推荐教程：MongoDB权威指南

# MongoDB

C++非关系型数据库

下载解压，第一次运行需要建立存放数据的目录，mongod默认写入/data/db/

sudo mkdir -p /data/db

sudo chown ‘id –u’ /data/db

启动时用--dbpath指明数据目录。

官方包不包含mongod控制脚本。

mongod --config /etc/mongod.conf

## 入门

mongo脚本是标准MongoDB版本的一部分，纯JS编写，提供Mongo所有标准方法。

mongod启动后，可以用./bin/mongo脚本链接Mongo，默认链接localhost的27017端口，默认使用test数据库。Mongo提示符后输入以下两个命令：

db.test.save( { a: 1 } ) 向test集合中插入述

db.test.find()查询test数据库的所有数据

db：显示当前数据库名称

show dbs：显示所有数据库 =show databases

use mydb：切换到mydb数据库，这时输show dbs，不会出现mydb。因此如果没有插入数据，mongo不会新建数据库。

MongoDB将在第一次使用集合时显式生成集合，你不需要在插入数据前新建集合。MongoDB采用动态策略，因此不需要指明集合结构。

新建j、k变量

j = { name : "mongo" }

k = { x : 3 }

将其加入testData集合中，此时mongod会新建mydb数据库和testData集合。

db.testData.insert( j )

db.testData.insert( k )

show collections：显示所有集合，mongo会返回当前数据库（mydb）的所有集合，其中包括testData和所有数据库都包含的system.indexes。

db.testData.find()：通过在testData上调用find方法确认testData存在。返回

{ "\_id" : ObjectId("4c2209f9f3924d31102bd84a"), "name" : "mongo" } { "\_id" : ObjectId("4c2209fef3924d31102bd84b"), "x" : 3 }

所有mongoDB都包含唯一的\_id字段。

插入25条数据

for (var i = 1; i <= 25; i++) db.testData.insert( { x : i } )

db.testData.find() 查看结果

当查询mongo时会返回一个cursor，其包含查询结果。Mongo通过遍历它显示所有结果。首先迭代20次cursor显示20条记录，等待shell输入it再迭代下20个结果。

也可以写个insertData方法，包含三个参数，数据库名，集合名，生成多少个数据。将其保存在.mongorc.js文件中，mongo每次开启对话时都会加载该文件。

function insertData(dbName, colName, num) {

var col = db.getSiblingDB(dbName).getCollection(colName);

for (i = 0; i < num; i++) {

col.insert({x:i});

}

print(col.count());

}

insertData("test", "testData", 4) 调用该方法

查询testData，将返回的cursor赋给c

var c = db.testData.find()

通过while遍历c，打印所有结果，如果c包含下一个文本，则hasNext返回true；printjson以JSON格式渲染文本

while ( c.hasNext() ) printjson( c.next() )

再次将cursor赋给c

var c = db.testData.find()

打印第四个结果，当以数组索引方式访问cursor时，首先调用cursor.toArray()，将所有文本载入内存，再映射索引到数据。该操作将遍历curosr并废弃它。对过大数据集，可能会溢出。

printjson( c [ 4 ] )

查询testData中x包含18

db.testData.find( { x : 18 } )

findOne同find的参数一致，但返回文本而不是cursor

db.testData.findOne()

为提高效率，通过在cursor上调用limit限制查询结果的个数

db.testData.find().limit(3)

## 驱动

链接MongoDB数据库的标准URI格式：

mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:portN]]][/[database][?options]]

mongodb://：必须前缀，指明采用标准链接格式。

Username:password@：可选，如果设置，则链接到mongod后，客户端会尝试用该用户名密码登陆指定数据库。

Host1：URI唯一需要指定的部分。指明链接的服务器地址。可以是hostname，ip，或Unix域名。

:port1：可选，默认27017端口。

HostX：可选，你可指定多个host。

:portX：可选，指定多个port。

/database：可选。如果URI中设置了username:password@，则为认证数据库名。如果没有设置而又设置了username:password@，则默认向admin数据库认证。

?option：可选参数。以name=value的格式。如果没有设置database，则必须在hostN后面设置一个/。然后再填? option。

常用例子：

通过以下mongod主机连接一个名为test的replica set

db1.example.net on port 27017

db2.example.net on port 2500

连接字符串应为

mongodb://db1.example.net,db2.example.net:2500/?replicaSet=test

连接本地默认端口

mongodb://localhost

以sysop用户和moon密码连接admin数据库

mongodb://sysop:moon@localhost

以sysop用户和moon密码连接records数据库

mongodb://sysop:moon@localhost/records

连接UNIX域名socket

mongodb:///tmp/mongodb-27017.sock

连接到两个成员的replica set，一个是db1.example.net，另一个是db2.example.com

mongodb://db1.example.net,db2.example.com

连接到三个成员的replica set，都运行在localhost，分别在27017, 27018, 27019端口

mongodb://localhost,localhost:27018,localhost:27019

连接到三个成员的replica set，分配reads到secondary。

mongodb://example1.com,example2.com,example3.com/?readPreference=secondary

如：设置了replicaSet和connectTimeoutMS两个参数。分隔符可以是&，还可以是;

mongodb://db1.example.net,db2.example.net:2500/?replicaSet=test&connectTimeoutMS=300000

replicaSet： replica set名。当连接到一个replica set，至少要提高两个mongod实例，如果只提供一个mongod实例，且没有replicaSet，则客户端建立standalone链接。

ssl：true：以SSL初始化链接，默认为false。所有驱动都不支持。

connectTimeoutMS：连接超时，毫秒。默认永不，不同驱动可能不同。

socketTimeMS：发送或接收socket超时，毫秒。默认永不，不同驱动可能不同。

maxPoolSize：连接池的最大链接数。默认100。

minPoolSize：最小链接数。默认0。所有驱动都不支持。

maxIdleTimeMS：

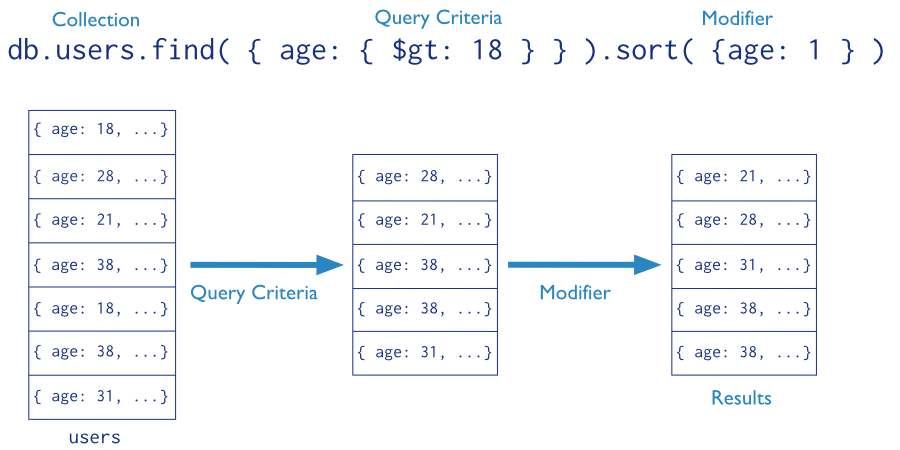
## CRUD

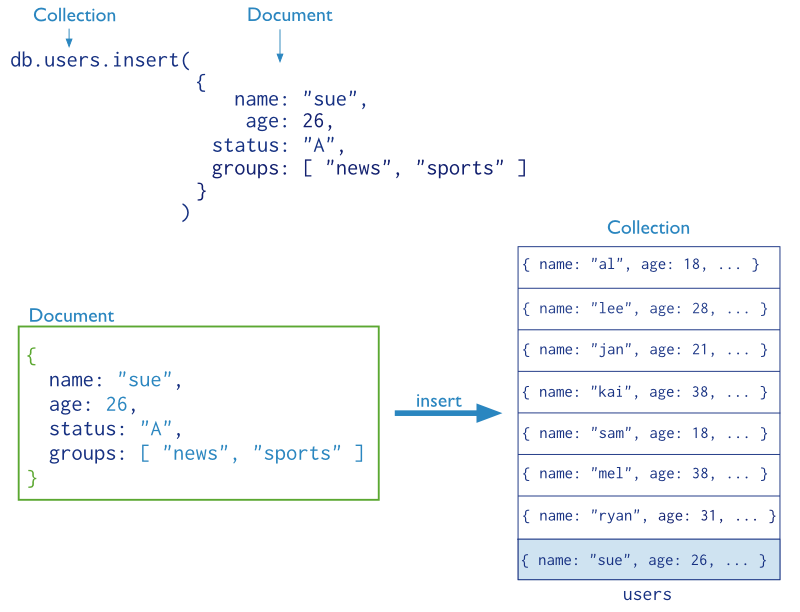
MongoDB以document格式存储数据，类似JSON的key-value对，称为BSON，即JSON添加额外信息。



所有document存为collection。Collection由一组相关的document加上共享索引组成。类似于关系型数据库中的表。

查询：针对特定collection，只有符合要求的document才返回。





索引：增加查询和修改的效率。MongoDB支持二次索引。

读偏好：对replica set和共享集群，应用指明读偏好，决定客户端如何读集合。

写关注：应用通过写关注控制写行为，对部署replica set非常有用，写关注可要求MongoDB提供写操作成功保证。

聚合：MongoDB能返回查询到的document数量（count）。

### 读操作

db.collection.find()接收查询条件和投影，返回cursor。投影限制MongoDB通过网络返回客户端的数据量。可选的增加限制、跳过、排序条件。



该操作是在users的collection中查询age>18，大于须用query selection operator的$gt表示。返回最多5条document，只返回\_id，name，address字段。

相应的SQL语句为：

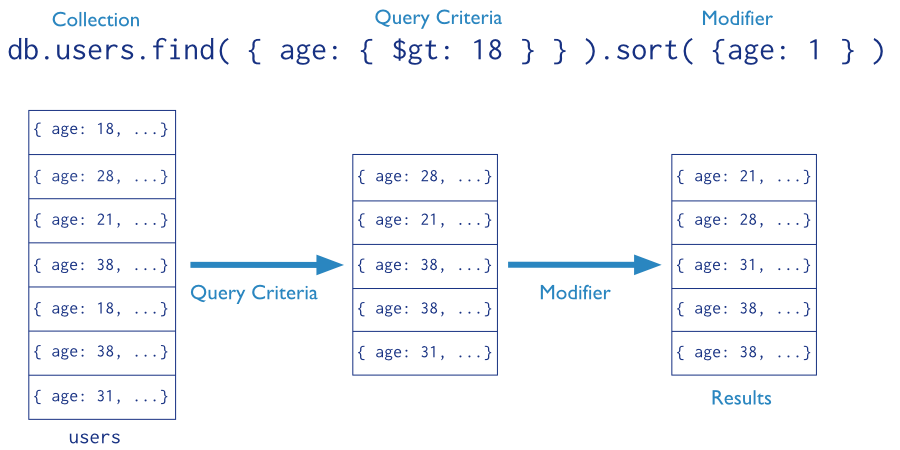


查询只针对单一集合。

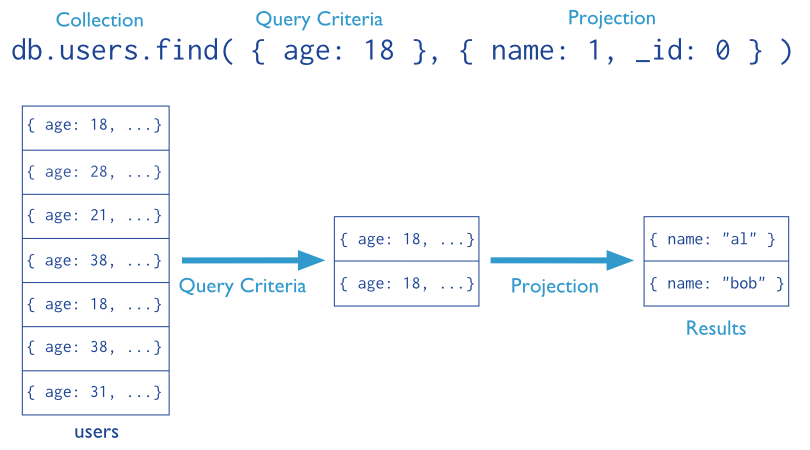
可增加limits，skips和sort。

db.collection.findOne()返回一条document。

排序例子如下：返回document的顺序不确定，除非指定sort。



投影例子如下：



查询user\_id小于42，去掉history字段。

db.records.find( { "user\_id": { $lt: 42} }, { history: 0} )

查询user\_id小于42，返回document中包含\_id，name，email字段

db.records.find( { "user\_id": { $lt: 42} }, { "name": 1, "email": 1} )

查询user\_id小于42，只返回name和email字段。

db.records.find( { "user\_id": { $lt: 42} }, { "\_id": 0, "name": 1 , "email": 1 } )

\_id字段除了显示排除否则查询时都返回。

字段包含数组，则采用$elemMatch，$slice，$投影操作符。

默认服务器会在10分钟没操作或废弃cursor后自动关闭cursor。通过设置noTimeout修改该操作，那么你需要手动关闭或废弃cursor。

var myCursor = db.inventory.find().addOption(DBQuery.Option.noTimeout);

adoption还可添加其他flag。

Cursor并不是孤立的。对document的写操作会反映在cursor上。

MongoDB批量返回查询结果，批量大小不超过maximum BSON document size。第一次批处理返回101个documents，或大小超过1megabyte的documents。接下来返回的批处理大小为4megabytes。

查询包含sort但没有index时，服务器会加载所有document到内存，然后执行sort，返回所有document。

### Cursor

通过遍历cursor访问返回的document，如果没有设置，默认遍历前20个，直到输入it再遍历20个，依次循环。

遍历cursor完批处理后，如果还有结果，调用cursor.next()执行getmore操作，获取下一批结果。ObjsLeftInBatch方法查看还剩多少结果。

var myCursor = db.inventory.find(); var myFirstDocument = myCursor.hasNext() ? myCursor.next() : null; myCursor.objsLeftInBatch();

通过cursorInfo获得cursor信息：

所有打开的cursor数量；当前客户端使用的cursor大小；server重启后共超时的cursor数量。

db.runCommand( { cursorInfo: 1 } )

返回结果如下：

{ "totalOpen" : <number>,

"clientCursors\_size" : <number>,

"timedOut" : <number>,

"ok" : 1

}

### 查询优化

生成索引有利于减少查询范围

例如有如下查询：

var typeValue = <someUserInput>;

db.inventory.find( { type: typeValue } );

为提高查询效率，通过db.collection.ensureIndex()在inventory集合上建立type的索引，如：db.inventory.ensureIndex( { type: 1 } )

一些查询操作如$nin不等于和$ne等于，通常要遍历所有数据，因此索引用处不大。

用正则表达式查询不能使用索引。

覆盖查询须满足2个条件：

1. 查询的字段都包含在索引中；
2. 投影与查询在同一个索引中。

例如：

inventory在type和item字段上有索引{type:1, item:1}

该查询覆盖了所有索引字段，只返回item，item在索引中

db.inventory.find( { type: "food", item:/^c/ },

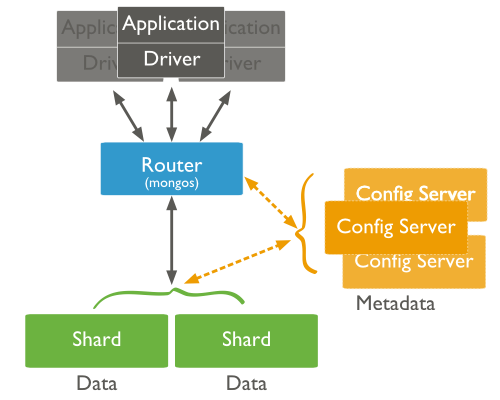
{ item: 1, \_id: 0 } )

该查询覆盖了所有索引字段，但返回\_id，\_id不在索引中

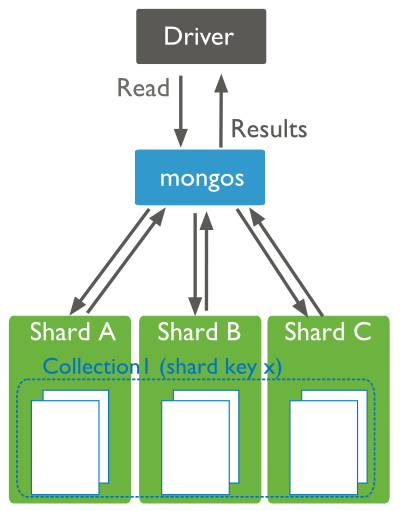
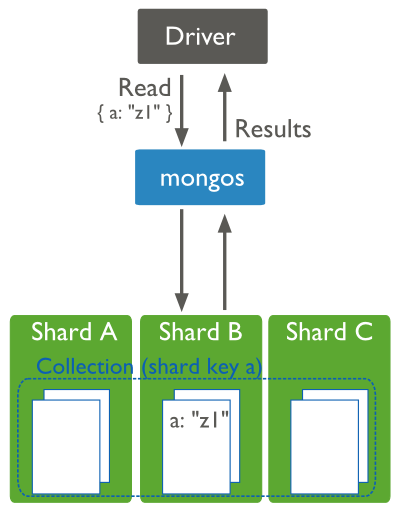
db.inventory.find( { type: "food", item:/^c/ },

{ item: 1 } )

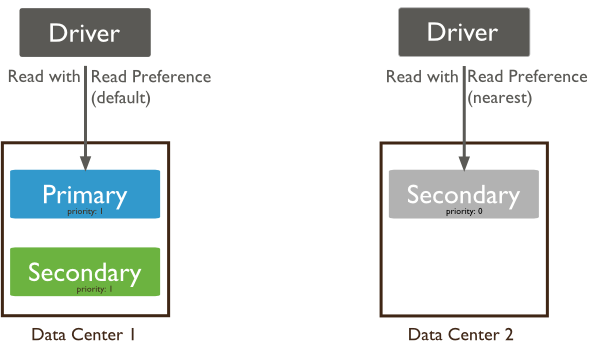
### 分布式读



在共享集群中，collection分布在多个节点，查询包含shard key，可以直接到指定位置获取数据，提高查询效率，否则就要到所有节点上遍历。



读偏好默认读取主本，可通过设置read preference mode，读取最近副本。必须考虑到副本和主本的状态不同步问题。



### 写操作

写操作：包含create，update和delete。都是原子的。

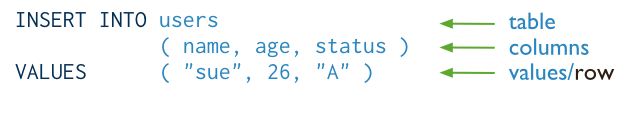
db.collection.insert()新增



在users集合新增document，字段为name，age，status和\_id，如果没有设\_id，则mongodb自动添加唯一\_id。如果设\_id，则必须保证其唯一性，否则报重复key exception。

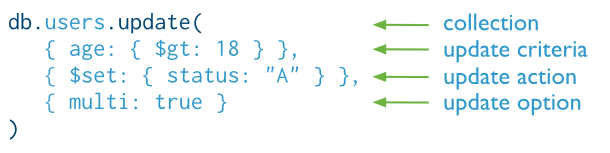
upsert是指更新document，如果不存在则新增。db.collection.update()和db.collection.save()都可设置upsert:true。

相当于sql的：

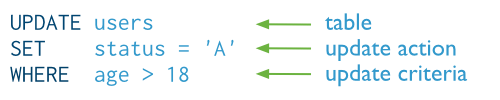


db.collection.update()跟find的参数差不多。默认只更改一条document，Multi设可更改多条document。

设所有age>18的status=A，

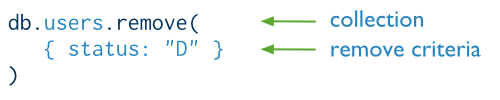


相当于sql的：

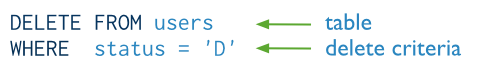


1. 默认更新一条document，设multi=true则更新所有document；
2. 更新导致document size超过分配空间，更新操作重分配document空间，可能还会重排序；

db.collection.remove()删除所有status=D的document。



相当于sql的



### 事务管理

保存2个账户信息

db.accounts.save({name: "A", balance: 1000, pendingTransactions: []})

db.accounts.save({name: "B", balance: 1000, pendingTransactions: []})

初始化事务

db.transactions.save({source: "A", destination: "B", value: 100, state: "initial"})

开始执行事务pending

t = db.transactions.findOne({state: "initial"})

db.transactions.update({\_id: t.\_id}, {$set: {state: "pending"}})

事务应用到账户中

db.accounts.update({name: t.source, pendingTransactions: {$ne: t.\_id}}, {$inc: {balance: -t.value}, $push: {pendingTransactions: t.\_id}})

db.accounts.update({name: t.destination, pendingTransactions: {$ne: t.\_id}}, {$inc: {balance: t.value}, $push: {pendingTransactions: t.\_id}})

db.accounts.find()

标记事务提交committed

db.transactions.update({\_id: t.\_id}, {$set: {state: "committed"}})

删除pending事务

db.accounts.update({name: t.source}, {$pull: {pendingTransactions: t.\_id}})

db.accounts.update({name: t.destination}, {$pull: {pendingTransactions: t.\_id}})

事务标记完成

db.transactions.update({\_id: t.\_id}, {$set: {state: "done"}})

两种回滚方法：

1. 如果已经提交事务，则新建一个事务反向操作一次；
2. 如果新建事务，但没提交，则采用以下步骤：

db.transactions.update({\_id: t.\_id}, {$set: {state: "canceling"}})

db.accounts.update({name: t.source, pendingTransactions: t.\_id}, {$inc: {balance: t.value}, $pull: {pendingTransactions: t.\_id}})

db.accounts.update({name: t.destination, pendingTransactions: t.\_id}, {$inc: {balance: -t.value}, $pull: {pendingTransactions: t.\_id}})

db.transactions.update({\_id: t.\_id}, {$set: {state: "canceled"}})

查询

db.collection.find({查询条件}，{投影})

limits，skips，sort，返回值cursor

db.collection.findOne()

光标

cursor.hasNext(), next(), forEach(),toArray()

cursor.explain(), hint().explain()对比

索引

db.collection.ensureIndex({type:1})

插入

db.collection.insert()

db.collection.update({更新条件},{更新操作},{更新选项}) upsert:true, multi:true

db.collection.save()

删除

db.collection.remove()

db.collection.drop()

$gt：大于

$lt：小于

# Spring MongoDB

## 配置

### pom.xml

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-mongodb</artifactId>

<version>1.1.1.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-mongodb-log4j</artifactId>

<version>1.2.1.RELEASE</version>

</dependency>

### mt.oa.mongodb.xml

<!-- mongo的工厂，通过它来取得mongo实例,dbname为mongodb的数据库名，没有的话会自动创建 -->

<mongo:db-factory dbname="mt\_oa\_pf"

host="${mongodb.host}"

port="${mongodb.port}"/>

<!-- mongodb的主要操作对象，所有对mongodb的增删改查的操作都是通过它完成 -->

<bean id="mongoTemplate" class="org.springframework.data.mongodb.core.MongoTemplate">

<constructor-arg ref="mongoDbFactory"/>

</bean>

MongoOperations mongoOperation = (MongoOperations)ctx.getBean( " mongoTemplate " );

用@Autowired自动注入

## 使用

MongoHelper定义了所有集合名称。

**public** **static** **final** String *COLL\_MDS* = "mt\_oa\_metadta";

**public** **static** **final** String *COLL\_OPLOG* = "mt\_oa\_oplog";

**public** **static** **final** String *COLL\_MLOG* = "mt\_oa\_mlog";

**public** **static** **final** String *COLL\_UILOG* = "mt\_oa\_uilog";

**public** **static** **final** String *COLL\_DS\_MODEL* = "mt\_oa\_ds\_model";

**public** **static** **final** String *COLL\_DATASET* = "mt\_oa\_dataset";

**public** **static** **final** String *COLL\_BS\_MODEL* = "mt\_oa\_bs\_model";

**public** **static** **final** String *COLL\_BS\_CODE* = "mt\_oa\_bs\_code";

**public** **static** **final** String *COLL\_POD\_MODEL* = "mt\_oa\_pod";

Criteria：用于生成查询条件。 Criteria.where(“”).is(“”).and(“”).or(“”)

Query ：利用Criteria生成查询，Query.query(c)

List<T> MongoOperations.find(Query query, Class<T> entityClass, String collectionName)

void MongoOperations.remove(Query query, String collectionName)

void MongoOperations.save(Object objectToSave, String collectionName)