB相关管理服务、数据库服务和底层操作系统占用的内存（如BIOS预留内存、内核和Hypervisor运行内存等），因此，您查看的可用内存会小于实例规格定义的内存。

数据库版本	规格类型	规格信息	规格代码	最大连接数	最大IOPS	存储空间
		2核8GB	mdb.shard.4x.large.d	3000	min{1800+50*存储空间,10500}	
		2核16GB	mdb.shard.8x.large.d	5000		
		4核8GB	mdb.shard.2x.xlarge.d	3000	min{1800+50*存储空间,21000}	
		4核16GB	mdb.shard.4x.xlarge.d	5000		
		4核32GB	mdb.shard.8x.xlarge.d	8000		
		8核16GB	mdb.shard.2x.2xlarge.d	5000		

数据库版本	规格类型	规格信息	规格代码	最大连接数	最大IOPS	存储空间
• MongoDB 5.0 • MongoDB 4.4	独享型云盘版	8核32GB	mdb.shard.4x.2xlarge.d	8000	min{1800+50*存储空间, 26250}	20~16000 GB，步长为 10 GB。
		8核64GB	mdb.shard.8x.2xlarge.d	16000		
		16核32GB	mdb.shard.2x.4xlarge.d	8000	min{1800+50*存储空间, 42000}	
		16核64GB	mdb.shard.4x.4xlarge.d	16000	min{1800+50*存储空间, 42000}	
		16核128GB	mdb.shard.8x.4xlarge.d	16000		
		32核64GB	mdb.shard.2x.8xlarge.d	16000	min{1800+50*存储空间, 50000}	
		32核128GB	mdb.shard.4x.8xlarge.d	16000		
		32核256GB	mdb.shard.8x.8xlarge.d	16000		
• MongoDB 4.2 • MongoDB 4.0 • MongoDB 3.4	通用型本地盘版	1核2GB	dds.mongo.mid	500	8000	10~2000 GB，步长为 10 GB。
		2核4GB	dds.mongo.standard	1000	8000	
		4核8GB	dds.mongo.large	3000	8000	
		8核16GB	dds.mongo.xlarge	5000	8000	
		8核32GB	dds.mongo.2xlarge	8000	14000	
		16核64GB	dds.mongo.4xlarge	16000	16000	
		2核16GB	mongo.x8.medium	2500	8000	250~3000 GB，步长为 10 GB。
		4核32GB	mongo.x8.large	5000	9000	500~3000 GB，步长为 10 GB。

数据库版本	独享型本地盘版规格类型	规格信息	规格代码	最大连接数	最大IOPS	存储空间
		8核64GB	mongo.x8.xlarge	10000	18000	1000~3000 GB，步长为 10 GB。
		16核128GB	mongo.x8.2xlarge	20000	36000	2000~3000 GB，步长为 10 GB。
		32核256GB	mongo.x8.4xlarge	40000	72000	
	独占物理机	60核440GB	dds.mongo.2xmonopolize	100000	100000	2000~6000 GB，步长为 10 GB。
		90核660GB	dds.mongo.3xmonopolize	100000	100000	2000~6000 GB，步长为 10 GB。

4.2.4. 分片集群实例规格表

本文介绍云数据库MongoDB分片集群实例支持规格的详细信息。

> **说明**
>
> - 实例规格定义的内存包括MongoDB相关管理服务、数据库服务和底层操作系统占用的内存（如BIOS预留内存、内核和Hypervisor运行内存等），因此，您查看的可用内存会小于实例规格定义的内存。
> - 仅部分地域的可用区支持创建MongoDB 5.0和MongoDB 4.4版本的分片集群实例，地域及可用区列表请参见创建分片集群实例。

数据库版本	节点类型	规格类型	规格信息	规格代码	最大连接数	最大IOPS	存储空间
			2核 8 GB	mdb.shard.4x.large.d	3000		
			2核 16 GB	mdb.shard.8x.large.d	5000		
			4核 8 GB	mdb.shard.2x.xlarge.d	3000		
			4核 16 GB	mdb.shard.4x.xlarge.d	5000		
			4核 32 GB	mdb.shard.8x.xlarge.d	8000		

数据库版本	节点类型	规格类型	规格信息	规格代码	最大连接数	最大IOPS	存储空间
• MongoDB 5.0 • MongoDB 4.4	Mongos	独享型云盘版	8核 16 GB	mdb.shard.2x.2xlarge.d	5000	无	无
			8核 32 GB	mdb.shard.4x.2xlarge.d	8000		
			8核 64 GB	mdb.shard.8x.2xlarge.d	16000		
			16核 32 GB	mdb.shard.2x.4xlarge.d	8000		
			16核 64 GB	mdb.shard.4x.4xlarge.d	16000		
			16核 128 GB	mdb.shard.8x.4xlarge.d			
			32核 64 GB	mdb.shard.2x.8xlarge.d			
			32核 128 GB	mdb.shard.4x.8xlarge.d			
			32核 256 GB	mdb.shard.8x.8xlarge.d			
	Shard	独享型云盘版	2核 8 GB	mdb.shard.4x.large.d	3000	min(1800+50*存储空间,10500)	20 GB~32000 GB，步长为10 GB。
			2核 16 GB	mdb.shard.8x.large.d	5000		
			4核 8 GB	mdb.shard.2x.xlarge.d	3000	min(1800+50*存储空间,21000)	
			4核 16 GB	mdb.shard.4x.xlarge.d	5000		
			4核 32 GB	mdb.shard.8x.xlarge.d	8000		
			8核 16 GB	mdb.shard.2x.2xlarge.d	5000	min(1800+50*存储空间,26250)	
			8核 32 GB	mdb.shard.4x.2xlarge.d	8000		
			8核 64 GB	mdb.shard.8x.2xlarge.d	16000		

数据库版本	节点类型	规格类型	规格信息	规格代码	最大连接数	最大IOPS	存储空间
			16核 32 GB	mdb.shard.2x.4xlarge.d	8000	min(1800+50*存储空间,42000)	
			16核 64 GB	mdb.shard.4x.4xlarge.d	16000		
			16核 128 GB	mdb.shard.8x.4xlarge.d			
			32核 64 GB	mdb.shard.2x.8xlarge.d		min(1800+50*存储空间,50000)	
			32核 128 GB	mdb.shard.4x.8xlarge.d			
			32核 256 GB	mdb.shard.8x.8xlarge.d			
	ConfigServer	独享型云盘版	4核 8 GB	mdb.shard.2x.xlarge.d	3000	min(1800+50*存储空间,21000)	固定为20 GB。
	Mongos	通用型本地盘版	1核 2 GB	dds.mongos.mid	1000	无	无
			2核 4 GB	dds.mongos.standard	2000		
			4核 8 GB	dds.mongos.large	4000		
			8核 16 GB	dds.mongos.xlarge	8000		
			8核 32 GB	dds.mongos.2xlarge	16000		
			16核 64 GB	dds.mongos.4xlarge			
		通用型本地盘版	1核 2 GB	dds.shard.mid	8000	8000	10 GB~2000 GB，步长为10 GB。
			2核 4 GB	dds.shard.standard			
			4核 8 GB	dds.shard.large			
			8核 16 GB	dds.shard.xlarge			
			8核 32 GB	dds.shard.2xlarge	16000	14000	
			16核 64 GB	dds.shard.4xlarge	32000	16000	

数据库版本	节点类型	规格类型	规格信息	规格代码	最大连接数	最大IOPS	存储空间
• MongoDB 4.2 • MongoDB 4.0 • MongoDB 3.4	Shard	独享型本地盘版	2核 16 GB	dds.shard.sn8.xlarge.3	8000	8000	10 GB~250 GB，步长为10 GB。
			4核 32 GB	dds.shard.sn8.2xlarge.3		9000	10 GB~500 GB，步长为10 GB。
			8核 64 GB	dds.shard.sn8.4xlarge.3		18000	10 GB~1000 GB，步长为10 GB。
			16核 128 GB	dds.shard.sn8.8xlarge.3	16000	36000	10 GB~2000 GB，步长为10 GB。
			32核 256 GB	dds.shard.sn8.16xlarge.3	32000	72000	10 GB~3000 GB，步长为10 GB。
	ConfigServer	通用型本地盘版	1核 2 GB	dds.cs.mid	8000	8000	固定为20 GB。

4.2.5. 历史规格表

本文介绍云数据库MongoDB实例的历史规格。新申请实例不再提供历史规格，建议您使用最新规格。

单节点实例

规格类型	规格信息	规格代码	最大连接数	最大IOPS
通用型	1核2GB	dds.n2.small.1	2000	min{30 * 存储空间，20000}
	2核4GB	dds.sn2.medium.1	4000	
	2核8GB	dds.sn4.large.1	6000	

副本集实例

规格类型	规格信息	规格代码	最大连接数	最大IOPS
通用型	1核2GB	dds.mongo.mid	200	800
	2核4GB	dds.mongo.standard	400	1600
	4核8GB	dds.mongo.large	1000	3200
	8核16GB	dds.mongo.xlarge	2000	6400
	8核32GB	dds.mongo.2xlarge	4000	12800
	16核64GB	dds.mongo.4xlarge	8000	12800
独享型	2核16GB	mongo.x8.medium	2000	4500
	4核32GB	mongo.x8.large	4000	9000
	8核64GB	mongo.x8.xlarge	8000	18000
	16核128GB	mongo.x8.2xlarge	16000	36000
	32核256GB	mongo.x8.4xlarge	32000	72000
独占物理机	60核440GB	dds.mongo.2xmonopolize	36000	40000

分片集群实例

节点类型	规格类型	规格信息	规格代码	最大连接数	最大IOPS
Mongos	通用型	1核2GB	dds.mongos.mid	200	无
		2核4GB	dds.mongos.standard	400	
		4核8GB	dds.mongos.large	1000	
		8核16GB	dds.mongos.xlarge	2000	
		8核32GB	dds.mongos.2xlarge	4000	
		16核64GB	dds.mongos.4xlarge	8000	
Shard	通用型	1核2GB	dds.shard.mid	无	800
		2核4GB	dds.shard.standard		1600
		4核8GB	dds.shard.large		3200
		8核16GB	dds.shard.xlarge		6400

节点类型	规格类型	规格信息	规格代码	最大连接数	最大IOPS
		8核32GB	dds.shard.2xlarge		12800
		16核64GB	dds.shard.4xlarge		12800
Configserver	通用型	1核2GB	dds.cs.mid		800

5.产品优势

云数据库MongoDB版（ApsaraDB for MongoDB）是阿里云基于飞天分布式系统和高可靠存储引擎研发的完全兼容MongoDB协议，并支持多节点高可用架构、弹性扩容、容灾、备份恢复、性能优化等功能的文档数据库服务。

灵活的部署架构

云数据库MongoDB版具有灵活的部署架构，能够满足不同的业务场景。

云数据库MongoDB版的部署架构详情如下：

- 单节点架构

  适用于开发、测试、学习培训及其他非企业核心数据存储的场景。您可以根据各类场景的差异适配对应的规格配置，为企业降低更多的成本支出。详情请参见单节点架构。

- 副本集架构

  适用于读多写少或有临时活动的突发业务场景。副本集实例提供一个可供读写访问的Primary节点（主节点），一个、三个或五个提供高可用的Secondary节点（从节点），一个隐藏的Hidden节点（隐藏节点），0~5个可选的ReadOnly节点（只读节点）。您可以根据业务需要，按需增删Secondary节点和ReadOnly节点，更好地实现读取性能扩展节点。详情请参见副本集架构。

- 分片集群架构

  适用于高并发读写的场景。分片集群实例是基于多个副本集（每个副本集使用三节点主从模式+0~5个只读节点）组成，提供Mongos、Shard、ConfigServer三个组件，您可以自由地选择Mongos和Shard节点的个数和配置，无限扩展性能及存储空间，组建不同能力的分片集群实例。详情请参见分片集群架构。

弹性扩容

云数据库MongoDB版提供了变更实例配置功能，方便您快速应对业务变化。您可以根据业务需要变更实例的配置（实例规格、存储空间和节点数量），您也可以设置变更配置的生效时间，将生效时间设置在业务低峰期，避免在变更配置过程中对业务造成影响。详情请参见变更配置方案概览。

兼容DynamoDB协议

云数据库MongoDB版不仅完全兼容MongoDB协议，而且云数据库MongoDB 4.0分片集群实例高度兼容了DynamoDB协议，您可以直接在控制台上创建兼容DynamoDB协议的分片集群实例，详情请参见创建DynamoDB协议兼容版实例。

支持阿里云自主研发工具

云数据库MongoDB版进行数据迁移和同步时，不但可以通过控制台和MongoDB官方提供的工具实现，还可以通过阿里云自主研发的工具实现，具体如下：

工具名称	说明
NimoShake	数据同步工具。您可以使用该工具将Amazon DynamoDB数据库迁移至阿里云，详情请参见使用NimoShake将Amazon DynamoDB迁移至阿里云。
MongoShake	阿里云以Golang语言编写的通用平台型服务工具，可以实现数据同步。您可以使用该工具实现MongoDB数据库间的数据同步，详情请参见使用MongoShake实现MongoDB副本集间的单向同步 。

工具名称	说明
NimoFullCheck	用于检验源端DynamoDB和目的端MongoDB数据一致性的工具。您可以使用该工具检验DynamoDB和阿里云数据库之间迁移数据的一致性，详情请参见使用NimoFullCheck校验迁移数据的一致性。

其他优势

云数据库MongoDB版在服务可用性、数据可靠性、安全性、运维成本等方面也具有很大的优势，详情请参见云数据库MongoDB与自建数据库对比优势。

6.云数据库MongoDB与自建数据库对比优势

与自建MongoDB数据库相比，云数据库MongoDB版（ApsaraDB for MongoDB）在服务可用性、数据可靠性、安全性、运维成本等方面具有极大优势。使用云数据库MongoDB版可帮助您快速上线业务、降低运维成本。

对比项	云数据库MongoDB	自建数据库
服务可用性	高可用性。	需要自行保障，自行搭建主从复制、RAID等。
	支持同城单节点或三可用区的高可用性容灾。	同城容灾关系需要自行部署维护，双可用区条件较难实现，无数据库级可用性保障。
	支持构建异地容灾实例。	需要借助第三方工具自行构建容灾实例。
数据可靠性	• 高可靠性。 • 同城单节点、三可用区副本集形态的RPO（Recovery Point Object）为0。	需要自行保障，自行搭建主从复制、RAID等。
系统安全性	事前防护：DDoS攻击防护、自动修复各种数据库安全漏洞、白名单控制访问、VPC网络隔离。	事前防护：需要额外采购安全硬件或软件，自行修复安全漏洞，成本较高。
	事中保护：SSL加密、透明数据加密。	事中保护：需要自行搭建SSL加密及TDE加密系统。
	事后审计：查询审计日志。	事后审计：需要额外采购审计系统。
备份恢复	• 内核完善，手动备份时同时支持物理备份和逻辑备份，备份效率提升3倍。 • 支持单库恢复。	• 开源版本仅支持逻辑备份，效率低下。 • 无单库恢复能力，恢复效率慢。 • 需要自行确保分布式架构恢复数据的准确性。
系统托管	无托管费用。	需要额外支付服务器托管费用，架构越复杂，所需托管的服务器越多，托管费用越高。
运维成本	无需投入额外运维成本。	需要专人维护，运维成本高。
	支持CloudDBA智能诊断，具体如下： • 性能趋势 • 实时性能 • 实例会话 • 慢日志 • 空间分析	没有相关性能诊断功能，排查慢查询较为繁琐。
部署和扩容	即时开通，弹性扩容。	需要自行完成采购硬件、机房托管、部署机器等工作，周期较长，且添加节点需要自行维护节点关系。

对比项	云数据库MongoDB	自建数据库
内核优化	• 根据通用场景优化Oplog同步性能、短连接性能。 • 通过逻辑快照等技术确保数据读一致性，在长时间的数据导入场景中，成本优化50%。	使用普通开源版本，无任何针对性优化，在部分场景下使用受限。

7.应用场景

云数据库MongoDB版支持单节点、副本集和分片集群三种部署架构，具备安全审计、时间点备份等多项企业能力。在互联网、物联网、游戏、金融等领域被广泛采用。

读写分离应用

云数据库MongoDB采用三节点副本集的高可用架构，三个数据节点位于不同的物理服务器上，自动同步数据。Primary节点和Secondary节点提供服务，两个节点分别拥有独立域名，配合MongoDB Driver实现读取压力分配。关于架构说明的详情，请参见云数据库MongoDB产品架构。

灵活多变的业务场景

云数据库MongoDB采用No-Schema的方式，免去您变更表结构的痛苦，非常适用于初创型的业务需求。您可以将模式固定的结构化数据存储在RDS（Relational Database Service）中，模式灵活的业务存储在MongoDB中，高热数据存储在云数据库Redis或云数据库Memcache中，实现对业务数据高效存取，降低存储数据的投入成本。

移动应用

云数据库MongoDB支持二维空间索引，可以很好地支撑基于位置查询的移动类App的业务需求。同时MongoDB动态模式存储方式也非常适合存储多重系统的异构数据，满足移动App应用的需求。

物联网应用

云数据库MongoDB具有高性能和异步数据写入功能，特定场景下可达到内存数据库的处理能力。同时，云数据库MongoDB中的分片集群实例，可按需配置Mongos和Shard组件的配置和个数，性能及存储空间可实现无限扩展，非常适合物联网高并发写入的场景。详情请参见变更配置方案概览。

MongoDB提供二级索引功能满足动态查询的需求，利用MongoDB的map-reduce聚合框架进行多维度的数据分析。

其他各领域应用

- 游戏应用：使用云数据库MongoDB作为游戏服务器的数据库存储用户信息。用户的游戏装备、积分等直接以内嵌文档的形式存储，方便进行查询与更新。
- 物流应用：使用云数据库MongoDB存储订单信息，订单状态在运送过程中会不断更新，以云数据库MongoDB内嵌数组的形式来存储，一次查询就能将订单所有的变更读取出来，方便快捷且一目了然。
- 社交应用：使用云数据库MongoDB存储用户信息以及用户发表的朋友圈信息，通过地理位置索引实现附近的人、地点等功能。并且，云数据库MongoDB非常适合用来存储聊天记录，因为它提供了非常丰富的查询，并在写入和读取方面都相对较快。
- 视频直播：使用云数据库MongoDB存储用户信息、礼物信息等。
- 大数据应用：使用云数据库MongoDB作为大数据的云存储系统，随时进行数据提取分析，掌握行业动态。

8.版本说明

8.1. MongoDB 5.0新特性概览

本文介绍MongoDB 5.0的主要新特性。

背景信息

MongoDB 5.0标志着一个新的发布周期的到来，以更快地交付新特性给用户。例如：版本化API与在线重新分片相结合，使用户不必担心未来的数据库升级以及业务变化问题；本地原生时间序列数据平台使MongoDB能够支持更广泛的工作负载和业务场景；新的MongoDB Shell能够提升用户体验等均为MongoDB 5.0的功能。

原生时间序列平台

MongoDB 5.0通过原生支持整个时间序列数据的生命周期（从采集、存储、查询、实时分析和可视化，到在线归档或随着数据老化自动失效），使构建和运行时间序列应用程序的速度更快、成本更低。随着MongoDB 5.0的发布，MongoDB扩展了通用的应用数据平台，使开发能够更容易地处理时间序列数据，进一步扩展其在物联网、金融分析、物流等方面的应用场景。

MongoDB的时间序列集合以高度优化和压缩的格式自动存储时间序列数据，减少了存储大小和I/O，以实现更好的性能和更大的规模。同时也缩短了开发周期，使您能够快速建立一个针对时间序列应用的性能和分析需求而调优的模型。

创建时间序列数据集合的命令示例：

```
db.createCollection("collection_name",{ timeseries: { timeField: "timestamp" } } )
```

MongoDB可以无缝地调整采集频率，并根据动态生成的时间分区自动处理无序的测量值。最新发布的MongoDB Connector for Apache Kafka实现了在本地支持时间序列，您可以直接从Kafka主题消息中自动创建时间序列集合，使您在收集数据的同时根据需要对数据进行处理和聚合，然后写入到MongoDB的时间序列集合。

时间序列集合自动创建一个按时间排序的数据聚集索引，降低查询数据的延迟。MongoDB查询API还扩展了窗口函数，您可以运行分析性查询（例如移动平均数和累积总和）。在关系型数据库系统中，这些通常被称为SQL分析函数，并支持以行为单位定义的窗口（即三行移动平均线）。MongoDB更进一步，还增加了指数移动平均线、导数和积分等强大的时间序列函数，支持您以时间为单位定义窗口（例如15分钟的移动平均线）。窗口函数可用于查询MongoDB的时间序列和常规集合，为多种应用类型提供了新的分析方式。另外，MongoDB 5.0也提供了新的时间运算符，包括 `$dateAdd` 、 `$dateSubstract` 、 `$dateDiff` 和 `$dateTrunc` ，使您可以通过自定义的时间窗口对数据进行汇总和查询。

您可以将MongoDB的时间序列数据与企业的其他数据相结合。时间序列集合可以与同一个数据库中的常规MongoDB集合放在一起，您不必选择一个专门的时间序列数据库（它不能为任何其他类型的应用提供服务），也不需要复杂的集成来混合时间序列和其他数据。MongoDB通过提供一个统一的平台，让您建立高性能和高效的时间序列应用的同时，也为其他用例或工作负载提供支持，从而消除了整合和运行多个不同数据库的成本和复杂性。

在线数据重新分片

数据库版本	特点	实现方法
MongoDB 5.0以前	重新分片过程复杂且需要手动分片。	• 方法一：先dump整个集合，然后用新的分片键把数据库重新加载到一个新的集合中。 由于这是一个需要离线处理的过程，因此您的应用程序在重新加载完成之前需要中断停服较长时间。例如：在一个三分片的集群上dump和重新加载一个10 TB以上的集合可能需要几天时间。 • 方法二：新建一个分片集群并重新设定集合的分片键，然后通过定制迁移方式，将旧分片集群中需要重新分片的集合，按新的分片键写入到新的分片集群中。 ○ 该过程中需要您自行处理查询路由和迁移逻辑、不断检查迁移进度，以确保所有数据迁移成功。 ○ 定制迁移是高度复杂的、劳动密集型的、有风险的任务，而且耗时很长。例如：某个MongoDB用户花了三个月才完成100亿个document的迁移。
MongoDB 5.0开始	• 运行**reshardCollection**命令即可启动重新分片。 • 重新分片的过程高效。 并不是简单地重新平衡数据，而是在后台将所有当前集合的数据复制并重新写入新集合，同时与应用程序新的写入保持同步。 • 重新分片是完全自动化的。 将重新分片花费的时间从几周或几个月压缩到几分钟或几小时，避免了冗长繁杂的手动数据迁移。 • 通过使用在线重新分片，可以方便地在开发或测试环境中评估不同分片键的效果，也可以在您需要时修改分片键。	您可以在业务运行（数据不断增长）的情况下，按需改变集合的分片键（Shard key），而不需要数据库停机或在数据集合中进行复杂的迁移。您只需要在MongoDB Shell中运行**reshardCollection**命令，选择您需要重新分片的数据库和集合，指定新的分片键即可。 `reshardCollection: "<database>.<collection>", key: <shardkey>` **说明** • <database>：需要重新分片的数据库名称。 • <collection>：需要重新分片的集合名称。 • <shardkey>：分片键的名称。 • 当您调用**reshardCollection**命令时，MongoDB会克隆现有集合，然后将现有集合中所有oplog应用到新集合中，当所有oplog被使用后，MongoDB会自动切换到新集合，并在后台删除旧集合。

版本化API

- 应用程序兼容性

  从MongoDB 5.0开始，版本化API定义了应用程序最常用的一组命令和参数（无论是数据库在年度重大发布还是季度快速发布期间，这些命令均不会改变）。通过将应用程序生命周期和数据库生命周期解耦，您可以将驱动程序固定在MongoDB API的特定版本上，即使数据库发生升级和改进，您的应用程序将可以继续运行数年而不需要修改代码。

- 灵活地添加新功能和改进内容

版本化API支持MongoDB灵活地在每个版本中为数据库添加新的功能和改进内容（以新版本兼容早期版本的方式）。当您需要改变API时，可以增加新版本的API，并与现有版本化的API在同一台服务器上同时运行。随着MongoDB版本发布的加速，版本化API能够使您更快、更轻松地使用到MongoDB最新版本的功能特性。

Write Concern默认Majority级别

从MongoDB 5.0开始，Write Concern默认级别为majority，仅当写入操作被应用到Primary节点（主节点）且被持久化到大多数副本节点的日志中的时候，才会提交并返回成功，“开箱即用”地提供了更强的数据可靠性保障。

> **说明** Write Concern是完全可调的，您可以自定义配置Write Concern，以平衡应用程序对数据库性能和数据持久性的要求。

连接管理优化

默认情况下，一个客户端连接对应后端MongoDB服务器上的一个线程（net.serviceExecutor配置为 `synchronous` ）。创建、切换和销毁线程都是消耗较大的操作，当连接数过多时，线程会占用MongoDB服务器较多的资源。

连接数较多或创建连接失控的情况称为“连接风暴”，产生该情况的原因可能是多方面的，且经常是在服务已经受到影响的情况下发生。

针对这些情况，MongoDB 5.0采取了以下措施：

- 限制在任何时候驱动程序尝试创建的连接数量，以简单有效的方式防止数据库服务器过载。
- 减少驱动程序监控连接池时的检查频率，给无响应或过载的服务器节点一个缓冲和恢复的机会。
- 驱动程序将工作负载导向具有最健康连接池的更快的服务器，而不是从可用的服务器中随机选择。

以上措施，加上之前版本在mongos查询路由层的改进，进一步提升了MongoDB承受高并发负载的能力。

长时间运行的快照查询

长时间运行的快照查询（Long-Running Snapshot Queries）增加了应用程序的通用性和弹性。您可以通过该功能运行默认时间为5分钟的查询（或将其调整为自定义持续时间），同时保持与实时事务性数据库一致的快照隔离，也可以在Secondary节点（从节点）上进行快照查询，从而在单个集群中运行不同的工作负载，并将其扩展到不同的分片上。

MongoDB通过底层存储引擎中一个名为Durable history的项目实现了长期运行的快照查询，该项目早在MongoDB 4.4中就已实现。Durable history将存储自查询开始以来所有变化的字段值的快照。通过使用Durable history，查询可以保持快照隔离，即使在数据发生变化的情况下，Durable history也有助于降低存储引擎的缓存压力，使得业务可以在高写入负载的场景下实现更高的查询吞吐量。

新版MongoDB Shell

为了提供更好的用户体验，MongoDB 5.0从头开始重新设计了MongoDB Shell（mongosh），以提供一个更现代化的命令行体验，以及增强可用性的功能和强大的脚本环境。新版MongoDB Shell已经成为MongoDB平台的默认Shell。新版MongoDB Shell引入了语法高亮、智能自动完成、上下文帮助和有用的错误信息，为您创造一个直观、互动的体验。

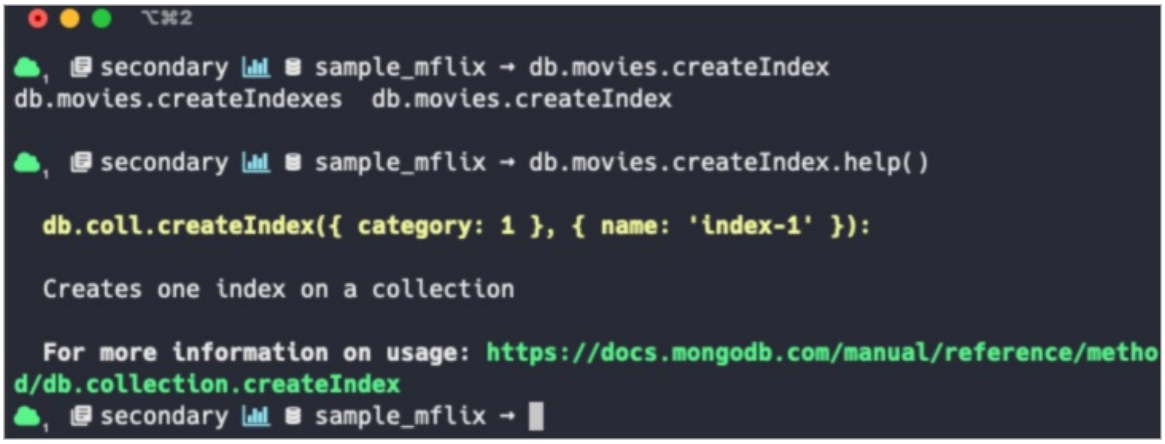
- 增强的用户体验
  - 更容易编写查询和聚合，更容易阅读结果。

    新版MongoDB Shell支持语法高亮功能，方便您区分字段、值和数据类型，以避免语法错误。如果仍然发生错误，新版MongoDB Shell也可以指出问题点并告诉您解决方法。

- 更快输入查询和命令。

  新版MongoDB Shell支持智能自动完成功能，即新版MongoDB Shell可以根据您连接的MongoDB的版本，为方法、命令、MQL表达式等给出自动完成选项的提示。

  示例：当您不记得某个命令的语法时，您可以直接从MongoDB Shell中快速查找该命令的语法。



- 高级脚本环境

  新版MongoDB Shell的脚本环境建立在Node.js REPL（交互式解释器）之上，您在脚本中可以使用所有的Node.js API和NPM的任何模块。您也可以从文件系统中加载和运行脚本（和旧版MongoDB Shell一样，您可以继续使用Load和Eval执行脚本）。

- 扩展性和插件

  新版MongoDB Shell具有易扩展性，使您能够使用MongoDB的所有功能以提高生产力。

  在新版MongoDB Shell中，允许安装Snippets插件。Snippets可以自动加载至MongoDB Shell中，且Snippets可以使用所有的Node.js API和NPM包。MongoDB也维护了一个Snippets仓库，提供了一些有趣的功能（例如分析指定集合模式的插件），您也可以自由地配置MongoDB Shell使用您选择的插件。

  > **说明** 插件当前仅为MongoDB Shell的一个实验性功能。

PyMongoArrow与数据科学

随着新的PyMongoArrow API的发布，您可以在MongoDB上使用Python运行复杂的分析和机器学习。PyMongoArrow可以快速将简单的MongoDB查询结果转换为流行的数据格式（例如Pandas数据框架和NumPy数组），帮助您简化数据科学工作流程。

Schema验证改进

Schema验证（模式验证）是对MongoDB进行数据应用管理控制的一种方式。MongoDB 5.0中，模式验证变得更加简单和友好，当操作验证失败时都会产生描述性的错误信息，帮助您了解不符合集合验证器的验证规则的文档及原因，以快速识别和纠正影响验证规则的错误代码。

可恢复的索引创建任务

MongoDB 5.0支持将正在进行中的索引创建任务在节点重新启动后自动会恢复至原来的位置，减少计划中维护动作对业务的影响。例如：重新启动或升级数据库节点时，您不需要担心当前正在进行的大集合索引创建任务失效。

版本发布调整

由于MongoDB支持很多版本和平台，每个发布版本都需在20多个MongoDB支持的平台上进行验证，验证工作量大，降低了MongoDB新功能的交付速度，所以从MongoDB 5.0开始，MongoDB发布的版本将分为Major Release（大版本）和Rapid Releases（快速发布版本），其中Rapid Releases作为开发版本提供下载和测试体验，但不建议用在生产环境。

更多特性

关于MongoDB 5.0更多特性，请参见MongoDB 5.0。

8.2. MongoDB 4.4功能概览

阿里云作为MongoDB官方战略合作伙伴，全网首次引入MongoDB 4.4版本并已于2020年11月发布。而MongoDB官方4.4版本已经在2020年7月30日正式发布。和往年的大版本不同，本次的4.4版本是以往版本的全面加强版，主要针对用户呼声最高的一些痛点重点进行了改进。

Hidden Indexes

众所周知数据库维护太多的索引会导致写性能下降，但是往往业务上的复杂性导致了运维人员不敢轻易去删除一个潜在的低效率索引，担心误删除会带来业务性能的抖动，而重建索引的代价也非常大。

为了解决上述问题，阿里云数据库MongoDB版和MongoDB官方达成战略合作后共同开发了Hidden Indexes功能。该功能支持通过 `collMod` 命令隐藏现有的索引，保证该索引在后续的查询中不会被使用。在观察一段时间后，确定业务没有异常即可以放心删除该索引。

参考代码：

```
db.runCommand( {
   collMod: 'testcoll',
   index: {
      keyPattern: 'key_1',
      hidden: false
   }
} )
```

需要注意的是，索引被隐藏之后仅对MongoDB的执行计划器不可见，这并不会改变索引本身的一些特殊行为，如唯一键约束、TTL淘汰等。

> **说明** 索引在隐藏期间也不会停止更新，所以当需要该索引时，可以通过取消隐藏使其立刻可用。

Refinable Shard Keys

在MongoDB分片集群中，一个好的Shard key至关重要，因为它决定了分片集群在指定的Workload（工作量）下是否有良好的扩展性。但是在实际使用MongoDB的过程中，即使我们事先仔细斟酌了要选择的Shard Key，也会因为Workload的变化而导致出现Jumbo Chunk（超过预设大小的Chunk），或者业务流量都打向单一分片的情况。

在4.0及之前的版本中，集合选定的Shard Key及其对应的Value都是不能更改的，到了4.2版本，虽然可以修改Shard Key的Value，但是数据的跨分片迁移以及基于分布式事务的实现机制导致性能开销很大，而且并不能完全解决Jumbo Chunk或访问热点的问题。例如，现在有一个订单表，Shard Key为 `{customer_id:1}` ，在业务初期每个客户不会有很多的订单，这样的Shard Key完全可以满足需求，但是随着业务的发展，某个大客户累积的订单越来越多，进而对这个客户订单的访问成为某个单一分片的热点，由于订单和 `customer_id` 天然的关联关系，修改 `customer_id` 并不能改善访问不均的情况。

针对上述类似场景，在4.4版本中，您可以通过refineCollectionShardKey命令给现有的Shard Key增加一个或多个Suffix Field来改善现有的文档在Chunk上的分布问题。例如在上面描述的订单业务场景中，通过 `refineCollectionShardKey` 命令把Shard key更改为 `{customer_id:1, order_id:1}` ，即可避免单一分片上的访问热点问题。

并且， `refineCollectionShardKey` 命令的性能开销非常低，仅更改Config Server节点上的元数据，不需要任何形式的数据迁移，数据的打散仍然在后续正常的Chunk自动分裂和迁移的流程中逐步进行。此外，Shard Key需要有对应的Index来支撑，因此 `refineCollectionShardKey` 命令要求提前创建新Shard Key所对应的Index。

由于并不是所有的文档都存在新增的Suffix Field，因此在4.4版本中隐式支持了Missing Shard Key功能，即新插入的文档可以不包含指定的Shard Key Field。但是由于很容易产生Jumbo Chunk，因此并不建议使用。

Compound Hashed Shard Keys

在4.4之前的版本中，您只能指定单字段的哈希片键，原因是当时版本的MongoDB不支持复合哈希索引，这样就很容易导致集合数据在分片上分布不均匀。

在最新的4.4版本中加入了复合哈希索引，即您可以在复合索引中指定单个哈希字段，位置不限，可以作为前缀，也可以作为后缀，进而也就提供了对复合哈希片键的支持。

参考代码：

```
sh.shardCollection(
  "examples.compoundHashedCollection",
  { "region_id" : 1, "city_id": 1, field1" : "hashed" }
)
sh.shardCollection(
  "examples.compoundHashedCollection",
  { "_id" : "hashed", "fieldA" : 1}
)
```

复合哈希索引有很多优点，例如如下两个场景：

- 应法律法规的要求，需要使用MongoDB的zone sharding功能，把数据尽量均匀打散在某个地域的分片上。
- 集合指定的片键的值是递增的，例如上文例子中的 `{customer_id:1, order_id:1}` 这个片键，如果 `customer_id` 是递增的，并且业务也总是访问最新顾客的数据，导致大部分的流量总是访问单一分片。

在没有复合哈希片键支持的情况下，只能提前对需要的字段进行哈希值的计算，并将结果存储到文档中的某个特殊字段中，然后再通过范围分片的方式指定其作为片键来解决上述问题。

而在4.4版本中只需直接把目标字段指定为哈希即可轻松解决上述问题。例如，针对上述第二个场景，仅需将片键设置为 `{customer_id:'hashed', order_id:1}` 即可在极大程度上简化业务逻辑的复杂性。

Hedged Reads

页面的响应速度和经济损失直接挂钩。Google有一个研究报告表明，如果网页的加载时间超过3秒，用户的跳出率会增加50%。针对这个问题，MongoDB在4.4版本中提供了Hedged Reads功能，即在分片集群场景下，mongos节点会把一个读请求同时发送给某个分片的两个副本集成员，然后选择最快的返回结果回复客户端，来减少业务上的P95（指过去十秒内95%的请求延迟均在规定范围内）和P99延迟（指过去十秒内99%的请求延迟均在规定范围内）。

Hedged Reads功能作为Read Preference参数的一部分来提供，因此可以在Operation粒度上进行配置，当Read Preference指定为 `nearest` 时，系统默认启用Hedged Reads功能，当指定为primary时，不支持Hedged Reads功能，当指定为其他时，需要显式地指定 `hedgeOptions` 才可以启用Hedged Reads。如下所示：

```
db.collection.find({ }).readPref(
   "secondary",                       // mode
   [ { "datacenter": "B" },  { } ],   // tag set
   { enabled: true }                  // hedge options
)
```

此外，Hedged Reads也需要mongos开启支持，配置readHedgingMode参数为 `on` ，使mongos开启该功能支持。

参考代码：

```
db.adminCommand( { setParameter: 1, readHedgingMode: "on" } )
```

降低复制延迟

本次4.4的更新带来了主备复制延迟的降低。对于MongoDB来说，主备复制的延迟会对读写有非常大的影响。在某些特定的场景下，备库需要及时地复制并应用主库的增量更新，才可以继续进行读写操作。因此，更低的复制延迟会带来更好的一致性体验。

Streaming Replication

在4.4之前的版本中，备库需要通过不断地轮询upstream来获取增量更新操作。每次轮询时，备库主动给主库发送一个 `getMore` 命令读取Oplog集合，如果有数据，会返回一个最大16MB的Batch，如果没有数据，备库也会通过awaitData选项来控制备库无谓的 `getMore` 开销，同时能够在有新的增量更新时，第一时间获取到对应的Oplog。拉取操作是通过单个OplogFetcher线程来完成，每个Batch的获取都需要经历一个完整的RTT（Round-Trip Time，往返时间），在副本集网络状况不好的情况下，复制的性能就严重受限于网络延迟。

而在4.4版本中，增量的Oplog是不断地主动流向备库的，而不是被动地依靠备库轮询。相比于备库轮询的方式，至少在Oplog的获取上节省了一半的RTT。在以下两个场景中，Streaming Replication功能会大大提升性能：

- 当用户的写操作指定了writeConcern参数为 `"majority"` 时，写操作需要等待足够多次数的“备库返回复制成功”。而在新的复制机制下，高延迟的网络环境也可以平均提升50%的 `majority` 写性能。
- 当用户使用了因果一致性（Causal Consistency）的场景下，为了保证可以在备库读到自己的写操作（Read Your Write），同样强依赖备库对主库Oplog的及时复制。

Simultaneous Indexing

4.4 之前的版本中，索引的创建需要在主库中完成之后，才会到备库上执行。备库上的创建动作在不同的版本中，因为创建机制和创建方式的不同，对备库Oplog的影响也大有不同。

但即使在4.2版本中统一了前后台索引创建机制，使用了相当细粒度的加锁机制（只在索引创建的开始和结束阶段对集合加独占锁），也会因为索引创建本身的CPU、IO性能开销导致复制延迟，或是因为一些特殊操作，例如使用 `collMod` 命令修改集合元信息，而导致Oplog的应用阻塞，甚至会因为主库历史Oplog被覆盖而进入Recovering状态。

在4.4版本中，主库和备库上的索引创建操作是同时进行的，这样可以大幅减少上述情况所带来的主备延迟，即使在索引创建过程中，也可以保证备库访问到最新的数据。

此外，新的索引创建机制规定，只有在大多数具备投票权限节点返回成功后，索引才会真正生效。所以，也可以减轻在读写分离场景下因为索引不同而导致的性能差异。

Mirrored Reads

在阿里云数据库MongoDB版以往提供服务的过程中，有一个现象，即大多数用户虽然购买的是三节点副本集实例，但是实际在使用过程中读写都是在Primary节点进行，其中一个可见的Secondary节点并未承载任何读流量，导致在偶尔的宕机切换之后，用户能明显感受到业务的访问延迟，经过一段时间后才会恢复到之前的水平，原因就在于新选举出的主节点之前从未提供过读服务，并不了解业务的访问特征，没有针对性地对数据做缓存，所以在突然提供服务后，读操作会出现大量的缓存未命中（Cache Miss），需要从磁盘重新加载数据，造成访问延迟上升。在大内存实例的情况下，这个问题尤为明显。

在4.4版本中，MongoDB针对上述问题实现了Mirrored Reads功能，即主节点会按一定的比例把读流量复制到备库上执行，来帮助备库预热缓存。这是一个非阻塞执行（Fire and Forgot）的行为，不会对主库的性能产生任何实质性的影响，但是备库负载会有一定程度的上升。

流量复制的比例是可动态配置的，通过mirrorReads参数设置，默认复制 `1%` 的流量。

参考代码：

```
db.adminCommand( { setParameter: 1, mirrorReads: { samplingRate: 0.10 } } )
```

此外，还可以通过 `db.serverStatus( { mirroredReads: 1 } )` 来查看Mirrored Reads相关的统计信息，如下所示：

```
SECONDARY> db.serverStatus( { mirroredReads: 1 } ).mirroredReads
{ "seen" : NumberLong(2), "sent" : NumberLong(0) }
```

Resumable Initial Sync

在4.4之前的版本中，如果备库在做全量同步时出现网络抖动而导致连接闪断，那么备库需要从头开始全量同步，导致之前的工作全部白费，这个情况在数据量比较大时会对业务造成巨大的影响。

在4.4版本中，MongoDB提供了从中断位置继续执行同步的能力。如果在闪断后一直无法连接成功，系统会重新选择一个同步源进行新的全量同步。该过程的默认超时时间为24小时，您可以通过 `replication.initialSyncTransientErrorRetryPeriodSeconds` 在进程启动时更改。

需要注意的是，对于全量同步过程中遇到的非网络异常导致的中断，仍然需要重新发起全量同步。

Time-Based Oplog Retention

MongoDB中的Oplog集合记录了所有数据的变更操作，除了用于复制，还可用于增量备份、数据迁移、数据订阅等场景，是MongoDB数据生态的重要基础设施。

Oplog是通过Capped Collection来实现的，虽然从3.6版本开始，MongoDB支持通过 `replSetResizeOplog` 命令动态修改Oplog集合的大小，但是往往不能准确反映下游对Oplog增量数据的需求，您可以考虑如下场景：

- 计划在凌晨2~4点对某个Secondary节点进行停机维护，需要避免上游Oplog被清理而触发全量同步。
- 下游的数据订阅组件可能会因为一些异常情况而停止服务，但是最慢会在3个小时之内恢复服务并继续进行增量拉取，也需要避免上游的增量缺失。

由此可见，在大部分应用场景下，需要保留最近一个时间段内的Oplog，而这个时间段内产生多少Oplog往往是很难确定的。

在4.4版本中，MongoDB支持通过storage.oplogMinRetentionHours参数定义需要保留的Oplog时长，也可以通过 `replSetResizeOplog` 命令在线修改这个值，参考代码如下：

```
// First, show current configured value
db.getSiblingDB("admin").serverStatus().oplogTruncation.oplogMinRetentionHours
// Modify
db.adminCommand({
  "replSetResizeOplog" : 1,
  "minRetentionHours" : 2
})
```

Union

在多表联合查询能力上，4.4之前的版本只提供了一个$lookup stage用于实现类似于SQL中的 `left outer join` 功能，而4.4版本中新增了$unionWith stage用于实现类似于SQL的 `union all` 功能，用于将两个集合中的数据聚合到一个结果集中，然后做指定的查询和过滤。区别于 `$lookup stage` 的是， `$unionWith stage` 支持分片集合。在Aggregate Pipeline中使用多个 `$unionWith stage` ，可以对多个集合数据做聚合，使用方式如下：

```
{ $unionWith: { coll: "<collection>", pipeline: [ <stage1>, ... ] } }
```

您也可以在pipeline参数中指定不同的stage，用于在对集合数据做聚合前进行一定的过滤，使用起来非常灵活。例如，某个业务上对订单数据按表拆分存储到不同的集合中，第二季度有如下数据：

```
db.orders_april.insertMany([
  { _id:1, item: "A", quantity: 100 },
  { _id:2, item: "B", quantity: 30 },
]);
db.orders_may.insertMany([
  { _id:1, item: "C", quantity: 20 },
  { _id:2, item: "A", quantity: 50 },
]);
db.orders_june.insertMany([
  { _id:1, item: "C", quantity: 100 },
  { _id:2, item: "D", quantity: 10 },
]);
```

假设需要列出第二季度中不同产品的销量，在4.4版本之前，可能需要业务自己把数据都读出来，然后在应用层面做聚合才能解决这个问题，或者依赖某种数据仓库产品来做分析，而在4.4版本中只需要如下一条Aggregate语句即可解决问题：

```
db.orders_april.aggregate( [
   { $unionWith: "orders_may" },
   { $unionWith: "orders_june" },
   { $group: { _id: "$item", total: { $sum: "$quantity" } } },
   { $sort: { total: -1 }}
] )
```

Custom Aggregation Expressions

4.4之前的版本中，您可以通过 `find` 命令中的$where operator或者MapReduce功能来实现在Server端执行自定义的JavaScript脚本，进而提供更为复杂的查询能力，但是这两个功能并没有做到和Aggregation Pipeline在使用上的统一。

在4.4版本中，MongoDB提供了$accumulator和$function这两个新的Aggregation Pipeline Operator用来取代 `$where operator` 和MapReduce。借助于Server Side JavaScript来实现自定义的Aggregation Expression，这样做到复杂查询的功能接口都集中到Aggregation Pipeline中，完善接口统一性和用户体验的同时，也可以把Aggregation Pipeline本身的执行模型利用上，实现一举多得的效果。

`$accumulator` 和MapReduce功能有些相似，会先通过 `init` 函数定义一个初始的状态，然后根据指定的 `accumulate` 函数更新每一个输入文档的状态，并且根据需要决定是否执行 `merge` 函数。

例如，假设在分片集合上使用了 `$accumulator operator` ，则需要将在不同分片上执行完成的结果做 `merge` ，并且如果指定了 `finalize` 函数，那么在所有输入文档处理完成后，还会根据该函数将状态转换为最终的输出。

`$function` 和 `$where operator` 在功能上基本一致，但其强大之处在于可以和其他Aggregation Pipeline Operator配合使用，此外也可以在 `find` 命令中借助 `$expr operator` 来使用 `$function operator` ，等价于之前的 `$where operator` ，MongoDB官方在文档中也建议优先使用 `$function operator` 。

其他易用性增强

除了上述 `$accumulator` 和 `$function operator` ，4.4版本中还新增了其他多个Aggregation Pipeline Operator，例如字符串处理、获取数组收尾元素、还有用来获取文档或二进制串大小的操作符，具体请参见下表：

操作符	说明
$accumulator	返回用户定义的accumulator operator结果。
$binarySize	返回指定字符串或二进制数据的大小（以字节为单位）。
$bsonSize	返回编码为BSON时指定文档（即bsontype对象）的字节大小。
$first	返回数组中的第一个元素。
$function	用来自定义aggregation表达式。
$last	返回数组中的最后一个元素。
$isNumber	如果指定的表达式类型为整数、十进制、双精度或长整型，则返回布尔值 `true` 。如果表达式类型为BSON类型、null或缺失字段，则返回布尔值 `false` 。
$replaceOne	替换第一个通过指定的字符串匹配到的实例。
$replaceAll	替换所有通过指定的字符串匹配到的实例。

Connection Monitoring and Pooling

4.4版本的Driver中增加了对客户端连接池的行为监控和自定义配置，通过标准的API来订阅和连接池相关的事件，包括连接的关闭和开启、连接池的清理。也可以通过API来配置连接池的一些行为，例如拥有的最大或最小连接数、每个连接的最大空闲时间、线程等待可用连接时的超时时间等。具体可以参见MongoDB官方设计文档。

Global Read and Write Concerns

在4.4之前的版本中，如果执行的操作没有显式地指定 `readConcern` 或者 `writeConcern` ，则会有默认行为。例如： `readConcern` 默认为 `local` ， `writeConcern` 默认为 `{w: 1}` 。但这个默认行为不可以变更，如果用户想让所有 `insert` 操作的 `writeConcern` 默认为 `{w: majority}` ，那么只能在所有访问MongoDB的代码中显式指定该值。

而在4.4版本中，可以通过setDefaultRWConcern命令来配置全局默认的 `readConcern` 和 `writeConcern` 。参考代码：

```
db.adminCommand({
  "setDefaultRWConcern" : 1,
  "defaultWriteConcern" : {
    "w" : "majority"
  },
  "defaultReadConcern" : { "level" : "majority" }
})
```

您也可以通过getDefaultRWConcern命令获取当前默认的 `readConcern` 和 `writeConcern` 。

此外，在4.4版本中记录慢日志或诊断日志的时候，会记录当前操作的 `readConcern` 或者 `writeConcern` 设置的来源，两者共通的来源有如下三种：

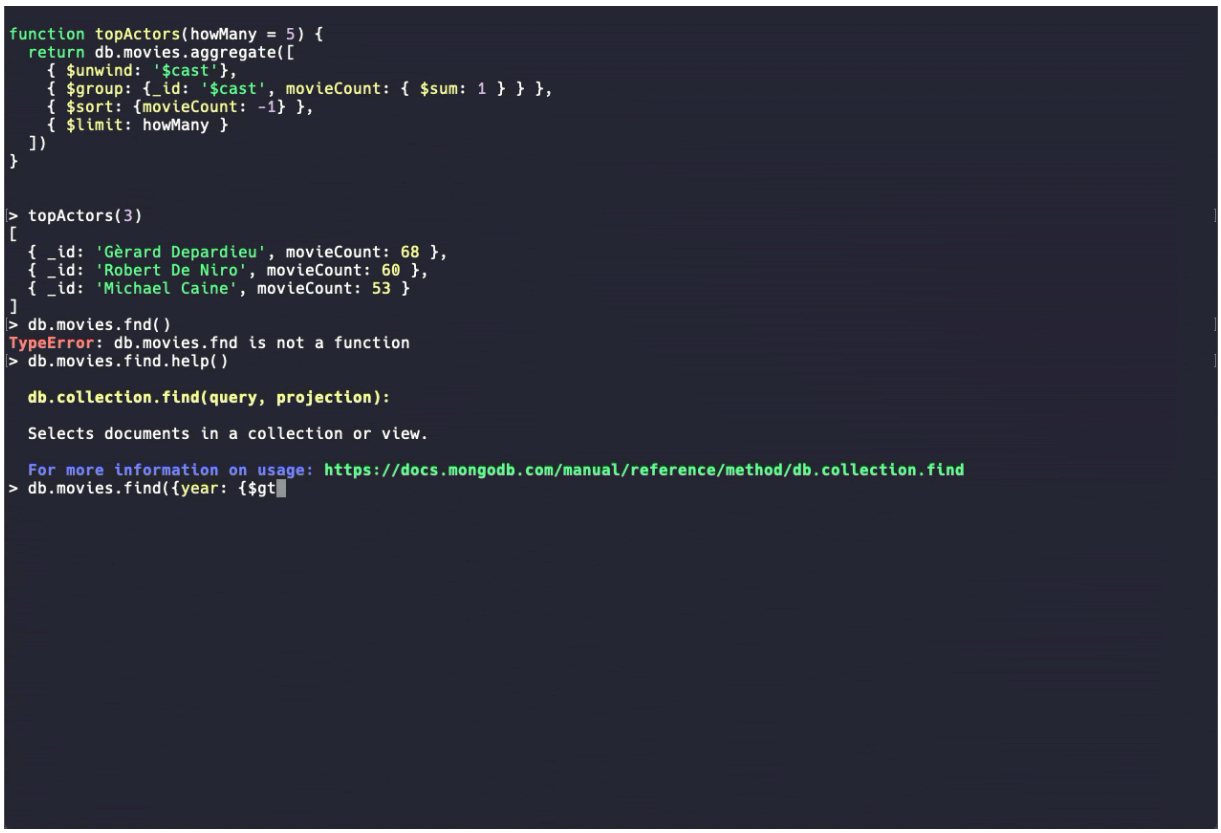
来源	说明
clientSupplied	由应用自己指定。
customDefault	由用户通过 `setDefaultRWConcern` 命令指定。
implicitDefault	完全没做任何配置，Server默认行为。

`writeConcern` 还有如下一种来源：

来源	说明
getLastErrorDefaults	继承自副本集的 `settings.getLastErrorDefaults` 配置。

New MongoDB Shell (beta)

对于MongoDB的运维人员来说，使用最多的工具可能就是Mongo Shell，4.4版本提供了新版本的Mongo SShell，增加了诸如代码高亮、命令自动补全、更具可读性的错误信息等非常人性化的功能。目前提供的是beta版本，很多命令还未提供支持，仅供体验。



结语

本次发布的4.4版本主要是一个维护性的版本，除了上述解读，还有很多其他小的优化，例如$indexStats优化、TCP Fast Open支持优化建连、索引删除优化等等。还有一些相对大的增强，例如新的结构化日志LogV2、新的安全机制支持等。详情请参见官方的Release Notes。

8.3. MongoDB数据库大版本升级说明

您可以通过云数据库MongoDB控制台或调用UpgradeDBInstanceEngineVersion接口升级数据库大版本。升级前，建议您了解不同系统架构的云数据库MongoDB实例支持的数据库大版本、大版本升级的支持情况和升级前后大版本间的变更内容，帮助您判断是否需要升级和需要升级至哪个大版本。

支持的数据库大版本

不同系统架构的云数据库MongoDB实例支持的数据库大版本如下：

系统架构	MongoDB 5.0	MongoDB 4.4	MongoDB 4.2	MongoDB 4.0	MongoDB 3.4	MongoDB 3.2	MongoDB 3.0

系统架构	MongoDB 5.0	MongoDB 4.4	MongoDB 4.2	MongoDB 4.0	MongoDB 3.4	MongoDB 3.2	MongoDB 3.0
单节点架构	✗	✗	✗	✓	✓	已下线，具体请参见【通知】2月4日云数据库MongoDB版下线3.2版本，上线4.2版本。	已下线。
副本集架构	✓	✓	✓	✓	✓		
分片集群架构	✓	✓	✓	✓	✓		

可以升级到的数据库大版本

不同系统架构的云数据库MongoDB实例可以升级到的数据库大版本如下：

系统架构	实例支持的版本	可以升级到的数据库大版本
单节点架构	MongoDB 3.4和MongoDB 4.0	不支持升级数据库大版本
副本集架构	MongoDB 5.0	不支持升级数据库大版本
	MongoDB 4.4	MongoDB 5.0
	MongoDB 4.2	不支持升级数据库大版本
	MongoDB 4.0	MongoDB 4.2
	MongoDB 3.4	MongoDB 4.0和MongoDB 4.2
分片集群架构	MongoDB 5.0	不支持升级数据库大版本
	MongoDB 4.4	MongoDB 5.0
	MongoDB 4.2	不支持升级数据库大版本
	MongoDB 4.0	MongoDB 4.2
	MongoDB 3.4	MongoDB 4.0和MongoDB 4.2

大版本变更说明

以下为相邻大版本间升级（低版本升级至高版本，例如MongoDB 4.0升级至MongoDB 4.2）时的变更说明。如果您需要跨版本升级，则变更说明为低版本到高版本间所有大版本的变更说明。

> **说明**
>
> - 不同系统架构的云数据库MongoDB实例可以升级到的数据库大版本请参见可以升级到的数据库大版本。
> - 您可以在实例运行期间手动升级数据库大版本，但大版本升级后不支持降级。升级方法请参见升级数据库版本。
> - 在MongoDB 3.4和MongoDB 4.0间，官方MongoDB还支持MongoDB 3.6，所以在将MongoDB 3.0、MongoDB 3.2或MongoDB 3.4升级至MongoDB 4.0或MongoDB 4.0以上大版本的过程中，变更内容包括升级至MongoDB 3.6的内容。
> - 变更说明仅属于MongoDB内核上的变动，不包含不同系统架构的云数据库MongoDB实例功能上的差异。

数据库大版本	变更说明
MongoDB 5.0	• Read Concern的默认级别为local，5.0 之前的版本，Secondary节点的默认值是 `available` 。 • write concern的默认值由“1”修改为“majority”。 • 不再支持**db.collection.ensureIndex()**，您可以使用**db.collection.createIndex()**代替。 • 从4.4.9 SERVER-53154开始，对 `saslStart` 以及 `saslContinue` 命令的参数进行严格校验，无法兼容gopkg/mgo.v2。 `saslContinue` 只需要 `conversationId` 和 `payload` 参数，而 `mgo` 提供了一个多余的参数 `mechanism` 。 更多信息，请参见Compatibility Changes in MongoDB 5.0。
MongoDB 4.4	• compact不再支持force选项。 • 不再支持geoSearch。 > **说明** **geoSearch**在MongoDB 5.0被删除。 • 支持在主库和备库上的同时进行索引创建操作，以减少索引创建带来的主备延迟。即使在索引创建过程中，也可以保证备库访问到最新的数据。不需要用户先分别在主库和备库轮询创建索引，然后再切换。 更多信息，请参见Compatibility Changes in MongoDB 4.4。

数据库大版本	变更说明
MongoDB 4.2	• 不再支持geoNear，您可以使用 $geoNear （aggregation）代替。 • 不再支持repairDatabase。 • 不再支持cloneCollection，您可以使用mongoexport和mongoimport代替。 说明 cloneCollection在MongoDB 4.2被删除。 • 不再支持mapReduce。 • 不能通过**db.collection.dropIndex("*")**的方式删除所有非 `_id` 索引，您可以使用**db.collection.dropIndexes()**代替。 • 统一前后台索引创建机制，使用了只在索引创建的开始和结束阶段对集合加独占锁的加锁机制。 说明 ◦ 在加独占锁时间段内不支持无锁，防止元数据被更改。 ◦ 弃用**db.collection.createIndex(keys, options, commitQuorum)**中的可选参数options的值background，如果在命令中为该参数指定了具体的值，MongoDB将忽略该选项。 • 对count中的选项名称进行更严格的验证。如果您指定未知的选项名称，则在MongoDB 4.2执行该命令会出错。 • cursor.min和cursor.max需要指定索引。 • 官方MongoDB 4.2+兼容驱动默认启用Retryable Writes。 更多信息，请参见Compatibility Changes in MongoDB 4.2。
MongoDB 4.0	• reIndex会加一个全局写锁，直到索引重建完成。 • 不再支持copydb和clone。 说明 copydb和clone在MongoDB 4.2被删除。 更多信息，请参见Compatibility Changes in MongoDB 4.0。

数据库大版本	变更说明

MongoDB 3.6	在MongoDB 3.4和MongoDB 4.0间，官方MongoDB还支持MongoDB 3.6，所以在将MongoDB 3.6以下的大版本升级至MongoDB 4.0或MongoDB 4.0以上大版本的过程中，变更内容还需包括如下内容：
	• aggregate不再支持返回单个文档，而是返回cursor，用户可以通过cursor指定 `batch size` 。
	• `$type: "array"` 能直接检测到数组类型的文档（能够同时检测到示例中 `_id` 为1和2的文档），之前只能检测到嵌套型的数组类型文档（仅能检测到示例中 `_id` 为2的文档）。**$type**的更多信息，请参见$type。
	示例：
	`{"_id":1,"a":[1,2,3]}`
	`{"_id":2,"a":[1,2,[3,4]]}`
	• 数组排序结果，发生以下变更：
	◦ find中，新增可选项sort，用于提供排序结果明细。
	◦ $sort (aggregation)中，$sort stage的内存限制为100 MB，如果超出该限制，$sort将产生错误。
	• 在进行更新操作时，如果需要同时更新多个字段，新字段将按照字典顺序添加。具体请参见$set。
	• 将ISODate（日期类型）的值转换成字符串返回；支持毫秒（ms），且在末尾添加Z。
	• 不再支持snapshot查询选项。

MongoDB 3.4	• 不再支持**group**，您可以使用**aggregate**中的$group (aggregation)代替。
	说明 group在MongoDB 4.2被删除。
	• 使用 `$in` 表达式进行匹配 `+ upsert: true` 的update。
	示例：
	`db.c.drop()`
	`db.c.update({a:{$in:[1]}},{$addToSet:{a:2}},{upsert:true}) //在MongoDB 3.4插入会失败，MongoDB 3.4之前的大版本可成功插入一条记录。`
	`db.c.update({a:{$elemMatch:{$in:[2]}}},{$addToSet:{a:2}},{upsert:true}) //在MongoDB 3.4可成功插入一条记录。`
	更多信息，请参见Compatibility Changes in MongoDB 3.4。

相关API

接口	说明
UpgradeDBInstanceEngineVersion	升级云数据库MongoDB实例的数据库大版本。

9.版本及存储引擎

本文介绍云数据库MongoDB支持的版本、引擎及版本和引擎之间的适配关系，帮助您选择适合您的业务需求的实例。

支持版本

云数据库MongoDB支持如下版本：

> **说明** 您可以在实例运行期间手动升级数据库版本，但版本升级后不支持降级。详情请参见升级数据库版本。

- MongoDB 5.0
- MongoDB 4.4
- MongoDB 4.2
- MongoDB 4.0
- MongoDB 3.4
- MongoDB 3.2

  已下线，详情请参见【通知】2月4日云数据库MongoDB版下线3.2版本，上线4.2版本。

存储引擎

存储引擎	适用场景	说明
WiredTiger	默认存储引擎，适用于大多数业务场景。	基于BTree结构组织数据，相比MongoDB早期的MMAPv1存储引擎性能提升明显，且支持数据压缩，存储成本更低。

版本和存储引擎的适配关系

存储引擎	5.0版本	4.4版本	4.2版本	4.0版本	3.4版本
WiredTiger	• 副本集实例 • 分片集群实例	• 副本集实例 • 分片集群实例	• 副本集实例 • 分片集群实例	• 单节点实例 • 副本集实例 • 分片集群实例	• 单节点实例 • 副本集实例 • 分片集群实例

MongoDB 5.0版本说明

标志着一个新的发布周期的到来，以更快地交付新特性给到用户。

- 原生时间序列平台

  通过原生支持整个时间序列数据的生命周期（从采集、存储、查询、实时分析和可视化，到在线归档或随着数据老化自动失效），使构建和运行时间序列应用程序的速度更快、成本更低。随着MongoDB 5.0的发布，MongoDB扩展了通用的应用数据平台，使开发能够更容易地处理时间序列数据，进一步扩展其在物联网、金融分析、物流等方面的应用场景。

- 在线重新分片

您可以在业务运行（数据不断增长）的情况下，按需改变集合的分片键（Shard key），而不需要数据库停机或在数据集合中进行复杂的迁移。您只需要在MongoDB Shell中运行reshardCollection命令，选择您需要重新分片的数据库和集合，指定新的分片键即可。

```
reshardCollection: "<database>.<collection>", key: <shardkey>
```

> 说明
> - <database>：需要重新分片的数据库名称。
> - <collection>：需要重新分片的集合名称。
> - <shardkey>：分片键的名称。
> - 当您调用reshardCollection命令时，MongoDB会克隆现有集合，然后将现有集合中所有oplog应用到新集合中，当所有oplog被使用后，MongoDB会自动切换到新集合，并在后台删除旧集合。

- 版本化API

  版本化API支持MongoDB灵活地在每个版本中为数据库添加新的功能和改进内容（以新版本兼容早期版本的方式）。当您需要改变API时，可以增加新版本的API，并与现有版本化的API在同一台服务器上同时运行。随着MongoDB版本发布的加速，版本化API能够使您更快、更轻松地使用到MongoDB最新版本的功能特性。

  版本化API定义了应用程序最常用的一组命令和参数（无论是数据库在年度重大发布还是季度快速发布期间，这些命令均不会改变）。通过将应用程序生命周期和数据库生命周期解耦，您可以将驱动程序固定在MongoDB API的特定版本上，即使数据库发生升级和改进，您的应用程序将可以继续运行数年而不需要修改代码。

- Write Concern默认Majority级别

  从MongoDB 5.0开始，Write Concern默认级别为majority，仅当写入操作被应用到Primary节点（主节点）且被持久化到大多数副本节点的日志中的时候，才会提交并返回成功，“开箱即用”地提供了更强的数据可靠性保障。

- 长时间运行的快照查询

  长时间运行的快照查询（Long-Running Snapshot Queries）增加了应用程序的通用性和弹性。您可以通过该功能运行默认时间为5分钟的查询（或将其调整为自定义持续时间），同时保持与实时事务性数据库一致的快照隔离，也可以在Secondary节点（从节点）上进行快照查询，从而在单个集群中运行不同的工作负载，并将其扩展到不同的分片上。

- 新版MongoDB Shell

  为了提供更好的用户体验，MongoDB 5.0从头开始重新设计了MongoDB Shell（mongosh），以提供一个更现代化的命令行体验，以及增强可用性的功能和强大的脚本环境。新版MongoDB Shell已经成为MongoDB平台的默认Shell。新版MongoDB Shell引入了语法高亮、智能自动完成、上下文帮助和有用的错误信息，为您创造一个直观、互动的体验。

- 版本发布调整

  从MongoDB 5.0开始，MongoDB发布的版本将分为Major Release（大版本）和Rapid Releases（快速发布版本），其中Rapid Releases作为开发版本提供下载和测试体验，但不建议用在生产环境。

MongoDB 4.4版本说明

针对之前版本中用户呼声最高的痛点重点进行了改进。

- Hidden Indexes

隐藏现有的索引，保证该索引在后续的查询中不会被使用，用来观察目标低效率索引的删除是否会导致业务性能抖动，如不造成影响即可放心删除该低效率索引。

- Refinable Shard Keys

  增加一个或多个Suffix Field来改善现有的文档在Chunk上的分布问题，避免所有访问集中在某个单一分片上，分散服务器的压力。

- Compound Hashed Shard Keys

  支持在复合索引中指定单个哈希字段，在极大程度上简化业务逻辑的复杂性。

- Hedged Reads

  在分片集群实例下，支持将一个读请求同时发送给某个分片中的两个副本集成员，并选择响应最快的返回结果来恢复客户端，以实现降低请求延迟的目的。

- Streaming Replication

  主库Oplog主动流向备库，相比之前版本的备库轮询方式，节省了近一半的往返时间，提升了主备复制的性能。

- Simultaneous Indexing

  主库与备库的索引创建操作同步进行，大幅减少主备库在索引创建过程中产生的延迟，保证备库能及时访问到最新的数据。

- Mirrored Reads

  主节点会按一定的比例把读流量复制到备库上执行，保证从节点承载一定的读流量，缓解业务的访问延迟。

- Resumable Initial Sync

  在主备库全量同步过程中，提供断点续传功能，避免因网络断连而导致全量同步工作从头进行。

- Time-Based Oplog Retention

  支持自定义指定Oplog的保留时长，避免主库Oplog被清理触发全量同步。

- Union

  新增$unionWith stage用于实现类似于SQL的 `union all` 功能，增强了MongoDB的查询能力。

- Custom Aggregation Expressions

  新增$accumulator和$function用来实现自定义的Aggregation Expression，完善接口统一性和用户体验。

> **说明** 更多MongoDB 4.4版本的新功能，请参见MongoDB 4.4功能概览。

MongoDB 4.2版本说明

采用二段提交方式，保证分片集群事务的ACID特性，极大拓展了适用的业务场景。

- 分布式事务

  采用二段提交方式，保证分片集群事务的ACID特性，极大拓展了MongoDB的业务场景，实现从NoSQL到NewSQL的飞跃。

- 可重试读

  增加可重试读功能，提供弱网环境下自动重试能力，降低业务端的逻辑复杂性，保证用户业务的连续性。

- 通配符索引

  对于非确定字段，支持创建通配符索引覆盖一个文档下的多个特征字段，管理方便且使用灵活。

- 字段级加密

  驱动层面支持字段级加密，可以针对特定的敏感信息（例如账号、密码、价格、手机号等）单独加密。避免全库加密，提升业务灵活性和安全性。

- 物化视图

  通过最新的物化视图可以缓存计算结果，避免重复计算提升运行效率，减少逻辑复杂度。

MongoDB 4.0版本说明

更适用于金融等对事务有依赖且使用NoSQL特性的场景。

- 跨文档事务支持

  首个支持跨文档事务的NoSQL云数据库，将文档模型的速度，灵活性、功能与ACID保证相结合。

- 迁移速度提升40%

  并发的读取和写入，使得新增的Shard节点能更快地完成数据迁移以承载业务压力。

- 读性能大幅扩展

  借助事务特性，Secondary节点不再因为同步日志而阻塞读取请求。 阿里云同时在全体系版本支持多节点扩展功能，可大幅提升业务读取能力。

MongoDB 3.4版本说明

在性能和安全性等方面较3.2版本均有不同程度的提升。

> **说明** MongoDB 3.2版本已下线，详情请参见【通知】2月4日云数据库MongoDB版下线3.2版本，上线4.2版本。

- 更快的主备同步

  在同步数据的同时建立所有索引（以前的版本仅建立_id索引）。同时在同步数据的阶段，Secondary节点不断读取新的oplog信息，确保Secondary节点的local数据库具备足够的存储空间来存储临时数据。

- 更高效的负载均衡

  3.2及以前版本中，分片集群的负载均衡由Mongos节点负责，多个Mongos节点会抢一个分布式锁，由抢锁成功的Mongos节点执行负载均衡任务，在Shard节点间迁移块；而在3.4版本中，负载均衡由ConfigServer节点中的Primary节点负责，负载均衡的并发度和效率上均有大幅提升。

- 更丰富的aggregation操作

  在3.4版本增加了大量的aggregation操作符，可支持更强的数据分析能力。例如，`bucket`能便捷地对数据进行分类；`$grahpLookup`相较于3.2版本的`$lookup`能支持更复杂的关系运算；`$addFields`使得文档操作更丰富（例如将某些字段求和存储为新的字段）。

- 支持Sharding Zones

  分片集群里引入了Zone的概念，主要取代现在的tag-aware sharding机制，能将某些数据分配到指定的一个或多个Shard节点中，该特性极大地方便sharding cluster的跨机房部署。

- 支持Collation

  在之前的版本里，文档里存储的字符串不论是中英文还是大小写，一律按字节来对比；引入Collation后，支持对字符串的内容进行解读，可以按使用的locale进行对比，也支持在对比时忽略大小写。

- 支持只读视图（Read-only views）

  在3.4中增加了对只读视图的支持，可以将集合中满足某个查询条件的数据虚拟成一个特殊的集合，用户可以在特殊的集合上做进一步的查询操作。

10.基本概念

本文将向您介绍云数据库MongoDB帮助文档中，相关名词和术语的解释。

概念	说明
地域	• 地域（Region）指的是用户所购买的云数据库MongoDB实例的服务器所处的地理位置。用户需要在开通云数据库MongoDB实例时指定地域，购买实例后暂不支持更改。 • 在购买云数据库MongoDB实例时，需要搭配阿里云服务器ECS使用，云数据库MongoDB支持内网访问，在地域选择时需要与ECS相同。关于内网连接云数据库MongoDB详情请参见MongoDB跨可用区内网访问实例。
可用区	• 可用区是指在同一地域下，电力、网络隔离的物理区域。 • 可用区之间内网互通，可用区内网络延时更小，不同可用区之间故障隔离。 • 单可用区是指云数据库MongoDB实例副本集中的三个节点处于相同的可用区。如果ECS和MongoDB部署在相同的可用区，网络延迟更小。
实例	• 云数据库MongoDB实例，或简称实例，是用户购买云数据库MongoDB服务的基本单位。 • 实例是阿里云数据库MongoDB版的运行环境，在主机上以单独的进程存在。 • 用户可通过控制台来创建、修改和删除云数据库MongoDB实例。各实例之间相互独立、资源隔离，相互之间不存在CPU、内存、IO等抢占问题。 • 每个实例拥有其自己的特性，例如数据库类型、版本等，系统有相应的参数来控制实例行为。
内存	云数据库MongoDB实例可以使用的内存上限。
磁盘容量	• 磁盘容量是用户购买云数据库MongoDB实例时，所选择购买的磁盘大小。 • 实例所占用的磁盘容量除集合数据外，还有实例正常运行所需要的空间，如系统数据库、数据库回滚日志、重做日志、索引等。 • 请确保云数据库MongoDB实例具有足够的磁盘容量来存储数据，否则可能导致实例被锁定。若因磁盘容量不足导致实例被锁定，用户可购买更大的磁盘容量来解锁实例。
IOPS	以4 KB为单位，每秒进行块设备读写操作的次数上限。
CPU核	实例可以使用的计算能力上限。 1个CPU拥有不低于2.3 GHz超线程（Intel Xeon系列Hyper-Threading）的计算能力。
连接数	客户端和云数据库MongoDB实例之间的TCP连接。 如果客户端使用了连接池，则客户端和云数据库MongoDB实例之间的连接为长连接，反之则为短连接。
分片集群	云数据库MongoDB支持分片集群架构，用户可以购买多个Mongos、多个Shard节点和一个ConfigServer组成分片集群，轻松得到一个MongoDB分布式数据库系统。

概念	说明
Mongos	• MongoDB分片集群请求入口，所有的请求都通过Mongos进行协调，Mongos是一个请求分发中心，它负责把对应的数据请求转发到对应的Shard服务器上。 • 用户可以选择多个Mongos作为请求的入口，防止其中一个挂掉所有的MongoDB请求都无法操作。
Shard	• MongoDB分片集群中的分片。 • 单个Shard是由三节点的副本集组成，保证单个分片的高可用性，用户可以根据自己的应用性能及存储要求，购买多个Shard来扩展读写性能及存储空间，实现一个分布式数据库系统。
ConfigServer	• 配置服务器，存储所有数据库元信息（路由、分片）的配置。Mongos本身没有存储，只是将Shard服务器和数据路由信息缓存在其内存里，配置服务器则实际存储（落盘）了这些数据。 • Mongos第一次启动或者关掉重启就会从ConfigServer加载配置信息，以后如果配置服务器信息变化会通知到所有的Mongos更新自己的状态，这样Mongos就能继续准确路由。 • ConfigServer存储了分片路由的元数据，服务可用性和数据可靠性要求极高，云数据库MongoDB采用三节点副本集的方式全方位保障ConfigSever的服务可靠性。