文档数据库服务

开发指南

文档版本 01

发布日期 2025-05-22

华为云计算技术有限公司

商标声明


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

注意

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

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

目 录

1 数据库使用规范

1.1 基础命令规范

1.1.1 查询命令

需要通过分析执行过程（查询计划）进行检查并优化，以避免慢查询。

db.collection.find().explain()

请参见**性能相关**，更多详细内容请参见**官方文档**。

注意事项

- 查询的结果，返回的是一个Currsor。Cursor使用完毕后要及时关闭，否则会产生内存堆积。
- 根据查询条件创建必要的索引，索引设计请参见**索引设计规范**。
  - 避免COLLSCAN全表扫描。
  - 查询条件和索引字段匹配有顺序性。
- 对于集群实例，根据业务对表进行合理地分片，分片设计请参见**分片设计规范**。
- 对于分片表，查询条件要基于shardKey进行，避免出现scatter-gather等不必要的查询。详情请参见**Distributed Queries**。
- 查询可以指定readConcern级别，详情请参见**Read Concern**。
- 查询可以指定readPerference参数，详情请参见**Read Preference**。

1.1.2 写入/更新命令

- 用户做了大量写入/更新操作后，实际数据量可能没有多大变化，但磁盘空间使用量增加了。是因为无论是写入、更新还是删除，以及索引插入和删除等操作，在后台实际上都会转成写入。因为底层的存储引擎（WiredTiger和RocksDB）采用都是appendOnly机制。只有当存储引擎内部数据状态满足一定条件后，会触发内部的compaction操作，进行数据压缩，进而释放磁盘空间。
- 写入/更新操作会涉及到备节点同步，可以指定writeConcern级别。详情请参见**Write Concern**。

注意事项

- update和upsert：upsert内部会先做一次查询，如果查询不存在则插入，否则执行update。如果业务上场景可以进行相关记录，则避免使用upsert命令，建议直接使用update或insert。
- update也需要匹配索引。
  - 避免COLLSCAN全表扫描。
  - 查询条件和索引字段匹配有顺序性。
- insert/update命令中涉及的文档，最大不能超过16MB。

⚠ 注意

使用写入/更新命令修改业务数据会造成业务数据变更。

1.1.3 删除命令

- 删除分为逻辑删除（基于条件的remove删除）和快速删除（dropCollection，dropDatabase）。
- 用户做了大量删除操作后，实际数据量可能没有多大变化，但磁盘空间使用量增加了。是因为无论是写入、更新还是删除，以及索引插入和删除等操作，在后台实际上都会转成写入。因为底层的存储引擎（WiredTiger和RocksDB）采用都是appendOnly机制。只有当存储引擎内部数据状态满足一定条件后，会触发内部的compaction操作，进行数据压缩，进而释放磁盘空间。
- 如果整个数据库不需要了，可以执行dropDatabase命令进行删除而不是逻辑删除，这样快速释放磁盘空间。
- 删除操作会涉及到备节点同步，可以指定writeConcern级别。详情请参见**Write Concern**。

注意事项

- 避免误删除。删除命令不能撤回，所以在删除前先执行**db**命令查看下当前所在的库。
- 一旦误删除后，恢复时注意以下几点：
  a. 根据历史备份文件，进行**备份**和**恢复**。
  b. 如果有备份实例，从备份实例中，通过**导入和导出**恢复。误删除后，恢复过程如果有新的数据写入，需要业务端考虑数据的一致性问题，即是否应该执行恢复。
  c. 实例最后的备份时间点至当前误删除时间点的数据，无法进行恢复。
- 删除命令，如果执行成功，则表示成功。如果执行失败，此时可能已经删除了部分数据了。所以不要继续使用删除的库表。建议继续下发删除命令，直到删除成功为止。

⚠ 注意

使用删除命令删除业务数据会造成业务数据丢失。

1.2 开发规范

数据库连接

使用DDS时，可能会遇到因为Mongod/dds mongos的连接数满了，导致客户端无法连接的问题。在Mongod/dds mongos的服务端，收到一个新的连接由一个单独的线程来处理，每个线程配置了1MB的栈空间，当网络连接数太多时，过多的线程会导致上下文切换开销变大，同时内存开销也会上涨。

- 客户端连接数据库的时候，要计算业务一共有多少个客户端，每个客户端配置的连接池大小是多少，总的连接数不要超过当前实例能承受的最大连接数的80%。
- 客户端与数据库的连接应尽量保持相对稳定的状态，每秒新增连接数建议保持在10以下。
- 建议客户端的连接超时时间至少设置为最大业务执行时长的3倍。
- 对于副本集实例，客户端需要同时配置主备节点的IP地址；对于集群实例，至少配置两个dds mongos的IP地址。
- DDS默认提供rwuser用户，使用rwuser用户登录时认证库必须是admin。

可靠性

write concern设置规则：对于关键业务，write concern设置为{w:n},n>0，数字越大，一致性实现更好，但性能较差。

- w:1表示实际写入主节点完成返回。
- w:1,journal:true表示写主节点和日志后返回。
- w:majority表示大多数备节点写入后返回。

说明

如果没有以w:majority写入数据，则发生主备倒换时，未同步到备机的数据有丢失风险。

对于可靠性有较高要求的，建议采用3AZ部署的集群。

性能相关

规范

- 业务程序禁止执行全表扫描的查询。
- 执行查询时，只选择需要返回的字段，不需要的字段不要返回。从而减少网络和进程处理的负载，修改数据时，只修改变化需要修改的字段，不要整个对象直接存储全部修改。
- 避免使用$not。DDS并不会对缺失的数据进行索引，因此$not的查询条件将会要求在一个结果集中扫描所有记录。如果$not是唯一的查询条件，会对集合执行全表扫描。
- 用$and时把匹配最少结果的条件放在最前面，用$or时把匹配最多结果的条件放在最前面。
- 单个实例中，数据库的总的个数不要超过200个，总的集合个数不要超过500个。集合数量过多会导致内存压力变高，并且集合数量多会导致重启以及主备倒换性能变差，影响紧急情况下的高可用性能。

- 业务上线前，一定要对数据库进行性能压测，评估业务峰值场景下，对数据库的负载情况。
- 禁止同时执行大量并发事务，且长时间不提交。
- 业务正式上线前， 所有的查询类别，都应该先执行查询计划检查查询性能。

建议

- 每个连接在后台都是由一个单独线程处理，每个线程会分配1MB的栈内存。所以连接数不宜过多，否则会占用过多的内存。
- 使用连接池，避免频繁地建立连接和断开连接，否则会导致CPU过高。
- 减少磁盘读写：避免使用不必要的upsert命令，避免查询不必要的数据。
- 优化数据分布：对数据进行分片，同时分散热点数据，均衡地使用实例资源。
- 减少锁冲突：避免对同一个Key过于频繁地操作。
- 减少锁等待：避免前台创建索引。

注意

开发过程中对集合的每一个操作都要通过执行explain()检查其执行计划，如：

db.T_DeviceData.find({"deviceId":"ae4b5769-896f"}).explain();

db.T_DeviceData.find({"deviceId":"77557c2-31b4"}).explain("executionStats");

对于查询而言，因为覆盖查询不需要读取文档，而是直接从索引中返回结果，这样的查询性能好，所以**尽可能使用索引覆盖查询**。如果explain()的输出显示indexOnly字段为真，则说明这个查询就被一个索引覆盖。

执行计划解析：

1. 看执行时间：executionStats.executionStages.executionTimeMillisEstimate和executionStats.executionStages.inputStage. executionTimeMillisEstimate时间越短越好。
   - executionStats.executionTimeMillis表示执行计划选择和执行的所有时间。
   - executionStats.executionStages.executionTimeMillisEstimate表示执行计划的执行完成时间。
   - executionStats.executionStages.inputStage. executionTimeMillisEstimate表示执行计划下的子阶段执行完成时间。
2. 看扫描条数：三个条目数相同为最佳。
   - executionStats. nReturned表示匹配查询条件的文档数。
   - executionStats .totalKeysExamined表示索引扫描条目数。
   - executionStats .totalDocsExamined表示文档扫描条目数。
3. 看Stage状态，性能较好的Stage状态组合如下。
   - Fetch+IDHACK
   - Fetch+ixscan
   - Limit+（ Fetch+ixscan ）
   - PROJECTION+ixscan

表 1-1 状态说明

状态名称	描述
COLLSCAN	全表扫描
SORT	内存中进行排序
IDHACK	根据_id进行查询
TEXT	全文索引
COUNTSCAN	未用索引计数
FETCH	索引扫描
LIMIT	使用Limit限制返回数
SUBPLA	未用索引的$or查询阶段
PROJECTION	限定返回字段时stage的返回
COUNT_SCAN	使用索引计数

Cursor 使用规则

如果cursor不使用了要立即关闭。由于cursor在10分钟内不活动，就会关闭，立即手动关闭会节省资源。

4.2 版本分布式事务使用规则

- Spring Data MongoDB不支持事务报错后重试机制，如果客户端使用Spring Data Mongo作为连接MongoDB的客户端，需要依照Spring Data Mongo的参考文档，使用Spring Retry进行事务的重试操作。
- 分布式事务操作数据的大小不能超过16MB。

备份相关注意事项

备份期间应避免进行DDL操作，规避备份失败的风险。

1.3 设计规范

命名规范

- 数据库对象（库名、表名、字段名、索引名）命名建议全部使用小写字母开头，后面跟字母或者数字，数据库对象（库名、表名、字段名、索引名）名字长度建议都不要大于32字节。
- 数据库名称不能使用特殊字符("",.,$,\,/,*,?,~,#,:,|")和空字符\0，数据库名称不能使用admin，local，config。
- 数据库集合名称建议使用字母和下划线组合，不能以system为前缀，<数据库名>.<集合名称> 总长度不超过120字符。

索引设计规范

索引创建，可以避免全表扫描，有效地提高查询命令的执行效率。

- 索引字段的长度不能超过512字节，索引名称长度不能超过64字符，单个复合索引所包含字段数最多不能超过16个。
- <数据库名>.<集合名>.$<索引名>的总长度不能超过128字符。
- 在高选择性字段上的创建索引。在低选择性字段上查询会返回较大的结果集。尽量避免返回较大的结果集。
- 对集合的写操作同时会操作集合上的索引，从而触发更多的IO操作，集合上的索引数量不要超过32。
- 不要创建不会被使用到的索引，因为DDS会加载索引到内存，无用索引加载到内存会浪费内存空间因业务逻辑变化而产生的无用索引也要及时清理。
- 索引创建必须使用后台创建索引，禁止前台创建索引。在集合前台创建索引的过程中，会持续持有父数据库的独占锁。这将导致该数据库及其所有集合上的所有读写操作被阻塞，直至索引创建完成。为避免此类阻塞问题，建议在创建索引时使用 background: true 选项，以非阻塞方式执行索引构建。具体操作请参考**创建索引。**
- 业务中查询，排序条件的key一定要创建索引，如果建立的是复合索引，索引的字段顺序要和这些关键字后面的字段顺序一致，否则索引不会被使用。
- 不要基于复合索引的靠前字段再创建索引。复合索引可以被用于一个索引中主要字段的查询。例如，对于复合索引(firstname,lastname)可以用于在firstname上的查询，再创建一个单独firstname的索引是不必要的。
- 创建索引会消耗较多的IO与计算资源，建议在业务低谷期进行索引创建，禁止同时并发创建超过5条索引。如果需要对同一集合创建多个索引，建议使用createIndexes命令一次性下发多条索引，可以减少性能损耗。
- 在计划删除某个索引前，请先核实备节点上是否正在创建索引。若发现备节点中有索引正在创建，禁止立即删除索引，此时删除索引可能会导致备节点陷入锁等待，进而引发一些意外的问题。

分片设计规范

对于使用DDS分片集群，建议尽可能地使用分片集合以充分利用性能，详情请参见**设置数据分片以充分利用分片性能**。

分片集合使用上建议如下：

- 对于大数据量（数据量过百万），并有较高读写请求的业务场景，数据量随着业务量增大而增大的，建议采用分片。
- 对于采用hash分片的集合，需要根据业务后面实际数据量大小，采用预分片，提前预置chunk数量，减少自动均衡和分裂对业务运行造成影响。
- 对于非空集合开启分片，应将均衡器的开启时间窗放在业务空闲时，避免分片间均衡数据与业务冲突影响性能。
- 需要基于分片键排序查询且增加数据时可以分布均匀建议使用范围分片，其他使用哈希分片。
- 合理设计shard key，防止出现大量的数据使用相同shard key，导致出现jumbo chunk。
- 使用分片集群，执行dropDatabase后，一定要执行**flushRouterConfig**命令，详情请参见**如何规避dds mongos路由缓存缺陷**。

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

2 数据库使用

2.1 连接数据库

文档数据库服务常用的连接方式如下表。

表 2-1 连接方式

连接方式	IP地址	使用场景	说明
内网连接（推荐）	内网IP地址	系统默认提供内网IP地址。 ● 当应用部署在弹性云服务器上，且该弹性云服务器与文档数据库实例处于同一区域、可用区、虚拟私有云子网内，建议单独使用内网IP通过弹性云服务器连接文档数据库实例。 ● 文档数据库服务和弹性云服务器在不同的安全组默认不能访问，需要在文档数据库服务所属安全组添加一条“入”的访问规则。 ● 文档数据库服务默认端口：8635，需要手动修改才能访问其它端口。	安全性高，可实现DDS的较好性能。
公网连接	弹性公网IP	● 当应用部署在弹性云服务器上，且该弹性云服务器与文档数据库实例处于不同区域时，建议单独使用弹性公网IP通过弹性云服务器连接文档数据库实例。 ● 当应用部署在其他云服务的系统上时，建议单独使用弹性公网IP通过弹性云服务器连接文档数据库实例。	降低安全性。

连接方式	IP地址	使用场景	说明
应用程序连接	内网IP地址	通过各类应用程序连接数据库。	• 通过Java连接数据库 • 通过Python连接数据库

2.2 创建和管理数据库

写入/更新和删除命令的规范请参见写入/更新命令和删除命令。

操作步骤

步骤1 创建info数据库。

use info

输入“db”当结果显示为如下信息，则表示当前已在这个数据库中。

```
info
```

步骤2 为数据库插入一条数据。

db.user.insert({"name": "joe"})

注意

对于DDS，隐式创建集合只有在内容插入后才会创建，即创建集合(数据表)后要再插入一个文档(记录)，集合才会真正创建。

步骤3 查看数据库。

查看所有数据库，可以使用如下命令。

show dbs

回显信息如下。

```
admin   0.000GB
config  0.000GB
info    0.000GB
local   0.083GB
```

步骤4 删除数据库。可以使用如下命令删除上面创建的info数据库：

db.dropDatabase()

回显信息如下表示删除成功。

```
{"dropped" : "info", "ok" : 1}
```

----结束

2.3 创建和管理集合

写入/更新和删除命令的规范请参见**写入/更新命令**和**删除命令**。

创建集合

步骤1 执行**db.createCollection(name, options)**创建集合。

```
db.createCollection(<name>, { capped: <boolean>,
                              autoIndexId: <boolean>,
                              size: <number>,
                              max: <number>,
                              storageEngine: <document>,
                              validator: <document>,
                              validationLevel: <string>,
                              validationAction: <string>,
                              indexOptionDefaults: <document>,
                              viewOn: <string>,
                              pipeline: <pipeline>,
                              collation: <document>,
                              writeConcern: <document>}
```

表 2-2 参数说明

字段	类型	说明
capped	boolean	可选的，如果要创建一个固定集合，该值为true，如果该值为ture，需要同时设置size字段。
autoIndexId	boolean	如果指定为false，表示禁止自动在_id字段创建索引。
size	number	可选，对于固定集合，指定集合的最大大小。
max	number	可选，对于固定集合，指定集合存储的最大的文档数。 更多详细参数说明请参考**官方文档**。

回显信息如下表示创建成功。

```
{ "ok" : 1 }
```

步骤2 为集合中插入一条数据。

db.coll.insert({"name": "sample"})

步骤3 查看已有集合。

show collections

步骤4 删除集合。

db.coll.drop()

----结束

创建固定集合

固定集合是指那些集合的大小或者文档数有最大值，方式集合的大小超过特定值，当集合的大小或者数量超过最大值后，集合的最早存储的值会被自动删除掉。

如下命令创建了一个集合，最大值是5MB，文档数量最多为5000。

db.createCollection("log", { capped : true, size : 5242880, max : 5000 })

创建分片集合

在DDS集群架构中，可以创建分片来充分利用数据库性能。创建分片的规范及建议请参见**分片设计规范**。

步骤1 使数据库可分片。

sh.enableSharding("info")

步骤2 创建分片表，并指定分片键。如下为info数据库的fruit集合分片，且分片键是"id"。

sh.shardCollection("info.fruit", {"_id": "hashed"})

说明

DDS分片集群支持两种分片策略：

- 范围分片，支持基于Shard Key的范围查询。
- 哈希分片，能够将写入均衡分布到各个Shard。

----结束

删除集合

执行**db.collection_name.drop()**删除集合。

2.4 创建和管理索引

DDS支持利用索引实现高效查询。如果没有索引，DDS必须执行集合扫描，即扫描集合中的每个文档，以选择那些与查询语句匹配的文档。如果一个查询存在适当的索引，DDS可以使用该索引来限制它必须检查的文档数量。

- 创建索引的规范及建议请参见**索引设计规范**。
- 写入/更新和删除命令的规范请参见**写入/更新命令**和**删除命令**。

索引分类

索引分类	说明
默认索引	在创建集合期间，DDS在_id字段上创建唯一索引。该索引可防止客户端插入两个具有相同值的文档。您不能将_id字段上的index删除。 在分片群集中，如果您不使用_id字段作为分片键，应用程序需要确保_id字段中值的唯一性以防止错误。这通常是通过使用标准的自动生成的ObjectId来完成的。
单字段索引	除DDS定义的_id索引外，DDS还支持在文档的单个字段上创建用户定义的升序/降序索引。 对于单字段索引和排序操作，索引键的排序顺序(升序或降序)并不重要，因为DDS可以从任何方向遍历索引。
复合索引	DDS还支持多个字段上的用户定义索引，即复合索引。 复合索引中列出的字段的顺序具有重要意义。例如，如果一个复合索引由{userid: 1, score: -1}组成，索引首先按userid排序，然后在每个userid值内按score排序。 对于复合索引和排序操作，索引键的排序顺序(升序或降序)可以决定索引是否支持排序操作。
多键索引	DDS使用多键索引来索引存储在数组中的内容。如果索引包含数组值的字段，DDS为数组的每个元素创建单独的索引项。这些多键索引允许查询通过匹配数组的一个或多个元素来选择包含数组的文档。DDS自动决定是否创建一个多键索引，如果索引字段包含数组值，您不需要显式地指定多键类型。

索引名称

索引的默认名称是索引键和索引中每个键的方向(即1或-1)的连接，使用下划线作为分隔符。例如，在**{ item : 1, quantity: -1 }**上创建的索引名称为**item*1_quantity*-1**。

您可以创建具有自定义名称的索引，比如比默认名称更易于阅读的索引。例如，考虑一个经常查询products集合以填充现有库存数据的应用程序。下面的createIndex() 方法在名为查询的商品和数量上创建一个索引:

db.products.createIndex({ item: 1, quantity: -1 } , { name: "query for inventory" })

您可以使用**db.collection.getIndexes()**方法查看索引名称。一旦创建索引，您将无法重命名。相反，您必须删除并使用新名称重新创建索引。

DDS提供了许多不同的索引类型来支持特定类型的数据和查询。

创建索引

步骤1 DDS使用如下命令创建索引:

db.collection.createIndex(keys, options)

- key 值为您要创建的索引字段，1 为指定按升序创建索引，-1代表降序创建索引。

- options接收可选参数，常用可选参数列表如下：

Parameter	Type	Description
background	Boolean	默认值为false。 建索引过程会阻塞其它数据库操作，background可指定以后台方式创建索引。
unique	Boolean	默认值为false。 建立的索引是否唯一。指定为true创建唯一索引。
name	string	索引的名称。如果未指定，MongoDB的通过连接索引的字段名和排序顺序生成一个索引名称。
expireAfterSeconds	integer	指定一个以秒为单位的数值，完成 TTL设定，设定集合的生存时间。

步骤2 创建索引。

- 单字段索引（Single Field Index）

  db.user.createIndex({"name": 1})

  上述语句针对name创建了单字段索引，其能加速对name字段的各种查询请求，是最常见的索引形式，DDS默认创建的id索引也是这种类型，{"name": 1} 代表升序索引，也可以通过{"name": -1}来指定降序索引，对于单字段索引，升序/降序效果是一样的。

- 复合索引（Compound Index）

  复合索引是单子索引的升级版本，它针对多个字段联合创建索引，先按第一个字段排序，第一个字段相同的文档按第二个字段排序，依次类推。

  db.user.createIndex({"name": 1, "age": 1})

- 多键索引
  - 当索引的字段为数组时，创建出的索引称为多键索引。
  - 多键索引会为数组的每个元素建立一条索引。

  如果为user集合加入一个habit字段（数组）用于描述兴趣爱好，需要查询有相同兴趣爱好的人就可以利用habit字段的多键索引。

  {“name”：“jack”，“age”：19, habit: [“football, runnning”]} //这是person表的一条用户信息。

  db.user.createIndex({"habit": 1}) //自动创建多key索引

  db.user.find({"habit": "football"}) //查询有相同兴趣爱好的人

步骤3 查看集合索引。

db.user.getIndexes()

步骤4 删除集合所有索引。

db.user.dropIndexes()

步骤5 删除集合指定索引。如下方式删除user集合中"name"索引。

db.user.dropIndex({"name": 1})

----结束

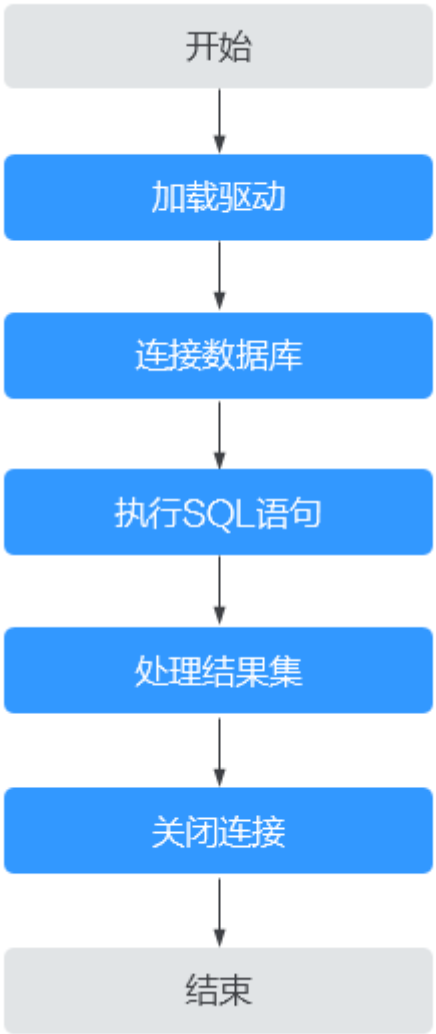
注意事项

DDS除了支持多种不同类型的索引，还能对索引定制一些特殊的属性。

- **唯一索引(unique index）**：保证索引对应的字段不会出现相同的值，比如_id索引就是唯一索引。
- **TTL索引**：可以针对某个时间字段，指定文档的过期时间（经过指定时间后过期或 在某个时间点过期）。
- **部分索引(partial index):** 只针对符合某个特定条件的文档建立索引。
- **稀疏索引(sparse index):** 只针对存在索引字段的文档建立索引，可看作是部分索引的一种特殊情况。

3 应用程序开发教程

3.1 开发流程



3.2 驱动侧通用参数配置

连接 DDS 常见配置项和推荐值

1. connectTimeoutMS连接超时时间确保驱动程序不会在连接阶段无限期等待。推荐配置：

   connectTimeoutMS = 10000ms

2. socketTimeoutMS防止TCP通信进入无限等待。推荐配置：

   时间为业务最长时间的2-3倍，最低不要小于10s。

   socketTimeoutMS = max（10000ms, 3倍最长业务时间）

3. minPoolSize连接池最小连接数。推荐配置：

   minPoolSize = 10

4. maxPoolSize连接池最大连接数。推荐配置：

   maxPoolSize = 50 - 100

5. maxIdleTimeMS连接在删除和关闭之前可以在池中保持的最大空闲时间。推荐配置：

   maxIdleTimeMS = 10000ms

⚠ 注意

不要使用socketTimeoutMS来阻止某个操作在数据库端的运行时长。需要使用**maxTimeMS**，以便服务器可以取消已经被客户端遗弃的操作。

3.3 基于 Java 开发

3.3.1 驱动包、环境依赖

DDS支持通过Java语言接口来操作数据，通过Java连接实例的方式有无需下载SSL证书连接和用户下载SSL证书连接两种，其中使用SSL证书连接加密功能，具有更高的安全性。

DDS新实例默认关闭SSL数据加密，开启SSL请参考**开启SSL**。

驱动安装

1. 进入Jar驱动**下载地址**下载连接DDS使用的驱动包“**mongo-java-driver-3.12.9.jar**”，此安装包提供了访问DDS数据库实例的API接口。
2. 驱动安装请参考**官方指南**。

环境类

客户端需配置JDK1.8。JDK是跨平台的，支持Windows，Linux等多种平台。

下面以Windows为例，介绍JDK配置流程：

步骤1 DOS窗口输入“java -version”，查看JDK版本，确认为JDK1.8版本。如果未安装JDK，请下载安装包并安装。

步骤2 在windows操作系统桌面中“此电脑”图标上单击右键，选择“属性”。

步骤3 在弹出的“系统”窗口中，单击左侧导航栏中“高级系统设置”。

步骤4 在弹出的“系统属性”窗口中，单击右下角的“环境变量”。

步骤5 在弹出的“环境变量”窗口中的“系统变量”区域框中设置如下变量名和变量值。

变量名	操作	变量值
JAVA_HOME	• 若存在，则单击“编辑”。 • 若不存在，则单击“新建”。	JAVA的安装目录。 例如：C:\Program Files\Java\jdk1.8.0_131
Path	编辑	• 若配置了JAVA_HOME，则在变量值的最前面加上： %JAVA_HOME%\bin; • 若未配置JAVA_HOME，则在变量值的最前面加上 JAVA安装的全路径：C:\Program Files\Java\jdk1.8.0_131\bin;
CLASSPATH	新建	.;%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar;

步骤6 单击“确定”，并依次关闭各窗口。

----结束

3.3.2 连接数据库

使用 SSL 证书连接

说明

该方式属于SSL连接模式，需要下载SSL证书，通过证书校验并连接数据库。

您可以在“实例管理”页面，单击实例名称进入“基本信息”页面，单击“数据库信息”模块“SSL”处的，下载根证书或捆绑包。

步骤1 通过Java连接MongoDB数据库，代码中的Java链接格式如下：

- 连接到单节点：
  mongodb://<username>:<password>**@**<instance_ip>:<instance_port>**/**<database_name>**?authSource=admin&ssl=true**
- 连接到副本集：
  mongodb://<username>:<password>**@**<instance_ip>:<instance_port>**/**<database_name>**?authSource=admin&replicaSet=replica&ssl=true**
- 连接到集群：
  mongodb://<username>:<password>**@**<instance_ip>:<instance_port>**/**<database_name>**?authSource=admin&ssl=true**

表 3-1 参数说明

参数	说明
<username>	当前用户名。
<password>	当前用户的密码。
<instance_ip>	如果通过弹性云服务器连接，“instance_ip”是主机IP，即“基本信息”页面该实例的“内网地址”。
	如果通过连接了公网的设备访问，“instance_ip”为该实例已绑定的“弹性公网IP”。
<instance_port>	端口，默认8635，当前端口，参考“基本信息”页面该实例的“数据库端口”。
<database_name>	数据库名，即需要连接的数据库名。
authSource	鉴权用户数据库，取值为admin。
ssl	连接模式，值为true代表是使用ssl连接模式。

连接MongoDB数据库的Java代码，可参考以下示例：

```
import java.util.ArrayList;
import java.util.List;
import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;
import com.mongodb.ServerAddress;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.MongoCollection;
import com.mongodb.MongoClientURI;
import com.mongodb.MongoClientOptions;
public class MongoDBJDBC {
public static void main(String[] args){
    try {
        System.setProperty("javax.net.ssl.trustStore", "/home/Mike/jdk1.8.0_112/jre/lib/
security/mongostore");
        // 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放(密码应密文存放、使用时解密)，确保安全；
        // 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
        String password = System.getenv("EXAMPLE_PASSWORD_ENV");
        System.setProperty("javax.net.ssl.trustStorePassword", password);
        ServerAddress serverAddress = new ServerAddress("ip", port);
        List addrs = new ArrayList();
        addrs.add(serverAddress);
        // 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放(密码应密文存放、使用时解密)，确保安全；
        // 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
        String userName = System.getenv("EXAMPLE_USERNAME_ENV");
        String rwuserPassword = System.getenv("EXAMPLE_PASSWORD_ENV");
        MongoCredential credential = MongoCredential.createScramSha1Credential("rwuser",
"admin", rwuserPassword.toCharArray());
```



```
            List credentials = new ArrayList();
            credentials.add(credential);
            MongoClientOptions opts= MongoClientOptions.builder()
            .sslEnabled(true)
            .sslInvalidHostNameAllowed(true)
            .build();
            MongoClient mongoClient = new MongoClient(addrs,credentials,opts);
            MongoDatabase mongoDatabase = mongoClient.getDatabase("testdb");
            MongoCollection collection = mongoDatabase.getCollection("testCollection");
            Document document = new Document("title", "MongoDB").
            append("description", "database").
            append("likes", 100).
            append("by", "Fly");
            List documents = new ArrayList();
            documents.add(document);
            collection.insertMany(documents);
            System.out.println("Connect to database successfully");
            } catch (Exception e) {
            System.err.println( e.getClass().getName() + ": " + e.getMessage() );
        }
    }
}
```

样例代码：

```
javac -cp .:mongo-java-driver-3.2.0.jar MongoDBJDBC.java
java -cp .:mongo-java-driver-3.2.0.jar MongoDBJDBC
```

----结束

无证书连接

说明

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

步骤1 通过Java连接MongoDB数据库实例，代码中的Java链接格式如下：

- 连接到单节点：
  mongodb://<username>:<password>**@**<instance_ip>:<instance_port>**/**<database_name>**?authSource=admin**
- 连接到副本集：
  mongodb://<username>:<password>**@**<instance_ip>:<instance_port>**/**<database_name>**?authSource=admin&replicaSet=replica**
- 连接到集群：
  mongodb://<username>:<password>**@**<instance_ip>:<instance_port>**/**<database_name>**?authSource=admin**

表 3-2 参数说明

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

参数	说明
	如果通过连接了公网的设备访问，“instance_ip”为该实例已绑定的“弹性公网IP”。
<instance_port>	端口，默认8635，当前端口，参考“基本信息”页面该实例的“数据库端口”。
<database_name>	数据库名，即需要连接的数据库名。
authSource	鉴权用户数据库，取值为admin。

连接MongoDB数据库的Java代码，可参考以下示例：

```
import com.mongodb.ConnectionString;
import com.mongodb.reactivestreams.client.MongoClients;
import com.mongodb.reactivestreams.client.MongoClient;
import com.mongodb.reactivestreams.client.MongoDatabase;
import com.mongodb.MongoClientSettings;
public class MyConnTest {
    final public static void main(String[] args) {
    try {
        // no ssl
        // 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放(密码应密文存放、使用时解密)，确保安全；
        // 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
        String userName = System.getenv("EXAMPLE_USERNAME_ENV");
        String rwuserPassword = System.getenv("EXAMPLE_PASSWORD_ENV");
    ConnectionString connString = new ConnectionString("mongodb://" + userName + ":" + rwuserPassword + "@192.*.*.*:8635,192.*.*.*:8635/test? authSource=admin");
    MongoClientSettings settings = MongoClientSettings.builder()
        .applyConnectionString(connString)
        .retryWrites(true)
        .build();
    MongoClient mongoClient = MongoClients.create(settings);
    MongoDatabase database = mongoClient.getDatabase("test");
     System.out.println("Connect to database successfully");
    } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Test failed");
        }
}
}
```

----结束

3.3.3 访问数据库

具体访问数据库前，引入如下相关类。

```
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import static com.mongodb.client.model.Filters.*;
import com.mongodb.client.model.CreateCollectionOptions;
import com.mongodb.client.model.ValidationOptions;
```

访问 DataBase

当已经有一个初始化好的MongoClient实例后，通过如下方式来访问一个database，示例如下：

```
MongoDatabase database = mongoClient.getDatabase("test");
```

访问集合

当获取一个MongoDatabase实例后，可以通过如下命令来得到要获取的集合：

```
MongoCollection<Document> coll = database.getCollection("testCollection");
```

显示的创建一个集合

也可以通过 createCollection()方法显示的创建一个集合，并在创建时候指定该集合的属性。

```
database.createCollection("testCollection", new CreateCollectionOptions()..sizeInBytes(200000))
```

插入数据

```
Document doc0 = new Document("name", "zhangsan")
                .append("age", 3)
                .append("gender", "male");

Document doc1 = new Document("name", "LiSi")
                .append("age", 2)
                .append("gender", "female");

Document doc2 = new Document("name", "wangmazi")
                .append("age", 5)
                .append("gender", "male");

List<Document> documents = new ArrayList<Document>();
documents.add(doc1);
documents.add(doc2);

collection.insertMany(documents);
```

删除数据

```
collection.deleteOne(eq("_id", new ObjectId("00000001")));
```

删除表

```
MongoCollection<Document> collection = database.getCollection("test");
collection.drop()
```

读数据

```
MongoCollection<Document> collection = database.getCollection("contacts");
MongoCursor<String> cursor = collection.find();
while (cursor.hasNext()) {
   Object result = cursor.next();
}
```

带过滤条件的查询

```
MongoCollection<Document> collection = database.getCollection("test");
MongoCursor<String> cursor = collection.find(
```

```
    new Document("name","zhangsan")
        .append("age: 5));
while (cursor.hasNext())  {
   Object result = cursor.next();
}
```

运行命令

执行buildInfo和collStats

```
MongoClient mongoClient = MongoClients.create();
MongoDatabase database = mongoClient.getDatabase("test");

Document buildInfoResults = database.runCommand(new Document("buildInfo", 1));
System.out.println(buildInfoResults.toJson());

Document collStatsResults = database.runCommand(new Document("collStats", "restaurants"));
System.out.println(collStatsResults.toJson());
```

创建建索引

```
MongoClient mongoClient = MongoClients.create();
MongoDatabase database = mongoClient.getDatabase("test");
MongoCollection<Document> collection = database.getCollection("test");
collection.createIndex(Indexes.ascending("age"));
```

3.3.4 完整示例

```
package mongodbdemo;
import org.bson.*;
import com.mongodb.*;
import com.mongodb.client.*;
public class MongodbDemo {
    public static void main(String[] args) {
        // 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放(密码应密文存放、使用时解密)，确保安全；
        // 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
        String userName = System.getenv("EXAMPLE_USERNAME_ENV");
        String rwuserPassword = System.getenv("EXAMPLE_PASSWORD_ENV");
        String mongoUri = "mongodb://" + userName + ":" + rwuserPassword + "@10.66.187.127:27017/admin";
        MongoClientURI connStr = new MongoClientURI(mongoUri);
        MongoClient mongoClient = new MongoClient(connStr);
        try {
            // 使用名为 someonedb 的数据库
            MongoDatabase database = mongoClient.getDatabase("someonedb");
            // 取得集合/表 someonetable 句柄
            MongoCollection<Document> collection = database.getCollection("someonetable");
             // 准备写入数据
            Document doc = new Document();
            doc.append("key", "value");
            doc.append("username", "jack");
            doc.append("age", 31);
             // 写入数据
            collection.insertOne(doc);
            System.out.println("insert document: " + doc);
             // 读取数据
            BsonDocument filter = new BsonDocument();
            filter.append("username", new BsonString("jack"));
```

```
            MongoCursor<Document> cursor = collection.find(filter).iterator();
            while (cursor.hasNext()) {
                System.out.println("find document: " + cursor.next());
            }
        } finally {
            //关闭连接
            mongoClient.close();
        }
    }
}
```

更多Java接口参考请参见**官方文档**。

3.4 基于 Python 开发

3.4.1 PyMongo 包

Python语言通过PyMongo来为DDS数据库提供统一访问接口，应用程序可基于PyMongo进行数据操作，PyMongo支持SSL连接，PyMongo内部通过连接池的方式支持多线程应用。

PyMongo的安装方式**官方指南**。

3.4.2 连接数据库

通过Python连接实例的方式有无需下载SSL证书连接和用户下载SSL证书连接两种，其中使用SSL证书连接通过了加密功能，具有更高的安全性。

DDS新实例默认关闭SSL数据加密，开启SSL请参考**开启SSL**。

前提条件

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

   curl *ip:port*

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

连接代码

- SSL开启

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

```
connection = MongoClient(conn_urls % (rwuser,
password),connectTimeoutMS=5000,ssl=True,
ssl_cert_reqs=ssl.CERT_REQUIRED,ssl_match_hostname=False,ssl_ca_certs=${path to
certificate authority file})
dbs = connection.database_names()
print "connect database success! database names is %s" % dbs
```

- SSL关闭

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

说明

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

3.4.3 访问数据库

假设客户端应用程序已经完成数据库连接，并初始化好一个 MongoClient client.

访问 DataBase

当已经有一个初始化好的MongoClient实例后，通过如下方式来访问一个database，示例如下：

```
db=client.test_database
```

或者采用如下方式指定：

```
db=client["test_database"]
```

访问集合

```
collection=db.test_collection
```

或者采用如下方式指定：

```
collection=db["test_collection"]
```

显示的创建一个集合

也可以通过 createCollection()方法显示的创建一个集合，并在创建时候指定该集合的属性。

```
collection = db.create_collection("test")
```

插入数据

```
student = {
    'id': '20170101',
```

```
    'name': 'Jordan',
    'age': 20,
    'gender': 'male'
}
result = collection.insert(student);
```

删除数据

```
result = collection.delete_one({'name': 'Kevin'})
```

删除表

```
db.drop_collection("test")
```

读数据

```
result = collection.find_one({'name': 'Mike'})
```

带过滤条件的查询

```
result = collection.find_one({"author":"Mike"}
```

运行命令

执行 buildInfo 和 collStats

```
db.command("collstats","test")
db.command("buildinfo")
```

计数

```
count = collection.find().count()db.command("buildinfo")
```

排序

```
results = collection.find().sort('name', pymongo.ASCENDING)
print([result['name'] for result in results])
```

创建索引

```
 result=db.profiles.create_index([('user_id',pymongo.ASCENDING)],... unique=True)
```

3.4.4 完整示例

```
#!/usr/bin/python
import pymongo
import random
import os
# 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放(密码应密文存放、使用时解密)，确保安全
# 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV
username = os.getenv('EXAMPLE_USERNAME_ENV')
password = os.getenv('EXAMPLE_PASSWORD_ENV')
mongodbUri = 'mongodb://%s:%s@10.66.187.127:27017/admin'
client = pymongo.MongoClient(mongodbUri % (username, password ))
db = client.somedb
db.user.drop()
element_num=10
for id in range(element_num):
```

```
  name = random.choice(['R9','cat','owen','lee','J'])
  gender = random.choice(['male','female'])
  db.user.insert_one({'id':id, 'name':name, 'gender':gender})
content = db.user.find()
for i in content:
  print i
```

更多PyMongo接口请参考**官方文档**。

3.5 基于 Golang 开发

3.5.1 驱动包

DDS支持通过Go语言接口来操作数据，通过Go连接实例的方式有开启SSL认证连接和关闭SSL认证连接两种，其中开启SSL证书连接加密功能，具有更高的安全性。

DDS新实例默认关闭SSL数据加密，开启SSL请参考**开启SSL**。

驱动下载

建议使用go mod下载驱动

```
require go.mongodb.org/mongo-driver v1.12.1
```

go文件中导入：

```
import (
"go.mongodb.org/mongo-driver/bson"
"go.mongodb.org/mongo-driver/mongo"
"go.mongodb.org/mongo-driver/mongo/options"
"go.mongodb.org/mongo-driver/mongo/readpref"
)
```

3.5.2 连接数据库

前提条件

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

   curl *ip:port*

   返回“It looks like you are trying to access MongoDB over HTTP on the native driver port.”，说明网络互通。
2. 如果开启SSL，需要在界面上下载根证书，并上传到弹性云服务器。

连接代码

- SSL开启

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

```
credential := options.Credential{
  AuthMechanism: "SCRAM-SHA-1",
  AuthSource:    "admin",
  Username:      username,
  Password:      password,
}
// 高可用 URI, 注意 SetDirect 设为 false
highProxyUri := "mongodb://host1:8635,host2:8635/?ssl=true"
clientOpts := options.Client().ApplyURI(highProxyUri)
clientOpts = clientOpts.SetTLSConfig(&tls.Config {
InsecureSkipVerify: true,
}).SetDirect(false).SetAuth(credential)
// 直连的 URI，注意 SetDirect 设为 true
//directUri := "mongodb://host:8635/?ssl=true"
//clientOpts := options.Client().ApplyURI(highProxyUri)
//clientOpts = clientOpts.SetTLSConfig(&tls.Config {
// InsecureSkipVerify: true,
//}).SetDirect(true).SetAuth(credential)
// 连接实例
ctx, cancel := context.WithTimeout(context.Background(), 5*time.Second)
defer cancel()
client, err := mongo.Connect(ctx, clientOpts)
if err != nil {
   fmt.Println("mongo实例连接失败：", err)
   return
}
// Ping 主节点
ctx, cancel = context.WithTimeout(context.Background(), 2*time.Second)
defer cancel()err = client.Ping(ctx, readpref.Primary())
if err != nil {fmt.Println("ping 主节点失败：",err)
   return
}
// 选择数据库和集合
collection := client.Database("test").Collection("numbers")
// 插入单条数据
ctx, cancel = context.WithTimeout(context.Background(), 5*time.Second)
defer cancel()
oneRes, err := collection.InsertOne(ctx, bson.D{{"name", "e"}, {"value", 2.718}})
if err != nil{fmt.Println("插入单条记录失败：",err)
   return
}else {
   fmt.Println(oneRes)
}
// 批量插入
ctx, cancel = context.WithTimeout(context.Background(), 100*time.Second)
defer cancel()
docs := make([]interface{}, 100)
for i := 0; i < 100; i++{
   docs[i] = bson.D{{"name", "name"+strconv.Itoa(i)}, {"value", i}}
}
manyRes, err := collection.InsertMany(ctx, docs)
if err != nil {
   fmt.Println("批量插入失败：",err)
   return
}else {
   fmt.Println(manyRes)
}
```

- SSL关闭

```
// 高可用连接 readPreference 取值 primary(默认)-只读主节点，primaryPreferred-主节点优先，如主节点不可用，则读从节点
// secondary-只读从节点，如从节点不可用会报错,secondaryPreferred-从节点优先，如从节点
```

```
不可用，则读主节点
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放(密码应密文存放、使用时解密)，确保安全；
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
username = System.getenv("EXAMPLE_USERNAME_ENV")
password = System.getenv("EXAMPLE_PASSWORD_ENV")
highProxyUri := fmt.Sprintf("mongodb://%v:%v@host1:8635,host2:8635/?authSource=admin&replicaSet=replica&readPreference=secondaryPreferred",username,password)
ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
defer cancel()
clientOpts := options.Client().ApplyURI(highProxyUri)
client, err := mongo.Connect(ctx, clientOpts)
// Ping 主节点
ctx, cancel = context.WithTimeout(context.Background(), 2*time.Second)
defer cancel()err = client.Ping(ctx, readpref.Primary())
if err != nil {
   fmt.Println("ping 主节点失败：",err)
   return
}
// 选择数据库和集合
collection := client.Database("test").Collection("numbers")
// 插入单条数据
ctx, cancel = context.WithTimeout(context.Background(), 5*time.Second)
defer cancel()
res, err := collection.InsertOne(ctx, bson.D{{"name", "e"}, {"value", 2.718}})
if err != nil{
   fmt.Println("插入单条记录失败：",err)
   return
}else {
   fmt.Println(res)
}
```

3.5.3 访问数据库

访问 DataBase

当已经有一个初始化好的MongoClient实例后，通过如下方式来访问一个database，示例如下：

```
db:= client.Database("test")
```

访问集合

当获取一个MongoDatabase实例后，可以通过如下命令来得到要获取的集合：

```
coll := db.Collection("testCollection")
```

显示的创建一个集合

也可以通过 CreateCollection()方法显示的创建一个集合，并在创建时候指定该集合的属性。

```
db:= client.Database("test")
ctx, cancel = context.WithTimeout(context.Background(), 5*time.Second)
defer cancel()
sizeInBytes := int64(200000)
testCollection :=
```

```
db.CreateCollection(ctx,"testCollection",&options.CreateCollectionOptions{SizeInBytes:
&sizeInBytes})
```

3.5.4 完整示例

注意事项

1. 业务的Context超时时间建议设置不低于10秒。
2. 同时在以下业务场景下，一定要配置MaxTimeMS：
   - Find
   - FindAndModify
   - DropIndexes
   - Distinct
   - Aggregate
   - CreateIndexes
   - Count

代码示例

```
import (
    "context"
    "fmt"
    "strconv"
    "time"

    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"
    "go.mongodb.org/mongo-driver/mongo/readpref"
)

const (
    ConnectTimeout        = 10 * time.Second
    SocketTimeout         = 60 * time.Second
    MaxIdleTime           = 10 * time.Second
    MaxPoolSize           = 100
    MinPoolSize           = 10
    DefaultContextTimeOut = 10 * time.Second
    MaxTimeMS             = 10 * time.Second
)

func main() {

    // 高可用连接串
    // 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存
放(密码应密文存放、使用时解密)，确保安全；
    // 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量
(环境变量名称请根据自身情况进行设置)EXAMPLE_USERNAME_ENV和
EXAMPLE_PASSWORD_ENV。
    username = System.getenv("EXAMPLE_USERNAME_ENV")
    password = System.getenv("EXAMPLE_PASSWORD_ENV")
    highProxyUri := fmt.Sprintf("mongodb://%v:%v@host1:8635,host2:8635/?

authSource=admin&replicaSet=replica&readPreference=secondaryPreferred",username,password
)
    clientOpts := options.Client().ApplyURI(highProxyUri)
```

```
    clientOpts.SetConnectTimeout(ConnectTimeout)
    clientOpts.SetSocketTimeout(SocketTimeout)
    clientOpts.SetMaxConnIdleTime(MaxIdleTime)
    clientOpts.SetMaxPoolSize(MaxPoolSize)
    clientOpts.SetMinPoolSize(MinPoolSize)

    // 连接数据库
    ConnectCtx, cancel := context.WithTimeout(context.Background(), ConnectTimeout)
    defer cancel()
    client, err := mongo.Connect(ConnectCtx, clientOpts)
    if err != nil {
        fmt.Println("mongo实例连接失败：", err)
        return
    }
    // Ping 主节点
    ctx, cancel := context.WithTimeout(context.Background(), DefaultContextTimeOut)
    defer cancel()
    err = client.Ping(ctx, readpref.Primary())
    if err != nil {
        fmt.Println("ping 主节点失败：", err)
        return
    }
    // 选择数据库和集合
    collection := client.Database("test").Collection("numbers")
    // 插入单条数据
    ctx, cancel = context.WithTimeout(context.Background(), DefaultContextTimeOut)
    defer cancel()
    oneRes, err := collection.InsertOne(ctx, bson.D{{"name", "e"}, {"value", 2.718}})
    if err != nil {
        fmt.Println("插入单条记录失败：", err)
        return
    } else {
        fmt.Println(oneRes)
    }
    // 批量插入
    ctx, cancel = context.WithTimeout(context.Background(), DefaultContextTimeOut)
    defer cancel()
    docs := make([]interface{}, 100)
    for i := 0; i < 100; i++ {
        docs[i] = bson.D{{"name", "name" + strconv.Itoa(i)}, {"value", i}}
    }
    manyRes, err := collection.InsertMany(ctx, docs)
    if err != nil {
        fmt.Println("批量插入失败：", err)
        return
    } else {
        fmt.Println(manyRes)
    }
    db := client.Database("test")
    // 分页查询
    ctx, cancel = context.WithTimeout(context.Background(), DefaultContextTimeOut)
    defer cancel()
    cursor, err := db.Collection("numbers").Find(ctx, struct{}{},
options.Find().SetBatchSize(100).SetMaxTime(MaxTimeMS).SetSkip(int64(1000)).SetLimit(100))
    if err != nil {
        fmt.Println("分页查询失败：", err)
        return
    }
    for cursor.Next(ctx) {
        fmt.Println(cursor.Current)
    }
}
```


3.6 更多教程

更多开发教程请参见**官方文档**。

4 管理数据库权限

4.1 默认权限机制

DDS与社区原生的版本相比，对安全进行一系列的增强，以应对越来越严峻的安全挑战。社区原生版本支持不鉴权的方式连接使用数据库的，而DDS采取默认安全策略，连接数据库必须通过鉴权，否则，无法使用数据库。

- 数据库实例创建后，系统会创建默认的管理员用户rwuser，但是需要需要客户指定，并满足密码复杂度要求。
- 通过此用户创建和管理客户自定义的角色和用户。该用户无默认密码，需要客户指定，并且需要满足密码复杂度要求。

使用场景

在执行mongodump和mongorestore操作时，如果对全实例进行备份恢复，则会出现权限验证失败。因为rwuser用户对实例上的admin库和config库权限受限。此时，明确指定用户业务相关库表，即可正常操作。

4.2 角色管理

DDS通过基于角色的管理来控制用户对数据访问的权限，角色共分为两类：预置角色和自定义角色。

4.2.1 预置角色

预置角色是系统自动生成的角色信息，客户端可用的预置角色名称有read，readWrite。

mongodb使用角色来管理数据库的，所以创建一个用户时就需要赋予一个角色。角色除了内置之外，也可以**自定义角色**。

表 4-1 常见内置角色

角色	权限描述	包含的操作命令
read	read角色包含读取所有非系统集合数据和订阅部分系统集合(system.indexes、system.js、system.namespaces)的权限。	changeStream、collStats、dbHash、dbStats、find、killCursors、listIndexes、listCollections
readWrite	readWrite角色包含read角色的权限同时增加了对非系统集合数据的修改权限，但只对系统集合system.js有修改权限。	collStats、convertToCapped、createCollection、dbHash、dbStats、dropCollection、createIndex、dropIndex、find、insert、killCursors、listIndexes、listCollections、remove、renameCollectionSameDB、update
readAnyDatabase	readAnyDatabase角色包含对除了config和local之外所有数据库的只读权限。同时对于整个集群包含listDatabases命令操作。	在MongoDB3.4版本之前，该角色包含对config和local数据库的读取权限。当前版本如果需要对这两个数据库进行读取，则需要在admin数据库授予用户对这两个数据库的read角色。
readWriteAnyDatabase	readWriteAnyDatabase角色包含对除了config和local之外所有数据库的读写权限。同时对于整个集群包含listDatabases命令操作。	在MongoDB3.4版本之前，该角色包含对config和local数据库的读写权限。当前版本如果需要对这两个数据库进行读写，则需要在admin数据库授予用户对这两个数据库的readWrite角色。
dbAdmin	dbAdmin角色包含执行某些管理任务(与schema相关、索引、收集统计信息)的权限，该角色不包含用户和角色管理的权限。	• 对于系统集合(system.indexes、system.namespaces、system.profile)包含命令操作：collStats、dbHash、dbStats、find、killCursors、listIndexes、listCollections、dropCollection and createCollection(仅适用system.profile) • 对于非系统集合包含命令操作：bypassDocumentValidation、collMod、collStats、compact、convertToCapped、createCollection、createIndex、dbStats、dropCollection、dropDatabase、dropIndex、enableProfiler、reIndex、renameCollectionSameDB、repairDatabase、storageDetails、validate

角色	权限描述	包含的操作命令
dbAdminAnyDatabase	dbAdminAnyDatabase角色包含类似于dbAdmin角色对于所有数据库管理权限，除了config数据库和local数据库。同时对于整个集群包含listDatabases命令操作。	在MongoDB3.4版本之前，该角色包含对config和local数据库的管理权限。当前版本如果需要对这两个数据库进行管理，则需要在admin数据库授予用户对这两个数据库的dbAdmin角色。
clusterAdmin	clusterAdmin角色包含MongoDB集群管理最高的操作权限。	clusterManager、clusterMonitor和hostManager三个角色的所有权限，并且还拥有dropDatabase操作命令的权限。
userAdmin	userAdmin角色包含对当前数据库创建和修改角色和用户的权限。该角色允许向其它任何用户(包括自身)授予任何权限，所以这个角色也提供间接对超级用户(root)的访问权限，如果限定在admin数据中，也包括集群管理的权限。	changeCustomData、changePassword、createRole、createUser、dropRole、dropUser、grantRole、revokeRole、setAuthenticationRestriction、viewRole、viewUser
userAdminAnyDatabase	userAdminAnyDatabase角色包含类似于userAdmin角色对于所有数据库的用户管理权限，除了config数据库和local数据库。	• 对于集群包含命令操作：authSchemaUpgrade、invalidateUserCache、listDatabases • 对于系统集合admin.system.users和admin.system.roles包含命令操作：collStats、dbHash、dbStats、find、killCursors、planCacheRead、createIndex、dropIndex

4.2.2 自定义角色

用户自定义角色是用户通过命令创建的定制化的角色，只包含CRUD操作的一种或多种，或者内置角色的一种或多种。可以通过命令针对不同的资源、不同的操作进行自定义，除了预置角色无法被修改以外，其他应用方式是相同的。

创建、修改和删除角色

- 要创建角色前，需使用具有权限的用户（可使用rwuser）连接到数据库实例。详情请参见**连接数据库**。
- 通过createRole创建自定义角色，可以针对不同db，不同的collection进行权限控制，也可以从其他角色上继承。
- 角色创建完成后如需调整权限可通过grantPrivilegesToRole，grantRolesToRole，revokeRolesFromRole或revokePrivilegesFromRole命令获取或者回收回权限。详情请参见**创建并管理角色**。

4.2.3 创建并管理角色

创建角色

db.createRole(role, writeConcern)

- 参数role为必选参数，类型为文档，详情如下：

```
{
  role: "<name>",
  privileges: [
     { resource: { <resource> }, actions: [ "<action>", ... ] },
     ...
  ],
  roles: [
     { role: "<role>", db: "<database>" } | "<role>",
      ...
  ],
  authenticationRestrictions: [
    {
      clientSource: ["<IP>" | "<CIDR range>", ...],
      serverAddress: ["<IP>" | "<CIDR range>", ...]
    },
    ...
  ]
}
```

各参数及含义如下：

字段	类型	说明
role	string	角色名称
privileges	数组	必选参数，数组元素为角色具备的权限。 该参数设置为空集合意味着该角色无任何权限。
resource	文档	用于指定库名或者集合名

字段	类型	说明
actions	数组	对应资源可用的操作列表, 常用action如下： • find • count • getMore • listDatabases • listCollections • listIndexes • insert • update • remove 更多actions请参见**官方文档**。
roles	数组	必选参数，数组元素为角色继承的角色名。 可以是系统预置的角色read或者readWrite，也可以是用户自定义的角色。
authenticationRestrictions	数组	可选。用于指定该角色可接入的IP地址或者IP地址段。

- writeConcern可选参数用来指定命令的write concern的等级。

更新角色

db.grantPrivilegesToRole(rolename,privileges,writeConcern)

db.revokePrivilegesFromRole(rolename,privileges,writeConcern)

上述命令用户给角色获取或者回收指定的权限。

- 参数rolename为必选参数，用于指定待更新的角色名称。
- 参数privileges为角色待调整的权限。

```
db.grantPrivilegesToRole(
   "< rolename >",
   [
      { resource: { <resource> }, actions: [ "<action>", ... ] },
      ...
   ],
   { < writeConcern > }
)
```


表 4-2 privileges 各字段含义

字段	类型	说明
resource	文档	用于指定库名或者集合名
actions	数组	参考createRole章节的说明。

除了上述命令，还支持updateRole对角色信息整体更新。

db.updateRole(role, update, writeConcern)

表 4-3 参数说明

字段	类型	说明
role	string	角色名称
update	数组	必选参数，与创建角色命令的privileges参与含义一致。用于整体替换对应角色的全部权限信息。
writeConcern	文档	可选参数用来指定命令的 write concern 的等级

删除角色

db.dropRole(rolename, writeConcern)

- 参数rolename为必选参数，待删除的角色名称。
- writeConcern可选参数用来指定命令的write concern的等级。

4.3 用户管理

DDS上用户的权限都是基于角色管理，通过给用户赋予不同的角色来进行差异化的权限控制。

为了给文档数据库实例提供管理服务，您在创建数据库实例时，文档数据库服务会自动为实例创建admin、monitor和backup账户。如果试图删掉、重命名、修改这些账户的密码和权限，会导致出错。

对于数据库管理员账户rwuser，以及您所创建的账户，允许修改账户的密码。

4.3.1 创建用户

操作须知

- 下面所有操作都对权限要求，默认rwuser用户具备所需权限，如果通过用户自定义用户进行管理，则需要关注是否具备操作权限。

- 使用具备权限的用户（可使用rwuser）连接到数据库实例。
- 通过createUser创建所需的用户，通过设置合适的角色来控制对应用户的权限。其中需要注意的是"passwordDigestor" 参数必须是 "server"，否则命令会执行失败，增加这个约束是为了避免安全隐患。

创建用户

db.createUser(user, writeConcern）

- 命令中user为必选参数，类型为文档，包含了要创建用户的身份认证和访问信息。
- writeConcern为可选参数，类型为文档，包含了创建操作的write concern级别。

user文档定义了用户，其格式如下:

```
{
  user: "<name>",
  pwd: "<cleartext password>",
  customData: { <any information> },
  roles: [
    { role: "<role>", db: "<database>" } | "<role>",
    ...
  ],
  authenticationRestrictions: [
     {
       clientSource: ["<IP>" | "<CIDR range>", ...],
       serverAddress: ["<IP>" | "<CIDR range>", ...]
     },
     ...
  ]
  mechanisms: [ "<SCRAM-SHA-1|SCRAM-SHA-256>", ... ],
  passwordDigestor: "<server|client>"
}
```

表 4-4 user 参数说明

字段	类型	说明
user	string	新用户名称。
pwd	string	用户密码，如果您在$external数据库上运行db.createUser()以创建将凭据存储在 MongoDB外部的用户，则pwd字段不是必需的。
customData	文档	可选的。任何任意信息，该字段可用于存储管理员希望与此特定用户关联的任何数据。例如，这可以是用户的全名或员工ID。
roles	数组	授予用户的角色。可以指定一个空数组[]来创建没有角色的用户。

字段	类型	说明
authenticationRestrictions	数组	可选的。服务器对创建的用户强制执行的身份验证限制，用于指定该角色可接入的IP地址或者IP地址段。
mechanisms	数组	可选的。指定用于创建SCRAM用户凭据的特定SCRAM机制。当前只包含SCRAM-SHA-1和SCRAM-SHA-256。
passwordDigestor	string	可选的。指示是在server端还是client端验证密码，默认是server。

示例

- **创建用户时不同数据库赋予不同角色**

  使用db.createUser()在products数据库中创建accountAdmin01用户。

  ```
  use products
  db.createUser( { user: "accountAdmin01",
                   pwd: "Changeme_123",
                   customData: { employeeId: 12345 },
                   roles: [ { role: "clusterAdmin", db: "admin" },
                            { role: "readAnyDatabase", db: "admin" },
                            "readWrite"] },
                 { w: "majority" , wtimeout: 5000 } )
  ```

  以上操作赋予用户accountAdmin01以下角色：

  - 在admin数据库中角色为clusterAdmin和readAnyDatabase。
  - 在products数据库中角色为readWrite。

- **创建用户时同一数据库赋予多个角色**

  以下操作创建一个在products数据库中角色为readWrite和dbAdmin的用户，用户名为accountUser。

  ```
  use products
  db.createUser(
    {
      user: "accountUser",
      pwd: "Changeme_123",
      roles: [ "readWrite", "dbAdmin" ]
    }
  )
  ```

- **创建用户时不赋予角色**

  以下操作在admin数据库中创建一个用户名为reportsUser的用户，但是没有赋予用户角色。

  ```
  use admin
  db.createUser(
    {
      user: "reportsUser",
      pwd: "Chagneme_123",
  ```

```
    roles: [ ]
  }
)
```

- **创建管理员用户并赋予角色**

  下面的操作将在admin 数据库中创建名为appAdmin的用户，并赋予该用户对config数据库的读写访问权限，使用户可以更改分片群集的某些设置，如分片均衡器设置。

```
use admin
db.createUser(
  {
    user: "appAdmin",
    pwd: "Changeme_123",
    roles:
      [
        { role: "readWrite", db: "config" },
        "clusterAdmin"
      ]
  }
)
```

- **创建有身份验证限制的用户**

  以下操作将在管理员数据库中创建一个名为restricted的用户。该用户只能从IP地址192.0.2.0连接到IP地址198.51.100.0时才能进行身份验证。

```
use admin
db.createUser(
  {
    user: "restricted",
    pwd: "Changeme_123",
    roles: [ { role: "readWrite", db: "reporting" } ],
    authenticationRestrictions: [ {
       clientSource: ["192.0.2.0"],
       serverAddress: ["198.51.100.0"]
    } ]
  }
)
```

- **仅使用SCRAM-SHA-256证书创建用户**

  下面的操作将创建一个只有SCRAM-SHA-256凭据的用户。

```
use reporting
db.createUser(
  {
    user: "reportUser256",
    pwd: "Changeme_123",
    roles: [ { role: "readWrite", db: "reporting" } ],
    mechanisms: [ "SCRAM-SHA-256" ]
  }
)
```

  如果设置了authenticationMechanisms参数，mechanisms字段只能包含authenticationMechanisms参数中指定的值。

4.3.2 更新用户

db.updateUser(username, update, writeConcern)

- 命令中参数username为要更新的用户名。
- update为文档类型，包含用户替换数据的文档。

- writeConcern为可选参数， 更新操作的write concern级别。

```
db.updateUser(
  "<username>",
  {
    customData : { <any information> },
    roles : [
      { role: "<role>", db: "<database>" } | "<role>",
      ...
    ],
    pwd: passwordPrompt(),      // Or  "<cleartext password>"
    authenticationRestrictions: [
      {
        clientSource: ["<IP>" | "<CIDR range>", ...],
        serverAddress: ["<IP>", | "<CIDR range>", ...]
      },
      ...
    ],
    mechanisms: [ "<SCRAM-SHA-1|SCRAM-SHA-256>", ... ],
    passwordDigestor: "<server|client>"
  },
  writeConcern: { <write concern> }
)
```

表 4-5 update 参数说明

字段	类型	说明
customData	文档	可选。任意信息
roles	数组	可选。授予用户的角色。roles数组的更新将覆盖以前的数组的值。
pwd	string	可选。用户密码
authenticationRestrictions	数组	可选。服务器对用户实施的身份验证限制，用于指定该角色可接入的IP地址或者IP地址段。
mechanisms	数组	可选的。指定用于创建SCRAM用户凭据的特定SCRAM机制。当前只包含SCRAM-SHA-1 和SCRAM-SHA-256。
passwordDigestor	string	可选的。指示是在server端还是client端验证密码，默认是server

示例

- **更新用户信息**

  products数据库中的用户appClient01，其信息如下：

  ```
  {
     "_id" : "products.appClient01",
  ```



```
  "token" : NumberLong("8424642624807814713"),
  "user" : "appClient01",
  "db" : "products",
  "customData" : {
     "empID" : "12345",
     "badge" : "9156"
  },
  "roles" : [
     {
        "role" : "readWrite",
        "db" : "products"
     },
     {
        "role" : "read",
        "db" : "inventory"
     }
  ],
  "mechanisms" : [
     "SCRAM-SHA-1",
     "SCRAM-SHA-256"
  ]
}
```

下面操作会替换用户的自定义数据和角色数据：

```
use products
db.updateUser( "appClient01",
{
  customData : { employeeId : "0x3039" },
  roles : [
    { role : "read", db : "assets"  }
  ]
} )
```

products数据库中的用户appClient01，经过更新后信息如下：

```
{
  "_id" : "products.appClient01",
  "token" : NumberLong("8424642624807814713"),
  "user" : "appClient01",
  "db" : "products",
  "customData" : {
     "employeeId" : "0x3039"
  },
  "roles" : [
     {
        "role" : "read",
        "db" : "assets"
     }
  ],
  "mechanisms" : [
     "SCRAM-SHA-1",
     "SCRAM-SHA-256"
  ]
}
```

- **更新用户以仅使用SCRAM-SHA-256凭证**

  reporting数据库中的用户reportUser256，其信息如下：

```
{
  "_id" : "reporting.reportUser256",
  "token" : NumberLong("2827251846225877395"),
  "user" : "reportUser256",
  "db" : "reporting",
  "roles" : [ ],
```



```
    "mechanisms" : [
        "SCRAM-SHA-1",
        "SCRAM-SHA-256"
    ]
}
```

以下操作会将当前同时拥有 SCRAM-SHA-256 和 SCRAM-SHA-1 全权证书的用户更新为只拥有 SCRAM-SHA-256 全权证书的用户。

⚠ 注意

- 如果密码未与mechanisms一起指定，则只能将mechanisms更新为用户当前SCRAM机制的子集。
- 如果密码与mechanisms一起指定，则可以指定任何受支持的SCRAM机制。
- 对于SCRAM-SHA-256，passwordDigestor必须是默认值 "server"。

```
db.updateUser(
  "reportUser256",
  {
    mechanisms: [ "SCRAM-SHA-256" ]
  }
)
```

reporting数据库中的用户reportUser256，经过更新后信息如下：

```
{
    "_id" : "reporting.reportUser256",
    "token" : NumberLong("2827251846225877395"),
    "user" : "reportUser256",
    "db" : "reporting",
    "roles" : [ ],
    "mechanisms" : [
        "SCRAM-SHA-256"
    ]
}
```

4.3.3 删除用户

db.dropUser(username, writeConcern)

- username为要从数据库中删除的用户名。
- writeConcern为可选参数， 移除操作的writeConcern级别。

示例

下面是操作将reportUser1用户从产品数据库中删除。

```
use products
db.dropUser("reportUser1", {w: "majority", wtimeout: 5000})
```

删除成功后执行db.getUser将显示null。

```
replica:PRIMARY> db.getUser("reportUser1")
null
```

5 系统集合

表 5-1 4.0 版本系统集合

系统集合	说明
admin.system.roles	存储创建并分配给用户的自定义角色，以提供对特定资源的访问权限。
admin.system.users	存储用户的身份验证凭据以及分配给该用户的所有角色。
admin.system.version	存储用户凭证文档的架构版本。
<database>.system.namespaces	包含了数据库中所有的集合信息。
<database>.system.indexes	列出了数据库中的所有索引。
<database>.system.profile	包含该数据库的慢日志信息。
<database>.system.js	包含用于服务器端 JavaScript的特殊 JavaScript 代码。
<database>.system.views	包含了该数据库的每个view信息。

6 常用操作

6.1 常用 CRUD 操作

选择对应的数据库版本后，您可以了解MongoDB常用的CRUD操作。详情请参见**官方文档**。