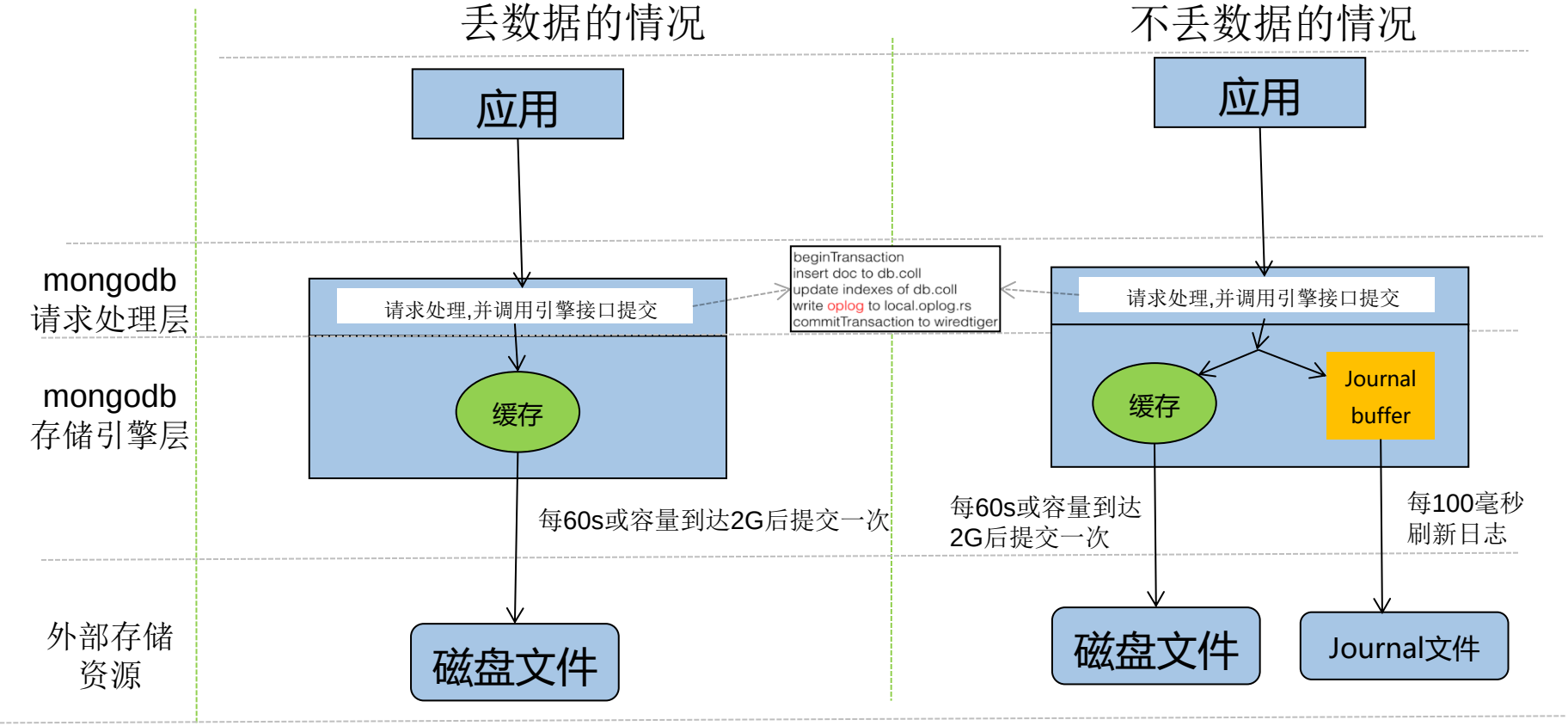
目录 CONTENTS

01 wiredTiger

MongoDB从3.0开始引入可插拔存储引擎的概念。目前主要有MMAPV1、WiredTiger存储引擎可供选择。在3.2版本之前MMAPV1是默认的存储引擎,其采用linux操作系统内存映射技术,但一直饱受诟病；3.4以上版本默认的存储引擎是wiredTiger,相对于MMAPV1其有如下优势:

- ✓ 读写操作性能更好,WiredTiger能更好的发挥多核系统的处理能力；
- ✓ MMAPV1引擎使用表级锁,当某个单表上有并发的操作,吞吐将受到限制。WiredTiger使用文档级锁,由此带来并发及吞吐的提高
- ✓ 相比MMAPV1存储索引时WiredTiger使用前缀压缩,更节省对内存空间的损耗；
- ✓ 提供压缩算法,可以大大降低对硬盘资源的消耗,节省约60%以上的硬盘资源；

02 mongodb数据会丢失？你需要了解WT写入的原理

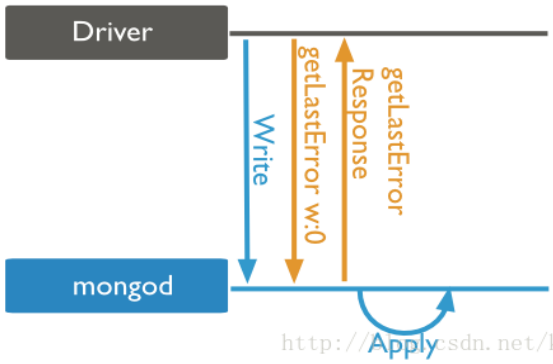


- **Journaling**类似于关系数据库中的事务日志。Journaling能够使MongoDB数据库由于意外故障后快速恢复。MongoDB2.4版本后默认开启了Journaling日志功能,mongod实例每次启动时都会检查journal日志文件看是否需要恢复。由于提交journal日志会产生写入阻塞,所以它对写入的操作有性能影响,但对于读没有影响。在生产环境中开启Journaling是很有必要的。

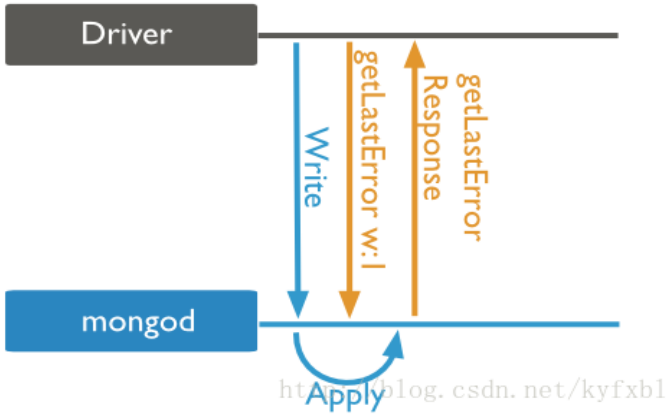
02 写策略解析

- com.mongodb.WriteConcern

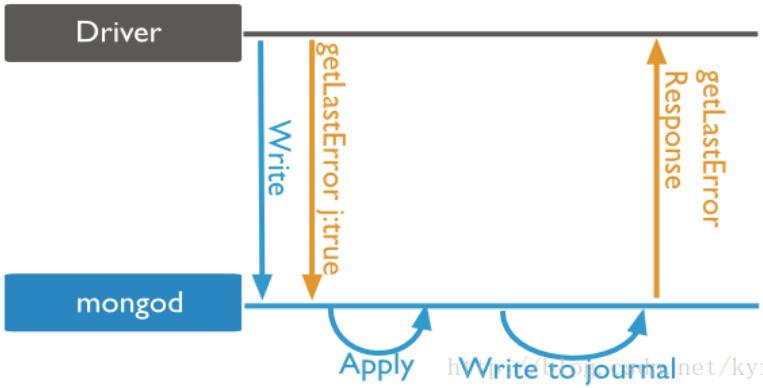
- {w: 0} Unacknowledged



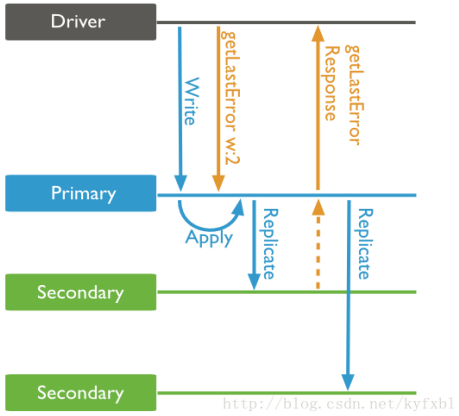
- write concern:1（acknowledged）



- concern:1 & journal:true（Jounaled）



- concern:2（Replica Acknowledged）



02 配置文件

压缩算法 Tips:
性能: none > snappy >zlib
压缩比:zlib > snappy > none

```
storage:
    journal:
        enabled: true
    dbPath: /data/zhou/mongo1/
    ##是否一个库一个文件夹
    directoryPerDB: true
    ##数据引擎
    engine: wiredTiger
    ##WT引擎配置
    WiredTiger:
        engineConfig:
            ##WT最大使用cache（根据服务器实际情况调节）
            cacheSizeGB: 1
            ##是否将索引也按数据库名单独存储
            directoryForIndexes: true
            journalCompressor:none （默认snappy）
        ##表压缩配置
        collectionConfig:
            blockCompressor: zlib (默认snappy,还可选none、zlib)
        ##索引配置
        indexConfig:
            prefixCompression: true
```

目录 CONTENTS

01 索引命令概要与类型

- 索引通常能够极大的提高查询的效率,如果没有索引,MongoDB在读取数据时必须扫描集合中的每个文件并选取那些符合查询条件的记录。索引主要用于排序和检索

单键索引

在某一个特定的属性上建立索引,例如:db.users. createIndex({age:-1});

- ✓ mongoDB在ID上建立了唯一的单键索引,所以经常会使用id来进行查询；
- ✓ 在索引字段上进行精确匹配、排序以及范围查找都会使用此索引；

复合索引

在多个特定的属性上建立索引,例如:db.users. createIndex({username:1,age:-1,country:1});

- ✓ 复合索引键的排序顺序,可以确定该索引是否可以支持排序操作；
- ✓ 在索引字段上进行精确匹配、排序以及范围查找都会使用此索引,但与索引的顺序有关；
- ✓ 为了性能考虑,应删除存在与第一个键相同的单键索引

多键索引

在数组的属性上建立索引,例如:db.users. createIndex({favorites.city:1});针对这个数组的任意值的查询都会定位到这个文档,既多个索引入口或者键值引用同一个文档

02 索引类型

◆ 哈希索引

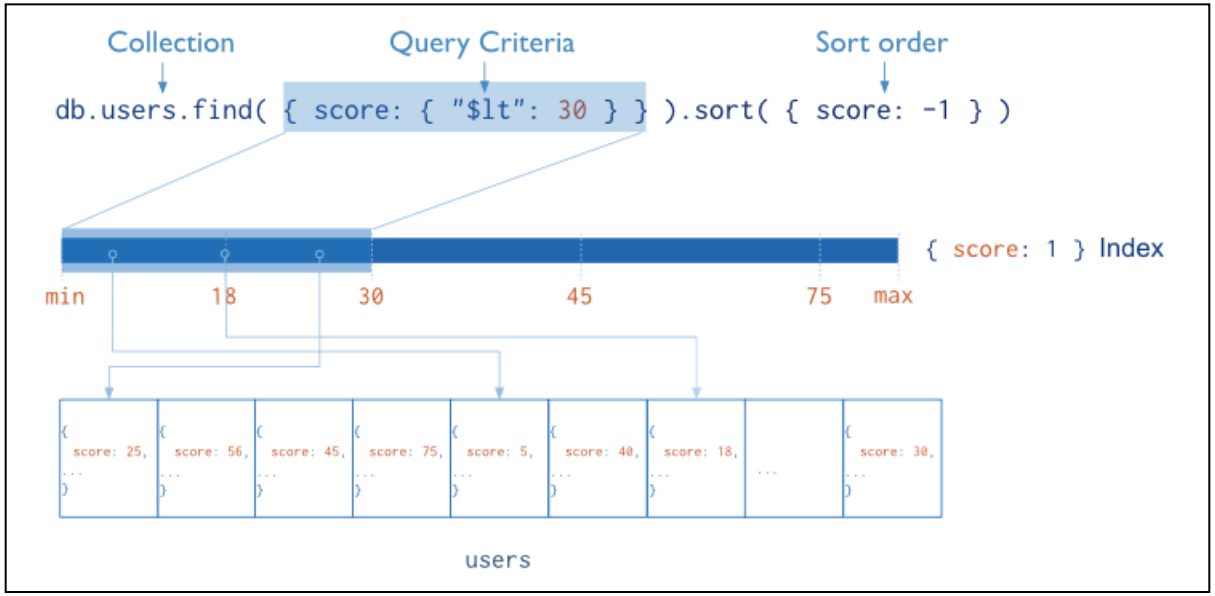
不同于传统的B-树索引,哈希索引使用hash函数来创建索引。

例如:db.users. createIndex({username : 'hashed'});

- ✓ 在索引字段上进行精确匹配,但不支持范围查询,不支持多键hash；
- ✓ Hash索引上的入口是均匀分布的,在分片集合中非常有用；

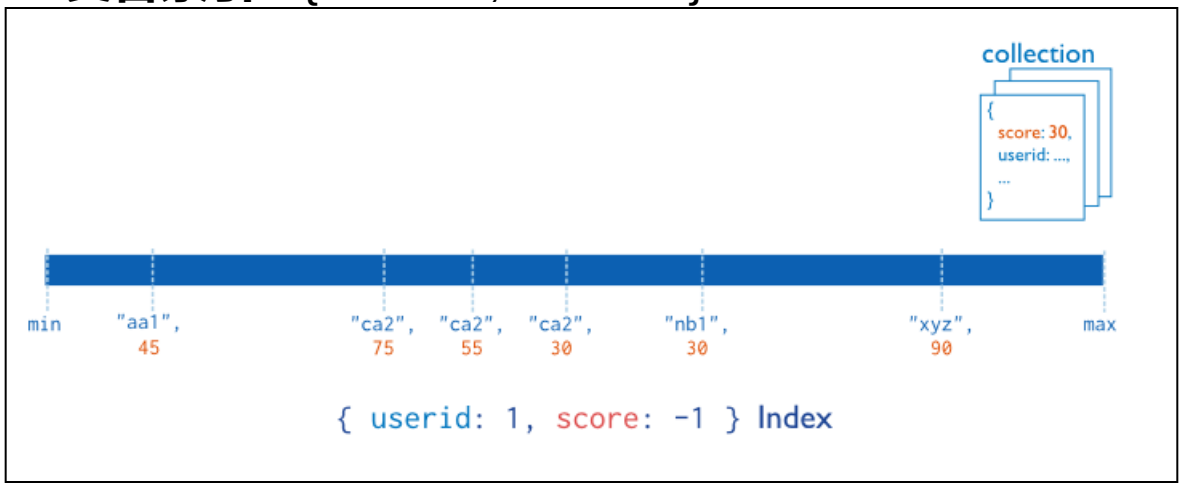
02 索引图示

◆ 单键索引



为什么mongodb很耗内存？
读和写都是基于内存的

◆ 复合索引 {userid:1, score:-1}



03 索引语法

- MongoDB使用 ensureIndex() 方法来创建索引,ensureIndex()方法基本语法格式如下所示:

  db.collection.createIndex(keys, options)

  - ✓ 语法中 Key 值为要创建的索引字段,1为指定按升序创建索引,如果你想按降序来创建索引指定为-1,也可以指定为hashed（哈希索引）。
  - ✓ 语法中options为索引的属性,属性说明见下表；

04 索引属性

属性名	类型	说明
background	boolean	是否后台构建索引,在生产环境中,如果数据量太大,构建索引可能会消耗很长时间,为了不影响业务,可以加上此参数,后台运行同时还会为其他读写操作让路
unique	boolean	是否为唯一索引
name	string	索引名字
sparse	boolean	是否为稀疏索引,索引仅引用具有指定字段的文档。

05 索引管理实战

创建索引

- ✓ 单键唯一索引:db.users. createIndex({username :1},{unique:true});
- ✓ 单键唯一稀疏索引:db.users. createIndex({username :1},{unique:true,sparse:true});
- ✓ 复合唯一稀疏索引:db.users. createIndex({username:1,age:-1},{unique:true,sparse:true});
- ✓ 创建哈希索引并后台运行:db.users. createIndex({username :'hashed'},{background:true});

删除索引

- ✓ 根据索引名字删除某一个指定索引:db.users.dropIndex("username_1");
- ✓ 删除某集合上所有索引:db.users.dropIndexs();
- ✓ 重建某集合上所有索引:db.users.reIndex();
- ✓ 查询集合上所有索引:db.users.getIndexes();

06 查询优化技巧 第一步

找出慢速查询

1. **开启内置的查询分析器,记录读写操作效率**:

   db.setProfilingLevel(n,{m}),n的取值可选0,1,2；

   - ✓ 0是默认值表示不记录；
   - ✓ 1表示记录慢速操作,如果值为1,m必须赋值单位为ms,用于定义慢速查询时间的阈值；
   - ✓ 2表示记录所有的读写操作；

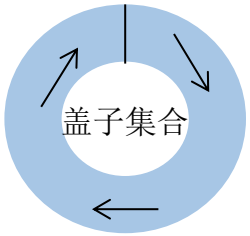
   例如:db.setProfilingLevel(1,300)

2. **查询监控结果**

监控结果保存在一个特殊的盖子集合system.profile里,这个集合分配了128kb的空间,要确保监控分析数据不会消耗太多的系统性资源；盖子集合维护了自然的插入顺序,可以使用$natural操作符进行排序,如:db.system.profile.find().sort({'$natural':-1}).limit(5)

盖子集合 Tips:

1. 大小或者数量固定；
2. 不能做update和delete操作；
3. 容量满了以后,按照时间顺序,新文档会覆盖旧文档



07 查询优化技巧 第二步

分析慢速查询

找出慢速查询的原因比较棘手,原因可能有多个:应用程序设计不合理、不正确的数据模型、硬件配置问题,缺少索引等；接下来对于缺少索引的情况进行分析:

使用explain分析慢速查询

例如:db.orders.find({'price':{'$lt':2000}}).explain('executionStats')

explain的入参可选值为:

- ✓ "queryPlanner" 是默认值,表示仅仅展示执行计划信息；
- ✓ "executionStats" 表示展示执行计划信息同时展示被选中的执行计划的执行情况信息；
- ✓ "allPlansExecution" 表示展示执行计划信息,并展示被选中的执行计划的执行情况信息,还展示备选的执行计划的执行情况信息；

08 查询优化技巧 第三步

- **解读explain结果**

queryPlanner（执行计划描述）

winningPlan（被选中的执行计划）

stage（可选项:COLLSCAN 没有走索引；IXSCAN使用了索引）

rejectedPlans(候选的执行计划)

executionStats(执行情况描述)

nReturned （返回的文档个数）

executionTimeMillis（执行时间ms）

totalKeysExamined （检查的索引键值个数）

totalDocsExamined （检查的文档个数）

优化目标 **Tips:**

1. 根据需求建立索引
2. 每个查询都要使用索引以提高查询效率, winningPlan. stage 必须为IXSCAN ；
3. 追求totalDocsExamined = nReturned

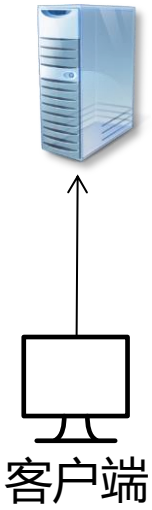
09 关于索引的建议

1. 索引很有用,但是它也是有成本的——它占内存,让写入变慢；
2. mongoDB通常在一次查询里使用一个索引,所以多个字段的查询或者排序需要复合索引才能更加高效；
3. 复合索引的顺序非常重要
4. 在生成环境构建索引往往开销很大,时间也不可以接受,在数据量庞大之前尽量进行查询优化和构建索引；
5. 避免昂贵的查询,使用查询分析器记录那些开销很大的查询便于问题排查；
6. 通过减少扫描文档数量来优化查询,使用explai对开销大的查询进行分析并优化；
7. 索引是用来查询小范围数据的，不适合使用索引的情况：
   - 每次查询都需要返回大部分数据的文档，避免使用索引
   - 写比读多

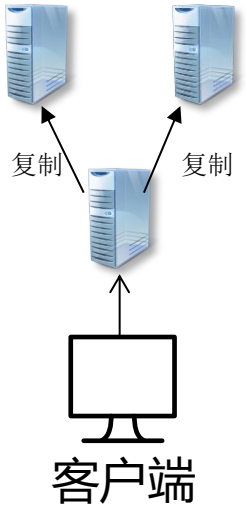
C目录 ONTENTS

01 部署模型概述

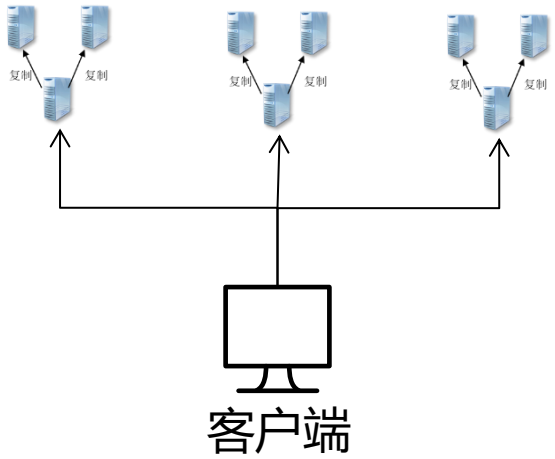
单机部署



可复制集



分片集群



系统可扩展性、吞吐量、健壮性和数据安全性依次递增

目录 CONTENTS

01 可复制集

可复制集是跨多个MongDB服务器（节点）分布和维护数据的方法。mongoDB可以把数据从一个节点复制到其他节点并在修改时进行同步,集群中的节点配置为自动同步数据；旧方法叫做主从复制,mongoDB 3.0以后推荐使用可复制集；

为什么要用可复制集？它有什么重要性？

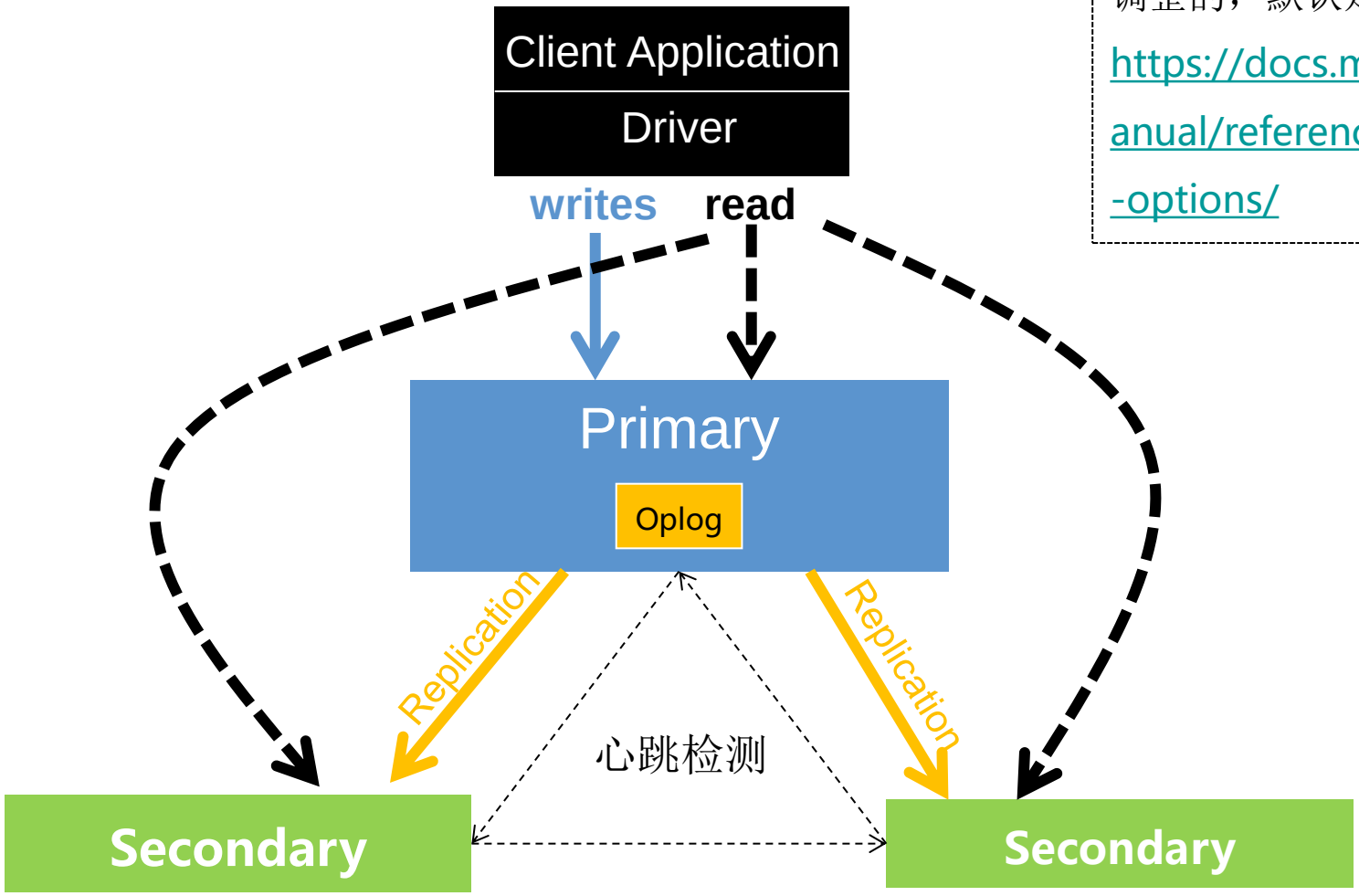
- ✓ 避免数据丢失,保障数据安全,提高系统安全性；
  （最少3节点,最大50节点）
- ✓ 自动化灾备机制,主节点宕机后通过选举产生新主机；提高系统健壮性；
  （7个选举节点上限）
- ✓ 读写分离,负载均衡,提高系统性能；
- ✓ 生产环境推荐的部署模式；

02 可复制集架构以及原理

Tips:

oplog是盖子集合，大小是可以调整的，默认是所在硬盘5%；

https://docs.mongodb.com/manual/reference/configuration-options/



09 可复制集的搭建过程

1. 安装好3个以上节点的mongoDB；
2. 配置mongodb.conf,增加跟复制相关的配置如下:

```
replication:
  replSetName: configRS  //集群名称
  oplogSizeMB: 50 //oplog集合大小
```

3. 在primary节点上运行可复制集的初始化命令,初始化可复制集,命令如下:

```
//复制集初始化,在主节点上执行,ip禁止使用localhost
rs.initiate({
      _id: "configRS",
      version: 1,
      members: [{ _id: 0, host : "192.168.1.142:27017" }]});
rs.add("192.168.1.142:27018");//有几个节点就执行几次方法
rs.add("192.168.1.142:27019");//有几个节点就执行几次方法
```

4. 在每个节点运行rs.status()或isMaster()命令查看复制集状态；
5. 测试数据复制集效果；
6. 测试故障失效转移效果；

打开从节点可读Tips:

只能在主节点查询数据,但如果想在副节点查询到数据需运行

rs.slaveOk();

09 使用正确的姿势连接复制集

MongoDB复制集里Primary节点是不固定的,不固定的,不固定的!

所以生产环境千万不要直连Primary,千万不要直连Primary,千万不要直连Primary!

重要的事情说3遍!

- **java原生驱动开发**

```
List<ServerAddress> asList = Arrays.asList(
                                new ServerAddress("192.168.1.142", 27018),
                                new ServerAddress("192.168.1.142", 27017),
                                new ServerAddress("192.168.1.142", 27019));
client = new MongoClient(asList);
```

- **Spring配置开发**

```
<mongo:mongo-client
replica-set="192.168.1.142:27017,192.168.1.142:27018,192.168.1.142:27017">
</mongo:mongo-client>
```

配置Tips:

1. 关注Write Concern参数的设置,默认值1可以满足大多数场景的需求。W值大于1可以提高数据的可靠持久化,但会降低写性能。
2. 在options里添加readPreference=secondaryPreferred即可实现读写分离,读请求优先到Secondary节点,从而实现读写分离的功能

目录 CONTENTS

01 分片集群（why）

分片是把大型数据集进行分区成更小的可管理的片,这些数据片分散到不同的mongoDB节点,这些节点组成了分片集群。

为什么要用分片集群？

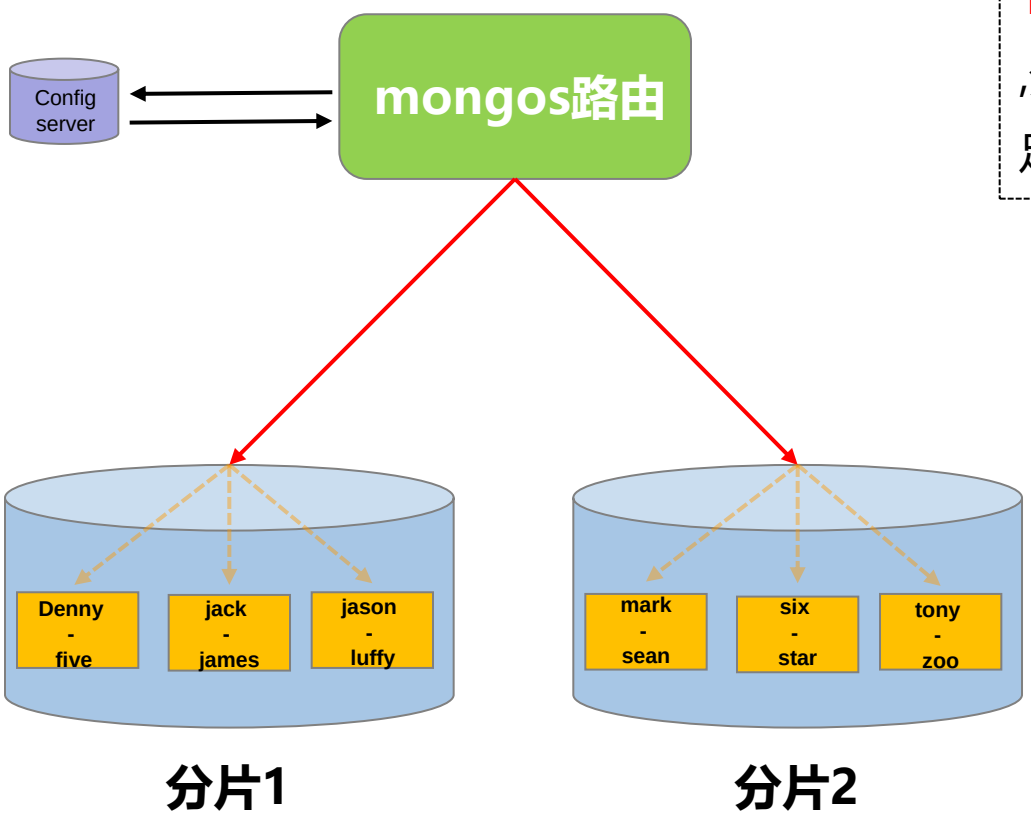
- ✓ 数据海量增长,需要更大的读写吞吐量 → **存储分布式**
- ✓ 单台服务器内存、cpu等资源是有瓶颈的 → **负载分布式**

Tips:分片集群是个双刃剑,在提高系统可扩展性和性能的同时,增大了系统的复杂性,所以在实施之前请确定是必须的。

02 分片到底是分的什么鬼？（what）

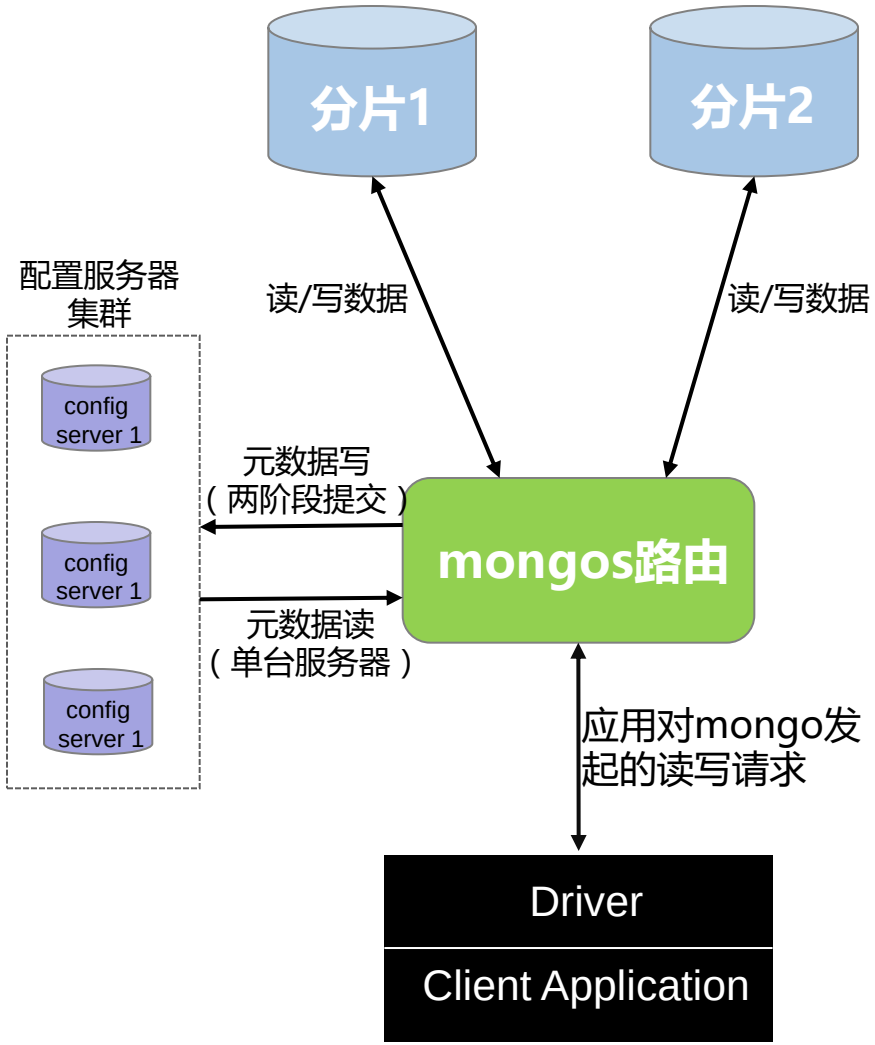
数据库？集合？文档？mongoDB分片集群推荐的模式是:分片集合,它是一种基于分片键的逻辑对文档进行分组,分片键的选择对分片非常重要,分片键一旦确定,mongoDB对数据的分片对应用是透明的；

- 以testOrder为例，使用useCode作为分片键



Tips:随着数据量的的增大,分片会分割和迁移,以满足数据的均匀分布。

02 分片集群架构图与组件（what）



- **分片:**在集群中唯一存储数据的位置,可以是单个mongo服务器,也可以是可复制集,每个分区上存储部分数据；生产环境推荐使用可复制集
- **mongos路由:**由于分片之存储部分数据,需要mongos路由将读写操作路由到对应的分区上；mongos提供了单点连接集群的方式,轻量级、非持久化所以通常mongos和应用部署在同一台服务器上；
- **配置服务器:**存储集群的元数据,元数据包括:数据库、集合、分片的范围位置以及跨片数据分割和迁移的日志信息；mongos启动时会从配置服务器读取元数据信息在内存中；配置服务器最低

04 分片搭建过程

？？

04 分片注意点与建议

分片注意点:

- **热点 :**某些分片键会导致所有的读或者写请求都操作在单个数据块或者分片上,导致单个分片服务器严重不堪重负。自增长的分片键容易导致写热点问题；
- **不可分割数据块:**过于粗粒度的分片键可能导致许多文档使用相同的分片键,这意味着这些文档不能被分割为多个数据块,限制了mongoDB均匀分布数据的能力；
- **查询障碍:**分片键与查询没有关联,造成糟糕的查询性能。

建议:

- 不要使用自增长的字段作为分片键,避免热点问题；
- 不能使用粗粒度的分片键,避免数据块无法分割；
- 不能使用完全随机的分片键值,造成查询性能低下；
- 使用与常用查询相关的字段作为分片键,而且包含唯一字段（如业务主键,id等）；
- 索引对于分区同样重要,每个分片集合上要有同样的索引,分片键默认成为索引；分片集合只允许在id和分片键上创建唯一索引；

目录 CONTENTS

01 最佳实践

1. 尽量选取稳定新版本64位的mongodb；
2. 数据模式设计；提倡单文档设计，将关联关系作为内嵌文档或者内嵌数组；当关联数据量较大时，考虑通过表关联实现，dbref或者自定义实现关联；
3. 避免使用skip跳过大量数据；（1）通过查询条件尽量缩小缩小数据范围；（2）利用上一次的结果作为条件来查询下一页的结果；
4. 避免单独使用不适用索引的查询符（$ne、$nin、$where等）
5. 根据业务场景，选择合适的写入策略，在数据安全和性能之间找到平衡点；
6. 索引建议很重要；
7. 生产环境中建议打开profile，便于优化系统性能；
8. 生产环境中建议打开auth模式，保障系统安全；
9. 不要将mongoDB和其他服务部署在同一台机器上（mongodb占用的最大内存是可以配置的）；
10. 单机一定要开启journal日志，数据量不太大的业务场景中，推荐多机器使用副本集，并开启读写分离；
11. 分片键的注意事项