云数据库 MongoDB

开发规范

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

开发规范

最近更新时间：2025-12-16 15:44:42

数据库设计规范

禁止类

禁止数据库中创建过多的表

在 WiredTiger 引擎中，每个集合都需要创建多个文件来保存元数据、数据及索引，磁盘上过多的小文件会导致性能下降。建议单个数据库表个数控制在100个以内，整个数据库实例表数量控制在2000个以内。

建议类

- 建议数据库名以 db 开头，不能包含除 _ 以外的特殊字符，所有字母全部小写，数据库名不超过64个字符。
- 建议集合名以 t_开头，不能包含除 _ 以外的特殊字符、集合名不超过120个字符。

索引设计规范

禁止类

- **禁止线上库不带 background 参数建索引**

  MongoDB 4.2之前的版本，createIndex() 命令默认是 foreground 模式，这种模式下创建索引会阻塞数据库的所有操作，造成业务中断，线上业务执行 createIndex() 务必添加 background参数。

  > ⚠ **注意：**
  > background 需要在 createIndex() 命令的 options 参数中，示例：`db.test.createIndex({a: 1}, {unique:true, background: true})`，切勿将不同的参数分开，示例：`db.test.createIndex({a: 1}, {unique:true}, {background: true})` 。

- **排序字段需要放到索引中，避免业务大量在内存中排序造成数据库 OOM（Out Of Memory）**
  - MongoDB 4.2及之前的版本，一条 sql 默认只允许使用32MB内存进行排序，如果超出会提示 `Sort operation used more than the maximum 33554432 bytes of RAM` 错误，此时可以通过执行 `db.runCommand({ getParameter : 1, "internalQueryExecMaxBlockingSortBytes" : 1 })` 调整排序内存。
    但是线上业务禁止调整，因为这样会让数据库 OOM 的概率增大。建议将排序字段加到索引中，通过索引排序实现排序能力。
  - MongoDB 4.4版本，虽然提供了磁盘排序的选项以避免排序消耗大量内存，但是建议最好使用索引排序。
- **禁止在索引同步过程中触发同命名空间的索引删除操作**

  在 MongoDB 4.4 以下版本中，索引创建是异步的。从节点通过回放 oplog 同步索引时，若对其所在集合执行 dropIndexes 操作，将引发锁冲突。此冲突会阻塞 oplog 回放线程，导致后续操作（包括新连接）排队等

待。持续的阻塞将迅速耗尽连接资源，最终使从节点无法响应新请求，服务不可用。
  - 临时方案：索引创建后，需确认所有分片的主从节点均构建完成，方可进行后续操作。
  - 根本方案：升级至 MongoDB 4.4及以上版本，其并发索引构建机制可彻底规避此问题。
- **禁止一个表内创建过多的索引**

  MongoDB 插入每条数据的时候同时需要写索引。索引越多，写入数据时就要花费更多的代价。因此禁止对索引的滥用，如对每个字段都建立索引，哪怕不会根据该字段进行查询。建议单表索引最多不超过10个。

建议类

- **建议定期清理无用索引**

  索引在写操作时会带来额外的资源消耗，因此需要尽量精简索引。
  MongoDB 4.4之后的版本，建议使用 hidden index 先隐藏无用的索引，隐藏后业务确认正常再删除索引。
- **按照最左匹配原则，如果单列索引已经被复合索引包含，建议删除**

  额外的索引会造成写操作时性能浪费。
- **建议尽量避免使用 $ne/$nin 等操作**

  和其他数据库（如 MySQL）一样，不等于 not in 类的操作无法有效利用索引，尽量应该避免。
- **建议在区分度较大的字段上建立索引**

  如果索引字段区分度较小，查询扫描的行数依然会比较多，查询效率较低，对数据库负载影响较大。因此建立索引的字段尽量应该有较大的区分度。
- **索引构建期间保持对各分片实例状态的监控，发现重启事件后立即检查并补齐索引**

  MongoDB 4.4 以下版本缺乏索引构建的持久化和跨节点协调机制，各分片独立执行索引构建，进度状态仅保存在本地内存中。若主节点在构建过程中重启，内存状态丢失且无法自动恢复续建，导致索引处于不完整状态，需手动在该分片上重新执行索引构建。
- **大集合创建索引构建期间建议关闭 Balancer，等加完再打开 Balancer**

  MongoDB 4.4 以下版本中，索引构建的批量键插入阶段会持有意向排他锁（IX 锁），而 chunk 迁移（moveChunk）内部操作同样需要获取相关锁资源。两者在并发执行时产生锁冲突，索引构建会阻塞 chunk 迁移进度，chunk 迁移也会延缓索引构建，导致分片集群在索引维护期间出现迁移停滞、请求延迟增加等问题。

数据库操作规范

禁止类

- **禁止上线未经过 explain() 确认执行计划的 sql**

  上线 sql 前需要 explain() 确认执行计划是否符合预期，否则上线可能会引起故障。
- **线上环境禁止关闭鉴权，特别是开放外网访问的数据库**

  关闭鉴权会将数据库暴露给所有人，特别是数据库服务器开通了外网。建议线上环境打开鉴权。如果一定要关闭鉴权，务必设置防火墙规则或 IP 白名单。
- **禁止在 admin 库、local 中存储业务数据**

  admin 库读写时会加 db 锁，影响性能；local库只会保存到本地，不会复制到从节点，如果发生主从切换会丢

失数据。因此禁止使用 admin 库和 local 库。

- **分片集群禁止执行 db.dropDatabase() 命令后再创建同名的 db**
  - MongoDB 4.0及之前的版本，官方文件要求删除 db 并创建同名 db 后，业务读写数据前需要在所有 mongos 节点上执行重启或 flushRouterConfig 命令。
  - MongoDB 4.2版本，官方文件要求删除 db 命令执行完以后，再执行一次删除 db 命令，并在所有 mongos 节点上执行重启或 flushRouterConfig 命令。MongoDB 4.4版本，官方文件要求删除 db 命令执行完以后，再执行一次删除 db 命令，方可重建同名的 db。
  - MongoDB 5.0及以上版本，执行一次删除 db 命令即可。

  因此禁止在业务代码中直接进行 db.dropDatabase() 命令后再创建同名的 db。运维人员在做该操作时，请务必按照官方文档要求进行所有必要的操作。
- **高并发高性能场景，禁止过度使用 in 和 or**

  in 或者 or 条件语句在数据库底层需要转换成多次查询，过多的 in 和 or 操作在高并发高性能场景，会严重影响请求的响应时延及数据库负载。
- **高并发高性能场景，禁止将复杂的运算操作交给数据库进行**

  MongoDB 提供了强大的计算能力（如 MapReduce 等），这些特性对开发人员非常友好，极大减轻了业务逻辑。但是这些运算不可避免是需要资源的，如果将复杂运算下沉到数据库层，高并发场景势必会给数据库造成极大的负担，数据库一旦故障会造成整个系统雪崩。

  建议在高并发高性能的场景下，数据库操作保持简单，复杂的运算交给服务器并适当在数据库前端增加缓存。
- **线上业务禁止直接进行批量数据 remove**

  remove 命令到数据库后会先查询符合删除条件记录的 _id，之后一条条按照 _id 进行删除，并记录到 oplog 中（删除每条记录都会写一条 oplog）。

  当满足 remove 条件的数据较多时对数据库压力较大，且极容易引起主从延迟突然增大。

  线上业务建议直接用 drop 集合或用脚本一条条删除并控制删除速度。或尽量使用 ttl 索引。
- **业务禁止自定义 _id 字段**

  _id 是 MongoDB 内部的默认主键，默认这是一个自增的序列。如果自定义 _id 并且业务无法保证 _id 递增，每次插入数据后，_id 索引不可避免需要对 B 树索引进行调整，这将对数据库带来额外的负担。
- **副本集直连 mongod 节点的场景，禁止只在连接串中配置单个 IP；分片集群禁止只连接单个 mongos 地址**

  线上业务如果只连接副本集主节点，一旦数据库发生 HA 会造成写入中断；如果只连接单个 mongos，这个 mongos 故障后会造成业务中断。
- **线上业务禁止设置 Write Concern j:false**

  Write Concern 默认一般为 j:true，表示服务端会写入 journal log 完成后再向 client 端返回。一般请勿设置 j:false，否则进程突然故障重启后，可能会造成数据丢失。
- **update 语句中禁止不带条件的更新**

  推荐保持 multi 为默认值（false），避免程序 bug（如由于某些异常造成 query 参数传了{}）造成全表数据更新。
- **禁止更新数组内部分元素时，将数组全部拿出来更新后再写回去**

  推荐使用 arrayFilters 仅对需要的元素进行修改。

建议类

- **建议局部读写而不是全读全写**
  查询语句中应尽量使用 $projection 运算符投影出需要的字段；在 update 命令中如果只是修改某个字段，建议使用 $set，请勿将文档全部读出来修改后再全量写进去。
- **线上环境慎重使用 db.collection.renameCollection() 命令**
  renameCollection() 在4.0及之前的版本会阻塞 db 的所有操作；在4.2及其之后版本会阻塞当前表及目标表的操作。而且 renameCollection() 执行期间会造成游标失效、changeStream 失效及带 --oplog 命令的 mongodump 失败等问题。线上环境禁止高峰期直接操作。
- **建议核心业务配置 WriteConcern 为 {w: "majority"} 参数**
  默认情况下，一般驱动的 WriteConcern 配置为 {w:1}，即在主节点写入完成后认为请求成功。如果机器突然发生故障并且写入的数据还未复制到从节点，这样的配置会导致数据丢失。因此对于线上的核心业务，建议配置 {w: "majority"}，这样的配置会等数据同步大多数节点后再返回客户端。当然可靠性和性能不能兼顾，选择了 {w: “majority” } 配置后请求的延迟也会相应的增加。
- **建议在低峰期执行 dropDatabase 操作**
  MongoDB 4.0 中 dropDatabase 命令需获取数据库级别排他锁（X锁），必须等待所有读写锁释放后才能执行。当存在长查询、写入操作、未关闭游标或后台索引构建时会产生阻塞，而该版本新增的多文档事务可能长时间持有锁，进一步加剧了 dropDatabase 的阻塞风险。
- **建议设置事务超时时间**
  MongoDB 4.0 中事务操作与 oplog 读取共享同一读取 tickets 池。极端情况可能因为事务持有 tickets 并等待 oplog 同步，而 oplog 读取又因等待 tickets 被阻塞，两者形成循环等待，导致死锁。
- **建议使用 MongoDB 官方推荐的事务回调 API 编写事务逻辑**
  事务回调 API 内部自动处理了事务重试逻辑（例如遇到 TransientTransactionError 等错误），能在发生网络波动或短暂锁冲突时安全地重试事务，显著提升事务成功率并减少因手动重试逻辑缺失造成的事务挂起风险。

不建议类

- **除非必要，不要在高性能场景大量使用多文档事务**
  MongoDB 4.0及之后的版本，MongoDB 提供了多文档事务。但是多文档事务只是 MongoDB 数据库能力的补充，在高并发高性能场景下，大规模使用多文档事务需要进行充分的压测。
  一般来说，多文档事务提交前需要在内存中保留快照，这可能消耗大量的 cache 从而导致性能下降。
- **不建议使用短连接**
  MongoDB 的认证逻辑是一个比较复杂的运算过程，而且默认 MongoDB 会为每个连接创建一个线程。大量短连接会对数据库产生较大的负担，特别是没有 mongos 的副本集集群。建议使用长连接，详细参考 MongoDB URL 中 Connection Pool 参数。

分片集群设计规范

禁止类

- **如果使用 _id 字段作为片键，禁止使用范围分片**
  id 默认是一个递增的序列，随着数据量的增加会一直增大。如果 _id 作为片键并使用范围分片，集群随着数据的插入不断的进行 balance。
- **分片集群禁止直连 mongod 节点写数据**
  分片集群应该通过 mongos 写数据，直接通过 mongod 写入的数据无路由信息，会导致访问不到。
- **线上环境禁止长时间关闭 balancer 和 autoSplit 配置**
  关闭 balance 会导致片之间数据不均衡，关闭 autoSplit 可能会产生 jumbo chunk。
- **分片表尽量避免不带片键的查询**
  分片表不带片键进行查询，需要扫描所有分片后在 mongos 聚合结果，比较消耗性能，不推荐使用。
- **线上环境务必设置 balancer 窗口，避免 balance 对业务造成影响**
  balance 过程会明显对数据库造成较大的压力，建议放在业务低峰期进行。

建议类

- **建议使用区分度较大的字段作为片键，最理想的情况是使用唯一主键作为片键**
  假设我们有一个存储人口信息的集合，其使用性别作为片键，这个片键认为区分度较低，因为集合中性别字段相同的数据理论上会有一半之多。
  如果片键区分度不大，可能导致大量的记录集中在某些片上，而这种不均衡也无法添加分片进行扩展。因此建议使用区分度较大的字段作为片键。
- **如果使用 hash 分片，建议进行预分配，特别是表比较大且经常需要大量插入数据**
  shardCollection() 命令默认每个分片只会创建2个 chunk，随着数据量的越来越大，MongoDB 需要不断的 balance 和 splitChunk，这将对数据库带来较大的负担。
  因此在对于大集合，建议提前进行预分片（shardCollection 命令指定 numInitialChunks 参数，每个分片最大支持8192个），特别是向大集合中批量导数据。

不建议类

- **没有按片键顺序扫描的强需求，不建议使用 range 分片，推荐 hash 分片**
  range 分片容易引起不均衡和数据热点，而且因为无法预分片所以随着数据的写入 balance 不可避免，因此不建议使用，除非有特殊的按片键范围查询需求。

  > ⚠ **注意：**
  > 在 shardCollection 的时候，sh.shardCollection("records.people", { zipcode: 1 }) 命令中1表示范围分片，sh.shardCollection("records.people", { zipcode: "hashed" }) 命令中"hashed"表示 hash 分片。需注意不要用错。

- **分片集群中不建议使用非分片表**
  MongoDB 的分片集群如果未执行 shardCollection 命令，默认数据只存储在主分片上。大量未分片的表会造成分片和分片间的数据量不一致。集群长时间运行下去，可能会造成某些片数据量特别多甚至会打满磁盘，运维在这种情况下不得不使用 movePrimary 手动进行数据搬迁，从而增加了运维复杂度。