# MongoDB开发使用手册

## 一．基础部分

### MongDB简介

#### NOSQL历史和产生原因

原因: 互联网用户数的增长和用户参与互联网习惯的改变

1. 初始的静态内容网站，提供中心化的内容服务，

特点: 中心化，用户阅读内容

系统：Apache

1. 动态网页内容，电子商务和论坛网站出现。

特点: 用户访问动态内容，并提供少量内容

系统：Apache+Mysql+Php, IIS+ASP+SQLSERVER, IIS+ASP.net+SQLSERVER, TOMCAT+JSP+ORACLE

3. 博客出现: 去中心化网站出现。

特点：Web2.0网站的雏形 用户阅读内容，并开始更多地参与网站的交互

系统: 高并发访问，数据库压力增大 Memcached 缓存的出现，并出现数据库集群的概念

1. 微博，淘宝等网站出现，以及云计算平台出现，如google，百度广告，等大规模系统

特点: 用户重复参与网站交互，和内容提供，而系统需要对用户行为进行分析

系统：支持高并发，及时响应，并能够实现分布式计算

NOSQL：Memcached, Redis, Hbase, 等NOSQL数据库，不仅仅是简单缓存，并能够提供分布式要求，包括复制，分片，水平分区，并提供复杂格式的数据存储

总结:

1. 高并发读写操作

普通关系型数据库，很难满足高并发读写的要求，上万次读写的需求

1. 海量数据存储

上亿用户产生大量数据，包括用户数据，访问日志，用户提供内容，用户状态记录等，这种海量数据的存储，关系型数据库已经很难满足，尤其是从海量数据中提取和汇总数据成为瓶颈

1. 高可用性和高扩展性

关系型数据库也能支持，同步复制，水平分区扩展，但维护工作相当繁杂，并很难做到热扩展

1. 内存操作(快速), 并发量高(非阻塞进程), 硬盘回写(数据完整和高效的平衡),备份和恢复

复制(高可用性，和连续服务), 分片(热扩展，海量数据) 操作简单(JSON)，负载均衡，

分布式部署(局域网，和物理部分), 事务支持

#### 2. 主流的NOSQL介绍

|  |  |  |  |
| --- | --- | --- | --- |
| 数据库类型 | 开发语言 | 特性 | 应用场景 |
| CouchDB | Erlang | 特点：DB一致性，易于使用  使用许可： Apache  协议： HTTP/REST  双向数据复制  持续进行或临时处理  处理时带冲突检查  因此，采用的是master-master复制(见编注2)  MVCC - 写操作不阻塞读操作  可保存文件之前的版本  Crash-only(可靠的)设计  需要不时地进行数据压缩  视图：嵌入式 映射/减少  格式化视图：列表显示  支持进行服务器端文档验证  支持认证  根据变化实时更新  支持附件处理  因此，CouchApps(独立的 js应用程序)  需要 jQuery程序库 | 适用于数据变化较少，执行预定义查询，进行数据统计的应用程序。适用于需要提供数据版本支持的应用程序。  如CMS系统，数据统计汇总 |
| Redis | C | 特点：运行异常快  使用许可： BSD  协议：类 Telnet   1. 读写操作异常快 2. 较复杂的数据格式 sets, 链表，hash 3. 事务支持 4. 消息订阅 pub/sub 5. 主从同步复制 6. 硬盘回写 7. 第三方sharding支持 | 适用于数据变化快且数据库大小可遇见(适合内存容量)的应用程序  股票价格、数据分析、实时数据搜集、实时通讯  配置下发，全局实时数据 |
| Mongodb | C++ | 特点：保留了SQL一些友好的特性(查询，索引)。  使用许可： AGPL(发起者： Apache)  协议： Custom, binary( BSON)   1. 支持javascript表达式 2. 硬盘回写 3. 主从复制和集群功能 4. 内建分片机制 5. 较快读写性能 6. 大格式数据支持   7．空数据库大约占 192Mb  8. 事务支持 | 1．大数据量查询和汇总  2．分布式部署，和水平扩展  数据统计汇总  业务数据库，替代Mysql |
| Neo4j | Java | 特点：基于关系的图形数据库  使用许可： GPL，其中一些特性使用 AGPL/商业许可  协议： HTTP/REST(或嵌入在 Java中)  1．可独立使用或嵌入到 Java应用程序  2．图形的节点和边都可以带有元数据  3．很好的自带web管理功能  4．使用多种算法支持路径搜索  5．使用键值和关系进行索引  6．支持事务(用 Java api)  7．使用 Gremlin图形遍历语言  8．支持 Groovy脚本  9．支持在线备份，高级监控及高可靠性支持使用 | 适用于图形一类数据。这是 Neo4j与其他nosql数据库的最显著区别  例如：社会关系，公共交通网络，地图及网络拓谱 |
| HBase | Java | 特点：支持数十亿行X上百万列  使用许可： Apache  协议：HTTP/REST (支持 Thrift，见编注4)   1. 采用分布式架构 Map/reduce 2. 对实时查询进行优化 3. 高性能 Thrift网关 4. 通过在server端扫描及过滤实现对查询操作预判 5. 支持 XML, Protobuf, 和binary的HTTP 6. 对配置改变和较小的升级都会重新回滚 7. 不会出现单点故障 | 适用于偏好BigTable:)并且需要对大数据进行随机、实时访问的场合  报表，数据挖掘 |
| Memcache | C | 1. 快速 2. 简单 3. 分布式支持和热扩展 | KV数据库  数据库辅助缓存 |

#### 3．Mongodb概述

##### 3.1．Mongodb 内部文件和内存管理

* **结构: Database,Collection,Document**



|  |  |
| --- | --- |
| 逻辑关系对比 | |
| Mongodb | Mysql |
| Db(数据库) | Database(数据库) |
| Collection(集合) | Table(表) |
| Document(文档) | Row(行) |
| Field(列) | Col(字段) |

* **每个数据库都有相应的数据库文件**

示例: 新建了两个数据库(test,test1)，并在两个collection(user1,user2)分别插入记录

显示相关数据

[root@localhost db]# ls -l

-rwxr-xr-x 1 root root 5 12-23 17:07 mongod.lock

-rw------- 1 root root 16777216 12-23 17:08 test.0

-rw------- 1 root root 33554432 12-23 17:08 test.1

-rw------- 1 root root 16777216 12-23 17:08 test1.0

-rw------- 1 root root 33554432 12-23 17:08 test1.1

-rw------- 1 root root 16777216 12-23 17:08 test1.ns

-rw------- 1 root root 16777216 12-23 17:08 test.ns

drwxr-xr-x 2 root root 4096 12-23 17:08 \_tmp

分别按两倍递增，16M,32M,64M,128M…..2G，最大为2G,所以mongodb单服务器和数据库最大存储上限为2G

* **内存管理方式**

1. Mongod启动时，会载入相应文件到物理内存，并把内存管理交给系统，内存使用和结构如下所示，如果数据量很大，mongod启动的速度会较慢
2. 物理内存，虚拟内存，swap, MMAP

使用MMAP映射文件到虚拟内存，并映射虚拟内存到实际的物理内存



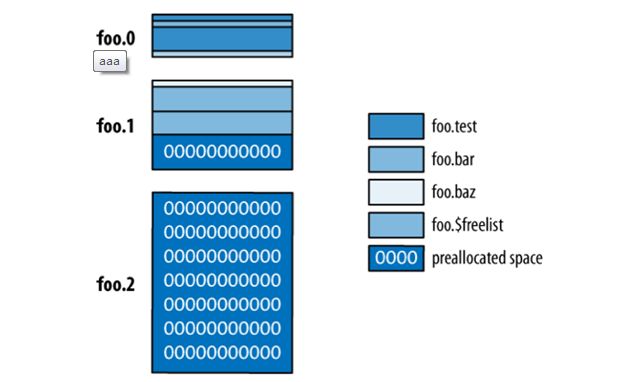
32bit 内存上限为 4G-1G(内核占用)-0.5堆栈 =2.5 即32位系统的内存可寻址上限是2.5G

62bit 内存上限为12T 所以mongodb的单数据库上限也相应的增加

所以32位系统mongod的单数据库文件上限为2G

1. 物理文件被分成相应的块，每个块之间使用双向链表来连接





1. 内存按照16M,32M,64M…2G递增方式进行分配，会预留相应的内存空间

这是mongod会大量消耗内存的原因，即使一条记录，也会占用16M的内存空间，并且还不包括test1.ns所预先分配的内存空间

这样的机制有利于防止较小的数据库浪费过多的磁盘空间，同时又能保证较大的数据库有相应的预留空间使用。

1. Test.ns 名字空间索引，保存着相应的文件索引

每一条记录保存着相应的名字空间(包括collection的关键信息)



1. 内存的分配，新加入的数据，看是否有适合的空闲内存块可以分配，如果没有则增加新的数据块

Mongodb需要定期来进行数据压缩，以释放掉相应的空闲内存，移动相应数据，把内存碎片整理成完整的内存块，以便于进行重新分配。

>repairDatabase()

>db.runCommand({ compact : 'yourCollection' });

分别进行数据库的压缩和单个集合的压缩

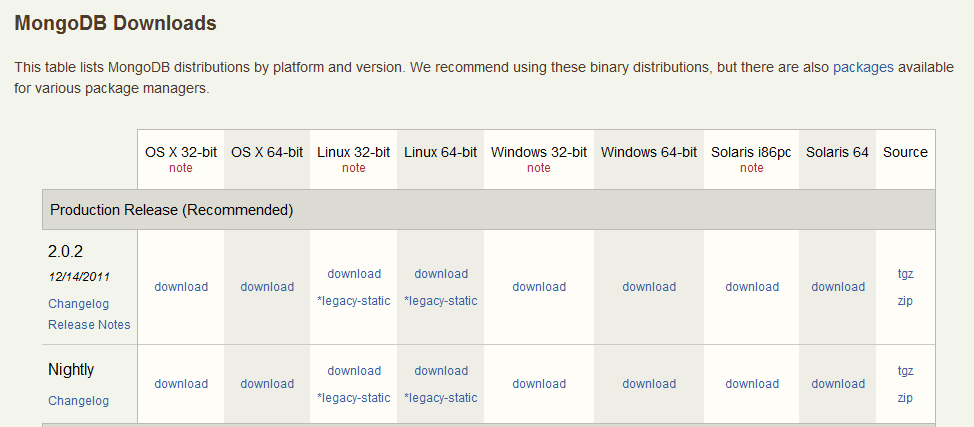
##### 3.2．BSON数据格式

索引的概念—数据库

### Mongdb的安装和配置

#### 下载

<http://www.mongodb.org/downloads>



* Linux ,windows, mac osx, solaris 编译和Source版安装
* 建议选择64bit版本，能够支持更大的数据存储，32bit仅仅支持最大2G的文件
* 建议选择编译版本安装，能够适应特性的硬件和系统平台，能够进行定制和配置安装，而且效率较高，不会出现兼容性错误，建议在产生环境使用
* 一般选择在linux平台安装

#### 在linux平台安装

步骤1: 解压

>cp xx.tgz /usr/local

>tar zxvf mongodb.tgz

步骤2: 建立相应的数据库目录和日志目录

>cd /usr/local/mongodb

>mkdir db

>mkdir logs

步骤3: 启动mongodb

>cd /usr/local/mongodb/bin

>/usr/local/mongodb/bin/mongod –dbpath=/usr/local/mongodb/db –logpath=/usr/local/mongodb/logs/mongo.log --fork

步骤4：安装相应的服务

> vi /etc/rc.local

添加 /usr/local/mongodb/bin/mongod –dbpath=/usr/local/mongodb/db –logpath=/usr/local/mongodb/logs/mongo.log --fork

步骤5 连接mongod

>mongo

步骤6 查看mongo日志

>vi /usr/local/mongodb/logs/mongodb.log

#### Windows平台安装

步骤1: 解压mongoxx.zip到指定目录 如c:\

步骤2: 建立相应的数据库目录和日志目录

>c:\mongodb\db

>c:\mongodb\logs

步骤3: 启动mongodb

>cd c:\mongodb\bin

>mongod –dbpath=c:\mongodb\db –logpath=c:\mongodb\logs\mongodb.log

步骤4：安装相应的服务

> mongod –dbpath=c:\mongodb\db –logpath=c:\mongodb\logs\mongodb.log

>net start mongodb

步骤5 连接mongod

>mongo

步骤6 查看mongo日志

#### Linux 源码安装(待完善)

#### 配置参数说明

基本配置

--------------------------------------------------------------------------------

--quiet # 安静输出

--port arg # 指定服务端口号，默认端口27017

--bind\_ip arg # 绑定服务IP，若绑定127.0.0.1，则只能本机访问，不指定默认本地所有IP

--logpath arg # 指定MongoDB日志文件，注意是指定文件不是目录

--logappend # 使用追加的方式写日志

--pidfilepath arg # PID File 的完整路径，如果没有设置，则没有PID文件

--keyFile arg # 集群的私钥的完整路径，只对于Replica Set 架构有效

--unixSocketPrefix arg # UNIX域套接字替代目录,(默认为 /tmp)

--fork # 以守护进程的方式运行MongoDB，创建服务器进程

--auth # 启用验证

--cpu # 定期显示CPU的CPU利用率和iowait

--dbpath arg # 指定数据库路径

--diaglog arg # diaglog选项 0=off 1=W 2=R 3=both 7=W+some reads

--directoryperdb # 设置每个数据库将被保存在一个单独的目录

--journal # 启用日志选项，MongoDB的数据操作将会写入到journal文件夹的文件里

--journalOptions arg # 启用日志诊断选项

--ipv6 # 启用IPv6选项

--jsonp # 允许JSONP形式通过HTTP访问（有安全影响）

--maxConns arg # 最大同时连接数 默认2000

--noauth # 不启用验证

--nohttpinterface # 关闭http接口，默认关闭27018端口访问

--noprealloc # 禁用数据文件预分配(往往影响性能)

--noscripting # 禁用脚本引擎

--notablescan # 不允许表扫描

--nounixsocket # 禁用Unix套接字监听

--nssize arg (=16) # 设置信数据库.ns文件大小(MB)

--objcheck # 在收到客户数据,检查的有效性，

--profile arg # 档案参数 0=off 1=slow, 2=all

--quota # 限制每个数据库的文件数，设置默认为8

--quotaFiles arg # number of files allower per db, requires --quota

--rest # 开启简单的rest API

--repair # 修复所有数据库run repair on all dbs

--repairpath arg # 修复库生成的文件的目录,默认为目录名称dbpath

--slowms arg (=100) # value of slow for profile and console log

--smallfiles # 使用较小的默认文件

--syncdelay arg (=60) # 数据写入磁盘的时间秒数(0=never,不推荐)

--sysinfo # 打印一些诊断系统信息

--upgrade # 如果需要升级数据库

\* Replicaton 参数

--------------------------------------------------------------------------------

--fastsync # 从一个dbpath里启用从库复制服务，该dbpath的数据库是主库的快照，可用于快速启用同步

--autoresync # 如果从库与主库同步数据差得多，自动重新同步，

--oplogSize arg # 设置oplog的大小(MB)

\* 主/从参数

--------------------------------------------------------------------------------

--master # 主库模式

--slave # 从库模式

--source arg # 从库 端口号

--only arg # 指定单一的数据库复制

--slavedelay arg # 设置从库同步主库的延迟时间

\* Replica set(副本集)选项：

--------------------------------------------------------------------------------

--replSet arg # 设置副本集名称

\* Sharding(分片)选项

--------------------------------------------------------------------------------

--configsvr # 声明这是一个集群的config服务,默认端口27019，默认目录/data/configdb

--shardsvr # 声明这是一个集群的分片,默认端口27018

--noMoveParanoia # 关闭偏执为moveChunk数据保存?

#### 配置文件示例

# mongo.conf

#where to log

logpath=/var/log/mongo/mongod.log

logappend=true

# fork and run in background

fork = true

#port = 27017

dbpath=/var/lib/mongo

# Enables periodic logging of CPU utilization and I/O wait

#cpu = true

# Turn on/off security. Off is currently the default

#noauth = true

#auth = true

# Verbose logging output.

#verbose = true

# Inspect all client data for validity on receipt (useful for

# developing drivers)

#objcheck = true

# Enable db quota management

#quota = true

# Set oplogging level where n is

# 0=off (default)

# 1=W

# 2=R

# 3=both

# 7=W+some reads

#oplog = 0

# Diagnostic/debugging option

#nocursors = true

# Ignore query hints

#nohints = true

# Disable the HTTP interface (Defaults to localhost:27018).

#nohttpinterface = true

# Turns off server-side scripting. This will result in greatly limited

# functionality

#noscripting = true

# Turns off table scans. Any query that would do a table scan fails.

#notablescan = true

# Disable data file preallocation.

#noprealloc = true

# Specify .ns file size for new databases.

# nssize = <size>

# Accout token for Mongo monitoring server.

#mms-token = <token>

# Server name for Mongo monitoring server.

#mms-name = <server-name>

# Ping interval for Mongo monitoring server.

#mms-interval = <seconds>

# Replication Options

# in replicated mongo databases, specify here whether this is a slave or master

#slave = true

#source = master.example.com

# Slave only: specify a single database to replicate

#only = master.example.com

# or

#master = true

#source = slave.example.com

# Address of a server to pair with.

#pairwith = <server:port>

# Address of arbiter server.

#arbiter = <server:port>

# Automatically resync if slave data is stale

#autoresync

# Custom size for replication operation log.

#oplogSize = <MB>

# Size limit for in-memory storage of op ids.

#opIdMem = <bytes>

#### 停止mongodb

**命令方式**

>use admin

switched to db admin

>db.shutdownServer()

**进程方式**

ps –aef | grep mongod

kill -2 <PID>

### 简单的数据操作

#### 1. 切换并产生数据库

>use tutorise

在切换数据库时，如果数据库不存在，则直接产生数据库

#### 2. 插入相应数据

>db.user.insert({username:'gaoyu'})

当collection不存在时，产生相应的数据库集，并插入相应数据

此过程会分配内存，并产生相应的文件，同时插入数据，所以第一次插入的速度会相对较慢

#### 3. 显示所有数据

>db.user.find()

{ "\_id" : ObjectId("4ef311a1776fbc2bc7dd038f"), "username" : "gaoyu" }

没有附带相应的查询条件，所以会显示所有的数据条目

#### 4. 继续插入数据

>db.user.save({username:'Jone'})

>db.user.count()

并统计整体collection 的数据条目数

解释相应的ObjectID,这是在Collection中唯一的数值，是mongodb自动进行分配的，HASH

#### 5. 进行相应的查询

>db.user.find({username:'gaoyu'})

查询username 为gaoyu的所有记录，

find(查询匹配表达式) 查询匹配表达式也是JSON形式，字段:'内容'

#### 6. 数据更新

数据更新分为两种形式，包括简单数据更新，和复杂数据更新

简单数据更新

>db.user.update({username:'gaoyu'},{$set:{countty:'tianjin'}})

db.user.update({更新查询条件},{$set:{更新内容}})

更新符合查询条件的内容，如果字段不存在，则添加相应字段并进行更新

更新复杂的数据

db.users.update( {username: "Jone"},

{ $set:{favorites:

{

cities: ["Chicago", "Cheyenne"],

movies: ["Casablanca", "The Sting"]

}

}

})

#### 7. 删除数据

进行Collection的整体删除

>db.user.remove()

仅仅是移除数据从user,而并不是进行物理删除，如果要进行实际的物理删除可以使用

>db.user.drop()

进行实际的物理删除

#### 8. 产生相应的索引

1. 插入相应的200000条数据

>for(i=0;i<2000000;i++){

>...db.numbers.save({num:i});

>...}

2. 查询相应的数据

>db.numbers.find()

>db.numbers.count()

>200000

3. 进行相应数据范围查询

db.numbers.find( {num: {"$gt": 199995 }} ).explain()

{

{

"cursor" : "BasicCursor",

"nscanned" : 200000,

"nscannedObjects" : 200000,

"n" : 4,

"millis" : 136,

"nYields" : 0,

"nChunkSkips" : 0,

"isMultiKey" : false,

"indexOnly" : false,

"indexBounds" : {

}

}

1. "cursor" : "BasicCursor", 表示仅仅使用了基础游标，对整表进行了相应的扫描

2. "nscanned" : 200000, 扫描记录数为200000

3. "n" : 4, 得到四条匹配记录

4. "millis" : 136, 用时 136毫妙

5. "indexOnly" : false, 是否使用索引

### 常用工具集

MongoDB在bin目录下提供了一系列有用的工具，这些工具提供了MongoDB在运维管理上方便。

 bsondump: 将bson格式的文件转储为json 格式的数据

 mongo: 客户端命令行工具，其实也是一个js 解释器，支持js 语法

 mongod: 数据库服务端，每个实例启动一个进程，可以fork 为后台运行

 mongodump/ mongorestore: 数据库备份和恢复工具

 mongoexport/ mongoimport: 数据导出和导入工具

 mongofiles: GridFS管理工具，可实现二制文件的存取

 mongos: 分片路由，如果使用了sharding 功能，则应用程序连接的是mongos而不是

 mongosniff: 这一工具的作用类似于tcpdump，不同的是他只监控MongoDB相关的包请求，并且是以指定的可读性的形式输出

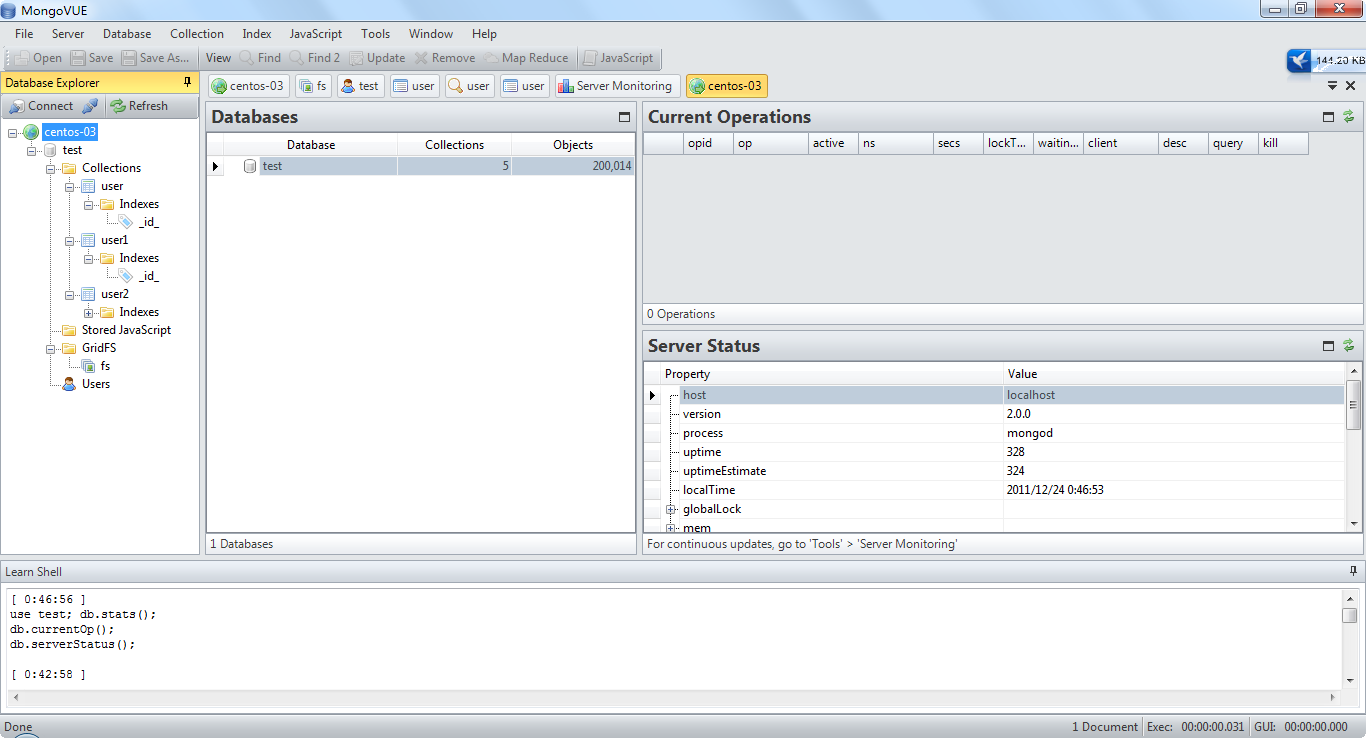
 mongostat: 实时性能监控工具

### 推荐客户端工具

#### [MongoVUE](http://blog.mongovue.com/)

<http://blog.mongovue.com/>

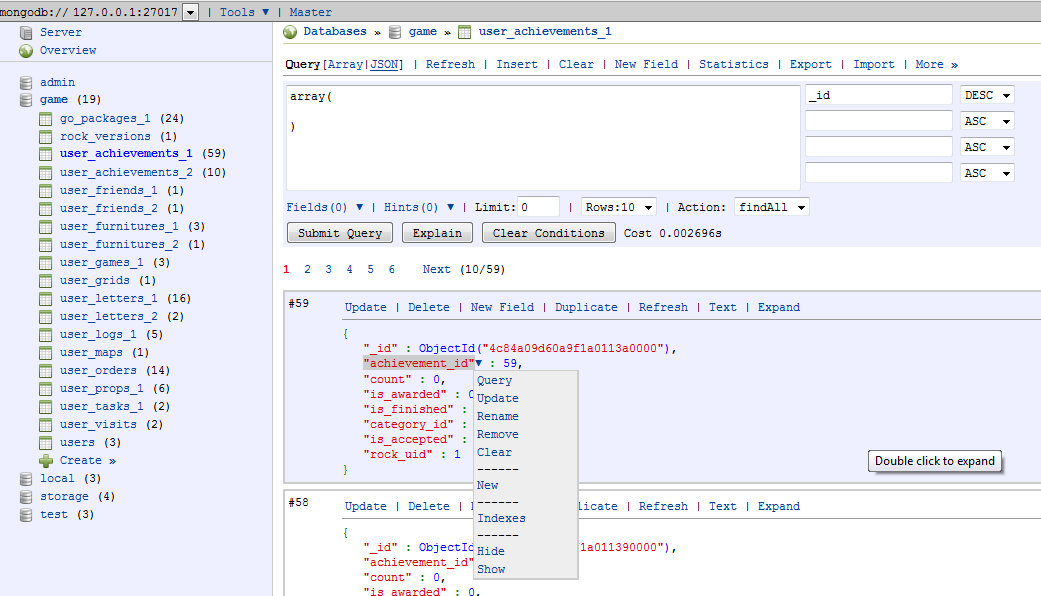
一个windows下的客户端管理工具，对于未来的功能有一个长长的roadmap。



#### rock\_mongo

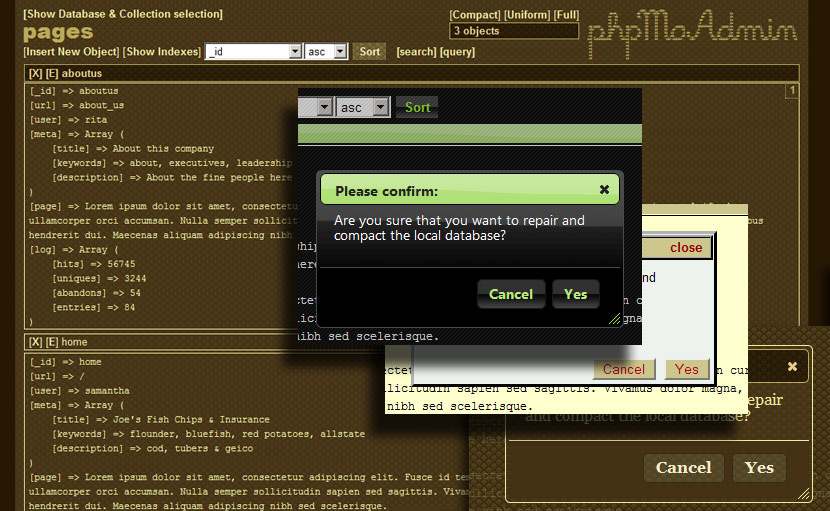
它的描述是“Best PHP based MongoDB administration GUI tool” ，最近 MongoDB 的讨论组上很多人推荐。

<http://code.google.com/p/rock-php/>



#### phpMoAdmin

<http://www.phpmoadmin.com>



## 二．mongoDB应用开发

### MongoDB机制

#### 组成

mongodb分为客户端，和服务器两部分，分别按照Mongodb的通信协议负责不同的职责，共同完成mongodb的操作

#### 客户端：

1. 负责生成\_id hash，按照相应的规则
2. 把客户端提交的内容JSON转化为BSON(二进制的JSON)
3. 和服务端通过TCP Socket进行通讯，把客户端提交内容传递给服务器
4. 接受服务端的返回，并把BSON反序列化为JSON

#### \_ID

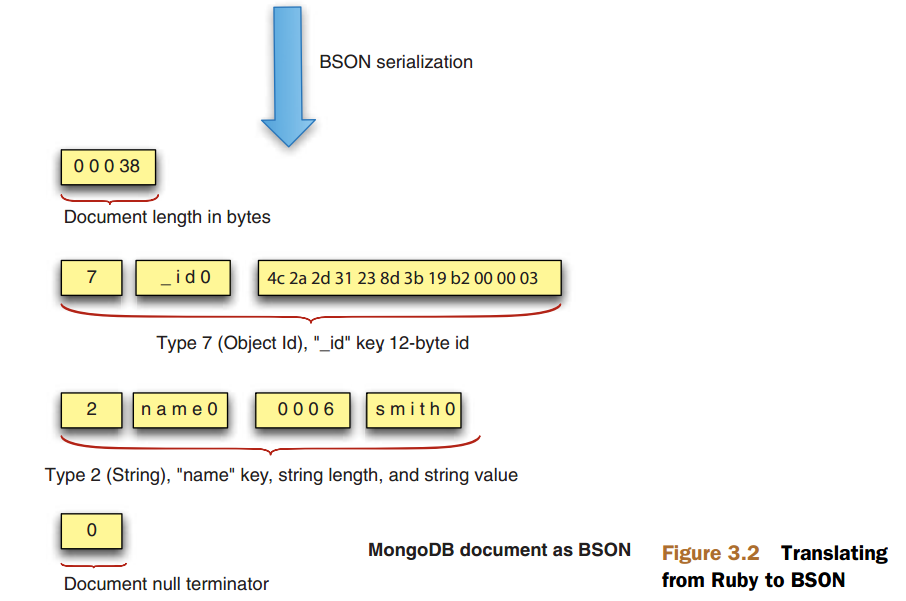
生成一个全局唯一的ID

4c342312 238d3c 19bc 000001

4位时间戳 机器ID 进程ID 计数器

#### BSON

<http://www.mongodb.org/display/DOCS/Mongo+Wire+Protocol>



#### 进行相应的处理

Mongodb进行相应处理，包括查询和插入操作，并返回相应的值(BSON)

### 设计mongodb数据库

包含以下主题

* 设计数据库的标准，能够满足应用程序的需求
* 能够表现，一对一，一对多，多对多模式

#### 电子商务数据库示例

**商品表**

products =

{\_id: new ObjectId("4c4b1476238d3b4dd5003981"),

slug: "book-barrow-9092",

sku: "9092",

name: "mongodb 开发手册",

descriptuon: "NOSQL数据库开发",

detail{

weight: 230,

weight\_unit: "g",

count: 200,

manufacturer: "Tech",

color: "red"

},

totel\_view: 4,

average\_view: 4.5

price{

retail: 45,

sale:40

}

price\_history{[

{

retail:45

sale:40

start\_date: new Date(2011,11,1),

end\_date: new Date(2011,12,25)

},

{

retail:45

sale:30

start\_date: new Date(2011,10,1),

end\_date: new Date(2011,10,31)

}

]},

category\_main: new ObjectId("6a5b1476238d3b4dd5000049"),

tags:["mongodb","computer","book","NOSQL"]

}

产生slug唯一索引，保证sku在products的唯一性

>db.products.ensureIndex({sku: 1}, {unique: true})

>db.products.insert(name:”redis 开发手册”,

>…sku: 9092,

>…slug: “redis cookbook”

>…safe:true

>…}

系统会拒绝进行相应的插入操作，因为唯一索引的问题

1. Field和字段的概念有所区别，能够递归包含

\

设计mongodb数据库

包含以下主题

 设计数据库的标准，能够满足应用程序的需求

 能够表现，一对一，一对多，多对多模式

1． 电子商务数据库示例

商品表

products =

{\_id: new ObjectId("4c4b1476238d3b4dd5003981"),

slug: "book-barrow-9092",

sku: "9092",

name: "mongodb 开发手册",

descriptuon: "NOSQL数据库开发",

detail{

weight: 230,

weight\_unit: "g",

count: 200,

manufacturer: "Tech",

color: "red"

},

totel\_view: 4,

average\_view: 4.5

price{

retail: 45,

sale:40

}

price\_history{[

{

retail:45

sale:40

start\_date: new Date(2011,11,1),

end\_date: new Date(2011,12,25)

},

{

retail:45

sale:30

start\_date: new Date(2011,10,1),

end\_date: new Date(2011,10,31)

}

]},

category\_main: new ObjectId("6a5b1476238d3b4dd5000049"),

tags:["mongodb","computer","book","NOSQL"]

}

产生slug唯一索引，保证sku在products的唯一性

>db.products.ensureIndex({sku: 1}, {unique: true})

>db.products.insert(name:”redis 开发手册”,

>…sku: 9092,

>…slug: “redis cookbook”

>…safe:true

>…}

系统会拒绝进行相应的插入操作，因为唯一索引的问题

1． Field和字段的概念有所区别，能够递归包含

#### 关系型数据库设计原则:

范式但出于性能和特殊要求，数据库设计的规则会发生相应的改变

##### 模式1: 一对多模式

和mysql不同，由于mongodb能够保存相应的子文档，所以表现一对多关系可以采用两种方式，分别是内嵌式和关联式

内嵌式表达:

关联式:

广告订单--广告投放

广告订单:

{

\_id: ObjectId("4d650d4cf32639266022018d"),

name:"宝洁广告订单1",

start\_date:"120909404",

end\_date:"133949494",

...

}

广告投放:

{

\_id: ObjectId("4d650d4cf32639266022ac01"),

name:"北京推广1",

order\_id:"4d650d4cf32639266022018d",

...

}

内嵌式:

{ title: "How to soft-boil an egg",

steps: [

{ desc: "Bring a pot of water to boil.",materials: ["water", "eggs"] },

{ desc: "Gently add the eggs a cook for four minutes.", materials: ["egg timer"]},

{ desc: "Cool the eggs under running water." },

]

}

1. 由于组织结构不同，处理一对多关系，不同于关系型数据库仅仅采用一种关联模式，而存在两种模式

2. 当关联不属于从属关系，而且存在双向查询，则采用关联模式，并需要进行关联键进行索引

3. 当两者属于包含关系，则采用内嵌方式，并且被包含对象不会经常变化，并不会进行双向查询，或对被包含对象进行其他关联查询

4. 是否属于从属关系，是否会进行双向查询

##### 模式2: 多对多关联

由于mongodb不存在连接查询，所以仅仅存在一种方式，即内嵌数组方式

产品--类目

products

{

\_id: ObjectId("4d650d4cf32639266022018d"),

title:"mongodb 开发手册"

category\_ids:[

ObjectId("4d6574baa6b804ea563c132a"),

ObjectId("4d6574baa6b804ea563c459d")

],

...

}

category

{

\_id: ObjectId("4d6574baa6b804ea563c132a"),

title:"技术",

...

}

category

{

\_id: ObjectId("4d6574baa6b804ea563c459d"),

title:"NOSQL",

...

}

1. 采用内嵌数据的方式表达相应的关联

2. 内嵌数据放在那里，决定于查询的方向频率，如果查询类目下所属产品的频率，高于查询产品所包含的类目，则应该把关联键放在产品下

3. 因为分别需要一步和两步

4. 为加速查询，关联键需要添加索引

##### 模式3: 树

{

\_id: ObjectId("4d692b5d59e212384d95001"),

depth: 0,

path:null,

created: ISODate("2011-02-26T17:18:01.251Z"),

username: "plotinus",

body:"Who was Alexander the Great's teacher?",

thread\_id: ObjectId("4d692b5d59e212384d95223a")

}

{

\_id: ObjectId("4d692b5d59e212384d951002"),

depth: 1,

path:"4d692b5d59e212384d95001",

created: ISODate("2011-02-26T17:21:01.251Z"),

username: "asophist",

body:"It was definitely Socrates.",

thread\_id: ObjectId("4d692b5d59e212384d95223a")

}

{

\_id: ObjectId("4d692b5d59e212384d95003"),

depth: 2,

path:"4d692b5d59e212384d95001:4d692b5d59e212384d951002",

created: ISODate("2011-02-26T17:21:01.251Z"),

username: "daletheia",

body:"Oh you sophist...It was actually Aristotle!",

thread\_id: ObjectId("4d692b5d59e212384d95223a")

}

1. 添加相应的索引 db.comments.ensureIndex({thread\_id: 1}) db.comments.ensureIndex({path: 1}

2. 查询整个树 db.comments.find({thread\_id: ObjectId("4d692b5d59e212384d95223a")})

3. 查询某节点的子节点 db.comments.find({path: /^4d692b5d59e212384d95001/})

##### 模式4 树2

{

\_id: ObjectId("4d692b5d59e212384d95003"),

comment\_id: 1,

nsleft:0,

nsright:0,

...

}

1. 采用左右数的方式能够支持更快速的树形遍历

2. 方便的进行索引

3. 但缺点是更新的删除操作需要一定的算法和附加的写操作

##### 模式5：可变属性

mongodb可以很好的支持可变属性

如两个产品类型

{

\_id: ObjectId("4d669c225d3a52568ce07646")

sku: "ebd-123"

name:"飞利浦耳机",

type:"小电器",

attrs: {

color: "银色",

freq\_low: 20,

freq\_hi: 22000,

weight: 0.5

}

}

{

\_id: ObjectId("4d669c225d3a52568ce07646")

sku: "ssd-456"

name:"SSD卡",

type:"硬件",

attrs: {

interface: "SATA",

capacity: 1.2 \* 1024 \* 1024 \* 1024,

rotation: 7200,

form\_factor: 2.5

}

}

第二种方式

{ \_id: ObjectId("4d669c225d3a52568ce07646")

sku: "ebd-123"

name:"耳机2",

type:"小家电",

attrs: [

{n: "color", v: "silver"},

{n: "freq\_low", v: 20},

{n: "freq\_hi", v: 22000},

{n: "weight", v: 0.5}

]

}

1. 表现可变属性可以使用两种方式，1：直接输用内嵌对象的方式，2:使用名值对数组

2. 采用哪种方式有两个决定因素，分别是属性变化是否可知，包括属性类型和属性数量

3. 方式1:便于检索，和显示，而方式2: 会有更大的通讯信息量，但便于进行索引

4. db.products.ensureIndex({"attrs.n": 1, "attrs.v": 1})

5. db.products.ensureIndex({"attrs.freq\_low": 1, "attrs.freq\_hi": 1},{sparse: true})

##### 模式6：不要忽视索引

索引对于mongodb的优化起到决定性作用，而通过分析应用程序的查询条件，进而建立相应的索引，对于系统的

优化起到决定性作用，不要忽视索引

db.user.ensureIndex("title":1}

##### 模式7: 不要混杂

由于mongodb的特性，所以容易引导把不同的对象，放在同一document中，所以需要大家尽量把不同的对象放在不同的文档中

便于后期的扩展和索引的优化，

##### 模式8：多值属性

可以参考模式1

不要采用相应的值直接写入

### 高级数据库操作

#### 数据类型

 string, integer, boolean, double, null, array, and object.

日期：

>x =new Date()

>d=new ISODate()

>d.getYear()

>d.getMonth()

>

#### 数据库

##### 建立数据库

产生数据库,转到相应数据，并产生数据库

1. 判断数据库是否存在，如果不存在则在硬盘产生相应的文件

>use magento

##### 删除数据库

删除相应的数据库，和数据库内部的集合，并删除相应的索引，同时删除相应物理文件

>use magento

>db.dropDatabase()

##### 查看数据库状态

> db.stats()

{

"collections" : 3,

"objects" : 10004,

"avgObjSize" : 36.005,

"dataSize" : 360192,

"storageSize" : 791296,

"numExtents" : 7,

"indexes" : 1,

"indexSize" : 425984,

"fileSize" : 201326592,

"ok" : 1

}

##### 系统命令

system.namespaces 显示所有名字空间

system.indexes 显示所有索引

system.profile stores database profiling information.

system.users 显示数据库用户

local.sources 显示所有源

#### 集合

##### 新建集合

产生集合，并分配相应的空间

>db.createCollection(“users”)

>db.createCollection("users", {size: 20000})

##### 显示集合列表

>show collections

##### 修改集合名称

>db.oldname.renameCollection(“newName”)

>db.runCommand({renameCollection:”oldname”,to:”mydb.newname”})

#### Capped Collection 集合

##### 概念

1. Capped集合是定长文档，记录集的长度是定长，定数量的
2. Capped集合的数据库尺寸是预先定制的，如1024K
3. 我们可以预算得到capped记录集的document数，
4. Capped 的按先后顺序进行插入，如果文档数溢出，则丢弃最早的记录，先进先出的定长队列，并且有系统自动实现
5. 不存在索引，插入和提取速度都相当快

##### 应用:

1. 日志系统: 保留最早日志，并自动抛弃较早日志，或写入到其他集合中
2. 最近浏览: 保存用户最近浏览的记录
3. 最新内容: 保存指定数量的最新内容

##### 产生

>db.createCollection("mycoll", {capped:true, size:100000})

>db.user.actions.count();

>db.user.actions.find();

>db.user.actions.find().sort({"$natural": -1})

#### 系统集合

##### 系统名字空间

> db.system.namespaces.find();

{ "name" : "garden.products" }

{ "name" : "garden.system.indexes" }

{ "name" : "garden.products.$\_id\_" }

{ "name" : "garden.user.actions", "options" :

{ "create": "user.actions", "capped": true, "size": 1024 } }

##### 系统索引

> db.system.indexes.find();

{ "name" : "\_id\_", "ns" : "garden.products", "key" : { "\_id": 1 } }

#### 索引

Mongo索引采用btree方式，回顾mysql索引

索引是?

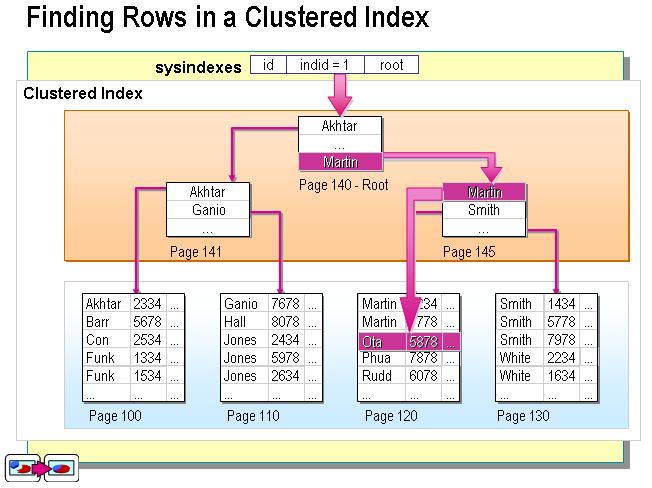
Mongodb索引的组织形式?

索引操作

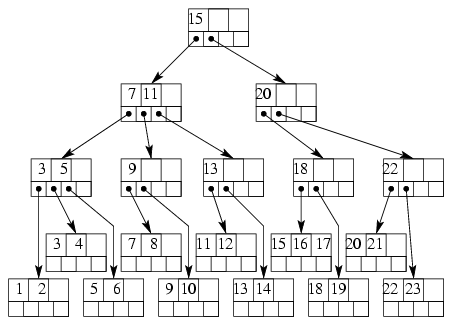
##### 索引的概念

聚类索引，一般索引，btree索引 btree+索引

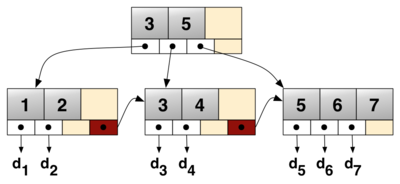
聚类



Btree



Btree+



##### 索引效果

1. 插入相应的200000条数据

>for(i=0;i<2000000;i++){

>...db.numbers.save({num:i});

>...}

2. 查询相应的数据

>db.numbers.find()

>db.numbers.count()

>200000

3. 进行相应数据范围查询

db.numbers.find( {num: {"$gt": 199995 }} ).explain()

{

{

"cursor" : "BasicCursor",

"nscanned" : 200000,

"nscannedObjects" : 200000,

"n" : 4,

"millis" : 136,

"nYields" : 0,

"nChunkSkips" : 0,

"isMultiKey" : false,

"indexOnly" : false,

"indexBounds" : {

}

}

1. "cursor" : "BasicCursor", 表示仅仅使用了基础游标，对整表进行了相应的扫描

2. "nscanned" : 200000, 扫描记录数为200000

3. "n" : 4, 得到四条匹配记录

4. "millis" : 136, 用时 136毫妙

5. "indexOnly" : false, 是否使用索引

> db.numbers.ensureIndex({num:1})

> db.numbers.find( {num: {"$gt": 199995 }} ).explain()

##### 建立索引

>db.numbers.ensureIndex({num:1},{background : true，unique:true, sparse:true, dropDups:true})

>db.factories.ensureIndex( { "metro.city" : 1, "metro.state" : 1 } );

##### 查看索引

> db.numbers.getIndexes()

>db.system.indexes.find()

##### 复合索引

db.things.ensureIndex({j:1, name:-1});

a,b,c

可以使用在下列情况

a

a,b

a,b,c

##### 唯一索引

db.things.ensureIndex({firstname: 1, lastname: 1}, {unique: true});

##### 删除索引

db.collection.dropIndexes();

db.collection.dropIndex({x: 1, y: -1})

##### 重建索引

db.myCollection.reIndex()

#### 插入

> doc = { author : 'joe', created : new Date('03/28/2009') }

> db.posts.insert(doc);

#### 修改

如果增加相应的field，则把修改记录移动到尾部，并进行field添加和修改

##### Update操作

db.collection.update( criteria, objNew, upsert, multi )

criteria:条件

objNew:对象

upsert:如果不存在，插入

multi:多值匹配

##### Set

{ $set : { field : value } }

> db.users.update({"name" : "joe"},

... {"$set" : {"favorite book" :

... ["cat's cradle", "foundation trilogy", "ender's game"]}} )

##### Unset

{ $unset : { field : 1} }

>db.user.update({name:”test”},{“$unset”:{score:1}})

##### Inc

{ $inc : { field : value } }

> > db.games.update({"game" : "pinball", "user" : "joe"},

... {"$inc" : {"score" : 10000}})

仅仅能够应用在数值类型的field,也可以使用负值，相当于-

##### push

{ $push : { field : value } }

> db.user.update({name:”gaoyu”},{“$push”:{favor:”tv”}})

>db.user.update({name:”gaoyu”},{“$push”:{favor:”basketball”}})

如果字段存在，并且是数组类型，则压入相应的值到数据，如果不存在，则创建数组字段，并压入值

##### pushAll

{ $pushAll : { field : value\_array } }

> db.user.update({name:”gaoyu”},{“$push”:{favor:[“tv1”,”tv2”]}})

压入一个数组，如果字段存在，则压入所有值，不存在，则添加数组，不是数组类型报错

##### pop

{ $pop : { field : 1 } } { $pop : { field : -1 } }

> db.user.update({name:”gaoyu”},{“$pop”:{favor:1}})

>db.user.update({name:”gaoyu”},{“$pop”:{favor:1}})

弹出最后一个值，堆栈,知道数组为空，如果不是数组类型则报错

##### Pull

{ $pull : { field : {<match-criteria>} } }

弹出符合条件的值

##### Rename

修改字段名

{ $rename : { old\_field\_name : new\_field\_name } }

##### 锁

db.students.update({score: {$gt: 60}, $atomic: true}, {$set: {pass: true}}, false, true)

分片不支持加锁

##### FindandModify

修改，并返回相应值

> db.jobs.save( {

name: "Next promo",

inprogress: false, priority:0,

tasks : [ "select product", "add inventory", "do placement"]

} );

> db.jobs.save( {

name: "Biz report",

inprogress: false, priority:1,

tasks : [ "run sales report", "email report" ]

} );

> db.jobs.save( {

name: "Biz report",

inprogress: false, priority:2,

tasks : [ "run marketing report", "email report" ]

} );

> job = db.jobs.findAndModify({

query: {inprogress: false, name: "Biz report"},

sort : {priority:-1},

update: {$set: {inprogress: true, started: new Date()}},

new: true

});

{

"\_id" : ...,

"inprogress" : true,

"name" : "Biz report",

"priority" : 2,

"started" : "Mon Oct 25 2010 11:15:07 GMT-0700 (PDT)",

"tasks" : [

"run marketing report",

"email report"

]

}

#### 删除

删除整体数据，但不删除索引

>db.a.remove()

删除指定条件的数据

>db.a.remove({username:”test”})

>db.a.remove({\_id:”assdd”}) 推荐

>db.a.remove(object) 不推荐

删除指定的集合，并删除索引，速度比单条删除记录要快

>db.test.drop();

删除锁

>db.videos.remove( { rating : { $lt : 3.0 }, $atomic : true } )

删除会在内存和文件产生碎片，通过repairDatabase()，来整理内存碎片

#### 查询

##### 查询整体数据集合

>db.catalog.find();

没有给出条件就是查询完全数据集，但客户端仅仅返回50条记录，如果继续查看请键入it

##### 根据条件进行查询

>db.catalog.find({product\_id:117})

>数据库.collection.find(查询条件), 其中查询条件为JSON格式

##### 组合条件查询

> db.catalog.find({"product\_id":117, "category\_id":3})

>数据库.collection.find({查询条件1, 查询条件2}) 两个条件之间使用,分割

##### 返回指定的字段

>db.catalog.find({product\_id: 117},{product\_id:1,category\_id:1})

格式：db.collection.find({查询条件},{field1:1,field2:1}) 第二个参数显示需要显示的字段

>db.catalog.find({product\_id: 117},{product\_id:1,category\_id:1,\_id:0})

排除某些字段不进行显示

##### 范围查询

>db.catalog.find({product\_id:{$gt:100}}) $gt 查询product\_id 大于100的记录

>db.catalog.find({product\_id:{$lt:100}}) $lt 查询product\_id 小于100的记录

>db.catalog.find({product\_id:{$gt:100,$lt:200}}) $gt,$lt组合使用，查询product\_id 大于100且小于200的记录

$lt:小于 $lte:小于等于 $gt 大于 $gte 大于等于 $ne不等于

##### In查询

$in 查询值在指定的集合内进行匹配

$nin 对不在指定集合条件的记录进行匹配

>db.catalog.find({product\_id: {$in:[121,128,129]}})

> db.catalog.find({product\_id: {$nin:[121,128,129]}})

##### 或查询

$or 满足两个条件之一的记录进行相应的匹配

>db.catalog.find({$or:[{product\_id:121},{category\_id:3}]})

>db.catalog.find({$or:[{product\_id:{$in:[121,128,129]},{category\_id:3}}]

##### Limit和skip

限制返回的记录集数

>db.catalog.find().limit(n)

掠过指定的记录集个数，从第n+1条记录开始显示

>db.catalog.find().skip(n)

##### Sort排序

分别按照指定顺序进行排序，类似于1正序，-1倒序

>db.c.find().sort({username:1,age:-1})

>db.c.find().limit(50).skip(50).sort({username:1})

##### 随机值查询

>var random = Math.random()

>result = db.foo.findOne({“random”:{“$gt”:random})

##### Null查询

>db.c.find(“z”:{“$exists”:true})

使用$exists查询指定字段为null值的记录，而不能使用 xxx:null

##### 数组查询

**单值匹配**

>db.c.find({fruit:”apple” })

多值匹配

>db.c.find({fruit:{all:[“apple”,”banana”]})

##### \*\*\*Sql查询的映射图

|  |  |
| --- | --- |
| Sql | Mongodb |
| SELECT \* FROM users | Db.users.find() |
| SELECT a,b FROM users | Db.users.find({},{a:1,b:1}) |
| SELECT \* FROM users WHERE age=33 | Db.users.find({age:33}) |
| SELECT a,b FROM users WHERE age=33 | Db.users.find({age:33},{a:1,b:1}) |
| SELECT \* FROM users WHERE age=33 ORDER BY name | Db.users.find({age:33}).sort({name:1}) |
| SELECT \* FROM users WHERE age>33 | Db.users.find({age:{“$gt”:33}}) |
| SELECT \* FROM users WHERE age!=33 | Db.users.find({age:{“$ne”:33}}) |
| SELECT \* FROM users WHERE name LIKE "%Joe%" | Db.users.find({name:”\Joe\”}) |
| SELECT \* FROM users WHERE name LIKE "Joe%" | Db.users.find({name:\^Joe\}) |
| SELECT \* FROM users WHERE age>33 AND age<=40 | Db.users.find({age:{“$gt”:33},age:{“lte”:40}}) |
| SELECT \* FROM users ORDER BY name DESC | Db.user.find().sort({name:-1}) |
| SELECT \* FROM users WHERE a=1 and b='q' | Db.users.find({a:1,b:”q”}) |
| SELECT \* FROM users LIMIT 10 SKIP 20 | Db.users.find().limit(10).skip(20) |
| SELECT \* FROM users WHERE a=1 or b=2 | db.users.find( { $or : [ { a : 1 } , { b : 2 } ] } ) |
| SELECT \* FROM users LIMIT 1 | Db.users.findOne() |
| SELECT order\_id FROM orders o, order\_line\_items li WHERE li.order\_id=o.order\_id AND li.sku=12345 | db.orders.find({"items.sku":12345},{\_id:1}) |

#### 聚合

**Count计数**

>db.catalog.count()

对collection的记录进行相应的计数，计数对于统计数据库记录数，并进行相应的分页

#### GRIDFS

##### 简介

GridFS主要是用来存储大型文件，如高清图片，视频，或文本等

### 安装PHP客户端

#### Linux

$ tar zxvf mongodb-mongdb-php-driver-<commit\_id>.tar.gz

$ cd mongodb-mongodb-php-driver-<commit\_id>

$ phpize

$ ./configure

$ sudo make install

编辑php.ini,添加

Extension=”mongo.so”

### 应用案例

#### 案例1 电子商务

产品表:

{

\_id: new ObjectId("4c4b1476238d3b4dd5003981"),

slug:"wheel-barrow-9092",

sku: "9092",

name:"Extra Large Wheel Barrow",

description: "Heavy duty wheelbarrow...",

details: {

weight: 47,

weight\_units: "lbs",

model\_num: 4039283402,

manufacturer: "Acme",

color: "Green"

},

total\_reviews: 4,

average\_review: 4.5,

pricing: {

retail: 589700,

sale:489700,

},

price\_history: [

{

retail: 529700,

sale: 429700,

start: new Date(2010, 4, 1),

end: new Date(2010, 4, 8)

},

{

retail: 529700,

sale: 529700,

start: new Date(2010, 4, 9),

end: new Date(2010, 4, 16)

},

],

category\_ids: [

new ObjectId("6a5b1476238d3b4dd5000048"),

new ObjectId("6a5b1476238d3b4dd5000049")

],

main\_cat\_id: new ObjectId("6a5b1476238d3b4dd5000048"),

tags:["tools", "gardening", "soil"],

}

db.products.ensureIndex({slug: 1}, {unique: true})

类目表

doc =

{ \_

id: new ObjectId("6a5b1476238d3b4dd5000048"),

slug:"gardening-tools",

ancestors: [

{

name: "Home",

\_id: new ObjectId("8b87fb1476238d3b4dd500003"),

slug: "home"

},

{ name: "Outdoors",

\_id: new ObjectId("9a9fb1476238d3b4dd5000001"),

slug: "outdoors"

}

],

parent\_id: new ObjectId("9a9fb1476238d3b4dd5000001"),

name:"Gardening Tools",

description: "Gardening gadgets galore!",

}

订单表

doc =

{ \_

id: ObjectId("6a5b1476238d3b4dd5000048")

user\_id: ObjectId("4c4b1476238d3b4dd5000001")

state: "CART",

line\_items: [

{

\_id: ObjectId("4c4b1476238d3b4dd5003981"),

sku: "9092",

name: "Extra Large Wheel Barrow",

quantity: 1,

pricing: {

retail: 5897,

sale: 4897,

}

},

{

\_id: ObjectId("4c4b1476238d3b4dd5003981"),

sku: "10027",

name: "Rubberized Work Glove, Black",

quantity: 2,

pricing: {

retail: 1499,

sale: 1299

}

}

],

shipping\_address: {

street: "588 5th Street",

city:"Brooklyn",

state: "NY",

zip: 11215

},

sub\_total: 6196

}

用户表

{

\_id: new ObjectId("4c4b1476238d3b4dd5000001"),

username: "kbanker",

email: "kylebanker@gmail.com",

first\_name: "Kyle",

last\_name: "Banker",

hashed\_password: "bd1cfa194c3a603e7186780824b04419",

addresses: [

{

name: "home",

street: "588 5th Street",

city:"Brooklyn",

state: "NY",

zip: 11215

},

{

name: "work",

street: "1 E. 23rd Street",

city:"New York",

state: "NY",

zip: 10010

}

],

payment\_methods: [

{

name: "VISA",

last\_four: 2127,

crypted\_number: "43f6ba1dfda6b8106dc7",

expiration\_date: new Date(2014, 4)

}

]

}

应用1: 查询某地区的用户，使用邮政编码进行查询

>db.user.find({ “address.zip”:{“$lt”:12334,”gt”:11123}})

应用2: 查询买过某商品的用户，并显示用户的详细信息

user\_ids = db.db.order.find(

{

'line\_items.sku': "9092",

purchase\_date: {'$gt': new Date(2009, 0, 1)}

},

{user\_id: 1, \_id: 0}

).toArray().map(

function(doc) {

return doc['\_id']

}

)

users = db.users.find({\_id: {$in:user\_ids}})

问题： 查询0-30的用户

问题: 返回单条购物超过97元的订单

问题： 返回颜色不是黑色和蓝色的商品

问题： 返回标签式 garden和3c的产品

问题： 返回制造商是ACM，并标签不是 garden的商品

问题: 返回last\_name 不以B为开头的用户

问题： 返回颜色为蓝色和绿色的商品

问题: 返回颜色为蓝色，或制造商是ACM的商品

问题： 返回属性中带有颜色的商品

问题： 返回制造商id为432的商品

问题： 返回标签为soil的商品

问题： 返回第一个标签为0的商品

问题： 返回第一个地址中state为NY的用户

问题： 返回地址中state为NY的用户

问题: 返回地址.name为home，并在state为NY的用户

问题： 返回有三个地址的用户

问题： 返回用户ID对3取余余1的用户

问题： 返回sku大于1000的商品，并仅仅显示商品名

问题：商品价格倒序

问题：用户按注册日期倒序，并掠过100，取10条记录

## 三．mongoDB分布式集群

### 分片

#### 分片的原因:

* 由于海量数据的原因，单台服务器不能满足数据存储的需求，包括硬盘和内存
* 数据库读写负载的增加，单台服务器不能满足读/写并发的需求，通过分担相应的负载到不同的服务器上，来满足需求。 尤其是写负载的情况
* 海量数据的数据提取和汇总的瓶颈
* 最终需要通过分片，满足海量数据，高并发写的需求
* 最终核心：把单中心节点的数据和计算，分散到不同分布式节点，每个节点仅仅对路由中心负责，而客户端仅仅知道路由中心

#### 分片实现方式（水平分区）

* 手动方式(客户端程序控制)
* 中间键方式: mysql+amoba
* 数据系统内置的分片机制，mysql+cluster,mongodb内置分片

程序方式:



优点: 完全控制，能够自己来定义写入和读取的规则

缺点:

* 程序复杂性提高
* 分布式计算，并进行汇总的效率不易进行优化
* 算法比较固定，如取余，分段，或业务逻辑分片
* 难于扩展，和删除节点，由于存在数据迁移
* 调整负载和数据分布不容易
* 维护工作繁杂，难于热扩展
* 分片对于客户端是不透明，并且是紧耦合的



优点:

* 分片对客户端透明
* 采用配置方式来设定分片方式
* 分片策略可定制
* 减少客户端的开发工作

缺点：

* 性能损失
* 节点扩展，数据迁移问题
* 如果分布算法不平衡,存在节点负载不均衡的问题

#### 何时进行分片

* 开始时不进行分片，只有当需要分片时才采取分片策略
* 单机内存不足
* Mongod已经不能满足高并发写需求
* 大数据量读写

#### Mong分片的目的

#### 分片的步骤

* **步骤1：建立配置服务器**

>./mongod –dbpath=/usr/local/mongodb/db –port=2000 –fork

说明:

* 配置服务器负责储mongos的相应配置，包括分片规则和分片数据对应关系
* 如同启动普通服务器
* **步骤2：建立mongos服务**

**>**./mongos –port=3000 –configdb localhost:2000 –logpath=/usr/local/mongodb/logs/mongs.log –fork

* **步骤3：建立相应分片节点**

**节点1**

> ./mongod --port=1000 --dbpath=/usr/local/mongodb/db --logpath=/usr/local/mongodb/logs/mongodb.log –fork

节点2

> ./mongod --port=1000 --dbpath=/usr/local/mongodb/db --logpath=/usr/local/mongodb/logs/mongodb.log –fork