

## MongoDB 数据库简单介绍（一）

Mongo 是一个高性能，开源，无模式的文档型数据库，它在许多场景下可用于替代传统的关系型数据库或键/值存储方式。Mongo 使用 C++ 开发，提供了以下功能：

1. 面向集合的存储：适合存储对象及 JSON 形式的数据。
2. 动态查询：Mongo 支持丰富的查询表达式。查询指令使用 JSON 形式的标记，可轻易查询文档中内嵌的对象及数组。
3. 完整的索引支持：包括文档内嵌对象及数组。Mongo 的查询优化器会分析查询表达式，并生成一个高效的查询计划。
4. 查询监视：Mongo 包含一个监视工具用于分析数据库操作的性能。
5. 复制及自动故障转移：Mongo 数据库支持服务器之间的数据复制，支持主-从模式及服务器之间的相互复制。复制的主要目标是提供冗余及自动故障转移。
6. 高效的传统存储方式：支持二进制数据及大型对象（如照片或图片）。
7. 自动分片以支持云级别的伸缩性（处于早期 alpha 阶段）：自动分片功能支持水平的数据库集群，可动态添加额外的机器。

MongoDB 的主要目标是在键/值存储方式（提供了高性能和高度伸缩性）以及传统的 RDBMS 系统（丰富的功能）架起一座桥梁，集两者的优势于一身。根据官方网站的描述，Mongo 适合用于以下场景：

1. 网站数据：Mongo 非常适合实时的插入，更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。
2. 缓存：由于性能很高，Mongo 也适合作为信息基础设施的缓存层。在系统重启之后，由 Mongo 搭建的持久化缓存层可以避免下层的数据源过载。
3. 大尺寸，低价值的数据：使用传统的关系型数据库存储一些数据时可能会比较昂贵，在此之前，很多时候程序员往往会选择传统的文件进行存储。
4. 高伸缩性的场景：Mongo 非常适合由数十或数百台服务器组成的数据库。Mongo 的路线图中已经包含对 MapReduce 引擎的内置支持。
5. 用于对象及 JSON 数据的存储：Mongo 的 BSON 数据格式非常适合文档化格式的存储及查询。

自然，MongoDB 的使用也会有一些限制，例如它不适合：

1. 高度事务性的系统：例如银行或会计系统。传统的关系型数据库目前还是更适用于需要大量原子性复杂事务的应用程序。
2. 传统的商业智能应用：针对特定问题的 BI 数据库会对产生高度优化的查询方式。对于此类应用，数据仓库可能是更合适的选择。
3. 需要 SQL 的问题

MongoDB 支持 OS X、Linux 及 Windows 等操作系统，并提供了 Python, PHP, Ruby, Java, C, C#, Javascript, Perl 及 C++ 语言的驱动程序，社区中也提供了对 Erlang 及 .NET 等平台的驱动程序

MongoDB 下载:

<http://www.mongodb.org/display/DOCS/Downloads>  
<http://downloads.mongodb.org/lin...686-v1.2-latest.tgz>

**MongoDB 安装:**

```
[falcon@www.fwphp.cn ~]$ tar xvzf mongodb-linux-i686-v1.2-latest.tgz
[falcon@www.fwphp.cn ~]$ mv mongodb-linux-i686-v1.2-latest mongodb
[falcon@www.fwphp.cn ~]$ cd mongodb
[falcon@www.fwphp.cn ~]$ ls
mongo  mongodump  mongofiles  mongorestore  mongosniff
mongod  mongoexport  mongoimport  mongos
```

只需要解压到相关目录即可，不需要编译安装

MongoDB 启动与关闭:

mongoDB 的服务器端程序为 mongod

```
[falcon@www.fwphp.cn ~]$ bin/mongod --help
```

```
** NOTE: when using MongoDB 32 bit, you are limited to about 2 gigabytes of data
**       see http://blog.mongodb.org/post/137788967/32-bit-limitations for more
```

Allowed options:

General options:

-h [ --help ]	show this usage information
--version	show version information
-f [ --config ] arg	configuration file specifying additional options
--port arg	specify port number
--bind_ip arg	local ip address to bind listener - all local ips bound by default
-v [ --verbose ]	be more verbose (include multiple times for more verbosity e.g. -vvvv)
--dbpath arg (=data/db/)	directory for datafiles 指定数据存放目录
--quiet	quieter output 静默模式
--logpath arg	file to send all output to instead of stdout 指定日志存放目录
--logappend	append to logpath instead of over-writing 指定日志是以追加还是以覆盖的方式写入日志文件
--fork	fork server process 以创建子进程的方式运行
--cpu	periodically show cpu and iowait utilization 周期性的显示 cpu 和 io 的使用情况
--noauth	run without security 无认证模式运行
--auth	run with security 认证模式运行

--objcheck	inspect client data for validity on receipt 检查客
客户端输入数据的有效性检查	
--quota	enable db quota management 开始数据库配额的管理
--quotaFiles arg	number of files allow per db, requires --quota 规定每个数据库允许的文件数
--appsrpath arg	root directory for the babble app server
--nocursors	diagnostic/debugging option 调试诊断选项
--nohints	ignore query hints 忽略查询命中率
--nohttpinterface	disable http interface 关闭 http 接口, 默认是 28017
--noscripting	disable scripting engine 关闭脚本引擎
--noprealloc	disable data file preallocation 关闭数据库文件大小预分配
--smallfiles	use a smaller default file size 使用较小的默认文件大小
大小	
--nssize arg (=16)	.ns file size (in MB) for new databases 新数据库 ns 文件的默认大小
--diaglog arg	0=off 1=W 2=R 3=both 7=W+some reads 提供的方式, 是只读, 只写, 还是读写都行, 还是主要写+部分的读模式
--sysinfo	print some diagnostic system information 打印系统诊断信息
--upgrade	upgrade db if needed 如果需要就更新数据库
--repair	run repair on all dbs 修复所有的数据库
--notablescan	do not allow table scans 不运行表扫描
--syncdelay arg (=60)	seconds between disk syncs (0 for never) 系统同步刷新磁盘的时间, 默认是 60s
Replication options:	
--master	master mode 主复制模式
--slave	slave mode 从复制模式
--source arg	when slave: specify master as <server:port> 当为从时, 指定主的地址和端口
--only arg	when slave: specify a single database to replicate 当为从时, 指定需要从主复制的单一库
--pairwith arg	address of server to pair with
--arbiter arg	address of arbiter server 仲裁服务器, 在主主中和 pair 中用到
--autoresync	automatically resync if slave data is stale 自动同步从的数据
--oplogSize arg	size limit (in MB) for op log 指定操作日志的大小
--opIdMem arg	size limit (in bytes) for in memory storage of op ids 指定存储操作日志的内存大小
Sharding options:	
--configsvr	declare this is a config db of a cluster 指定 shard 中的

## 配置服务器

```
--shardsvr          declare this is a shard db of a cluster 指定 shard 服务
器
```

在启动 mongoDB 之前，我们必须新建一个存放 mongoDB 数据和日志的目录

```
[falcon@www.fwphp.cn ~]$ mkdir m_data m_log
```

到此 mongodb 启动的准备工作做完了，现在我们启动 mongodb

```
[falcon@www.fwphp.cn ~/mongodb]$ bin/mongod --dbpath=/home/falcon/m_data
--logpath=/home/falcon/m_log --logappend &
```

现在 mongodb 启动了，检查是否启动的方法：

```
[falcon@www.fwphp.cn ~/mongodb]$ ps -ef|grep mongod
falcon    2533  2271  0 08:21 pts/0    00:00:00 bin/mongod
--dbpath=/home/falcon/m_data --logpath=/home/falcon/m_log --logappend
falcon    2541  2271  0 08:22 pts/0    00:00:00 grep mongod
```

查看 mongodb 的端口是否启动，默认是 28017，在启动服务器时，可以通过--port 来指定

```
[falcon@www.fwphp.cn ~/mongodb]$ netstat -an -t|grep 28017
```

```
tcp        0      0
0.0.0.0:28017          0.0.0.0:*           LISTEN
```

到此，证明 mongoDB 已经启动完成

关闭的方法很简单：

Killall mongod 或者是 kill [pid]

使用方法：

利用客户端程序 mongo 登录 mongoDB

[falcon@www.fwphp.cn ~/mongodb]\$ bin/mongo	
MongoDB shell version: 1.2.4-	
url: test	显示数据库名
connecting to: test	显示当前数据库中的集合集
type "help" for help	显示当前数据库的用户
> help	显示最后系统用时大于 1ms 的系统概要
HELP	切换到数据库
Show dbs	help on DB methods
show collections	help on collection methods
show users	list objects in collection foo
show profile	
use <db name>	
db.help()	
db.foo.help()	
db.foo.find()	

```

db.foo.find( { a : 1 } )      list objects in foo where a == 1
it                           result of the last line evaluated; use to
further iterate
> show dbs    默认情况下有 2 数据库
admin
local
> use admin           切换到 admin 数据库
switched to db admin
> show collections    显示 admin 数据库下面的集合集
system.indexes

```

下面我们来简单的新建集合集，插入、更新、查询数据，体验 mongodb 带给我们不一样的生活

新建数据库的方法是在

新建集合集：

```

> db.createCollection("user");
{ "ok" : 1 }
> show collections
system.indexes
user
>

```

查入数据：

```

> db.user.insert({uid:1,username:"Falcon.C",age:25});
> db.user.insert({uid:2,username:"aabc",age:24});

```

查询数据：

```

> db.user.find();
{ "_id" : ObjectId("4b81e74c1f0fd3b9545cba43"), "uid" : 1, "username" : "Falcon.C",
"age" : 25 }
{ "_id" : ObjectId("4b81e74d1f0fd3b9545cba44"), "uid" : 2, "username" : "aabc",
"age" : 24 }

```

查询数据的方式很丰富，有类似于 SQL 的条件查询，将会在以后的文档中详细介绍

如：我想查询 uid 为 1 的用户信息

```

> db.user.find({uid:1});
{ "_id" : ObjectId("4b81e74c1f0fd3b9545cba43"), "uid" : 1, "username" : "Falcon.C",
"age" : 25 }

```

等等，丰富的查询还有 limit , sort , findOne, distinct 等

更新数据：

```
> db.user.update({uid:1}, {$set:{age:26}})
> db.user.find();
{ "_id" : ObjectId("4b81e76f1f0fd3b9545cba45"), "uid" : 1, "username" : "Falcon.C",
"age" : 26 }
{ "_id" : ObjectId("4b81e7701f0fd3b9545cba46"), "uid" : 2, "username" : "aabc",
"age" : 24 }
> db.user.update({uid:1}, {$inc:{age:-1}})
> db.user.find();
{ "_id" : ObjectId("4b81e76f1f0fd3b9545cba45"), "uid" : 1, "username" : "Falcon.C",
"age" : 25 }
{ "_id" : ObjectId("4b81e7701f0fd3b9545cba46"), "uid" : 2, "username" : "aabc",
"age" : 24 }
>
```

除了以上的 2 种用法，更新的条件还有\$unset、\$push、\$pushAll、\$pop、\$pull、\$pullAll

以上就是 MongoDB 简单的使用介绍，在以后的文档中将会详细的介绍 mongoDB 非常酷的 CURD 方法， mongoDB 的 Replication 及分布式

开发文档: <http://www.mongodb.org/display/DOCS/Developer+Zone>

管理文档: <http://www.mongodb.org/display/DOCS/Admin+Zone>

下载地址: <http://www.mongodb.org/display/DOCS/Downloads>

## MongoDB 主从复制介绍（二）

MongoDB 的主从复制其实很简单，就是在运行主的服务器上开启 mongod 进程时，加入参数 --master 即可，在运行从的服务器上开启 mongod 进程时，加入--slave 和 --source 指定主即可，这样，在主数据库更新时，数据被复制到从数据库中

(这里日志文件和访问数据时授权用户暂时不考虑)

下面我在单台服务器上开启 2deamon 来模拟 2 台服务器进行主从复制：

```
$ mkdir m_master m_slave
$mongodb/bin/mongod --port 28018 --dbpath ~/m_master --master &
$mongodb/bin/mongod --port 28019 --dbpath
~/m_slave --slave --source localhost:28018 &
```

这样主从服务器都已经启动了，可以利用 netstat -an -t 查看 28018、28019 端口是否开放

登录主服务器：

```
$ mongodb/bin/mongo --port 28018
MongoDB shell version: 1.2.4-
```

```
url: test
connecting to: 127.0.0.1:28018/test
type "help" for help
> show dbs
admin
local
test
> use test
switched to db test
> show collections
```

这里主上的 test 数据什么表都没有，为空，查看从服务器同样也是这样

```
$ mongodb/bin/mongo --port 28019
MongoDB shell version: 1.2.4-
url: test
connecting to: 127.0.0.1:28019/test
type "help" for help
> show dbs
admin
local
test
> use test
switched to db test
> show collections
```

那么现在我们来验证主从数据是否会像想象的那样同步呢？

我们在主上新建表 user

```
> db
test
>db.createCollection("user");
> show collections
system.indexes
user
>
```

表 user 已经存在了，而且 test 库中还多了一个 system.indexes 用来存放索引的表

到从服务器上查看 test 库：

```
> db
test
> show collections
system.indexes
User
```

```
> db.user.find();  
>
```

从服务器的 test 库中 user 表已经存在，同时我还查了一下 user 表为空

现在我们再来测试一下，向主服务器 test 库的 user 表中插入一条数据

```
> show collections  
system.indexes  
user  
> db.user.insert({uid:1, name:"Falcon.C", age:25});  
> db.user.find();  
{ "_id" : ObjectId("4b8226a997521a578b7aea38"), "uid" : 1, "name" : "Falcon.C",  
"age" : 25 }  
>
```

这时我们查看从服务器的 test 库 user 表时会多出一条记录来：

```
> db.user.find();  
{ "_id" : ObjectId("4b8226a997521a578b7aea38"), "uid" : 1, "name" : "Falcon.C",  
"age" : 25 }  
>
```

MongoDB 还有 Replica Pairs 和 Master – Master

参考地址：<http://www.mongodb.org/display/DOCS/Master+Slave>

### MongoDB 主主复制介绍（三）

MongoDB 一般情况下都可以支持主主复制，但是在大部分情况下官方不推荐使用

运行的 master – master 的准备工作是：

新建存放数据库文件的路径

```
$mkdir mongodata/mm_28050 mongodata/mm_28051
```

运行 mongodb 数据库，一个端口为： 28050，一个为： 28051

```
$ mongoDB/bin/mongod --port 28050 --dbpath ~/mongodata/mm_28050 --master --slave  
--source localhost:28051 > /dev/null &  
$ mongoDB/bin/mongod --port 28051 --dbpath ~/mongodata/mm_28051 --master --slave  
--source localhost:28050 > /dev/null &
```

可以通过 ps -ef | grep mongod 或 netstat -an -t 来检查是否运行功能

测试 master - master 模式：

```
$ mongodb/bin/mongo --port 28050
MongoDB shell version: 1.2.4-
url: test
connecting to: 127.0.0.1:28050/test
type "help" for help
> show dbs
admin
local
> db
test
> db.user.insert({_id:1,username:"Falcon.C",age:25,sex:"M"});
> db.user.find();
{ "_id" : 1, "username" : "Falcon.C", "age" : 25, "sex" : "M" }
> db.user.find(); //在 28051 端口插入数据后，再来查询，看数据是否同步
{ "_id" : 1, "username" : "Falcon.C", "age" : 25, "sex" : "M" }
{ "_id" : 2, "username" : "NetOne", "age" : 24, "sex" : "F" }
>
```

```
$ mongodb/bin/mongo --port 28051
MongoDB shell version: 1.2.4-
url: test
connecting to: 127.0.0.1:28051/test
type "help" for help
> db
test
> show collections          端口 28050 已经新建了一个 user 表并插入了一条数据，这里
多出 2 表
system.indexes
user
> db.user.find();           //查询表 user 发现数据已经同步
{ "_id" : 1, "username" : "Falcon.C", "age" : 25, "sex" : "M" }
> db.user.insert({_id:2,username:"NetOne",age:24,sex:"F"});在此插入数据看数据是
否双向同步
> db.user.find();
{ "_id" : 1, "username" : "Falcon.C", "age" : 25, "sex" : "M" }
{ "_id" : 2, "username" : "NetOne", "age" : 24, "sex" : "F" }
>
```

通过以上开启两终端分别连接到 28050、28051 端口，分别插入测试数据发现，一切正常，正如我们所想的那样实现数据的双向同步

参考信息：<http://www.mongodb.org/display/DOCS/Master+Master+Replication>

#### MongoDB 的使用技巧（四）

如果想查看当前连接在哪个数据库下面，可以直接输入 db

```
> db  
Admin
```

想切换到 test 数据库下面

```
> use test  
switched to db test  
> db  
Test
```

想查看 test 下有哪些表或者叫 collection，可以输入

```
> show collections  
system.indexes  
user
```

想知道 mongodb 支持哪些命令，可以直接输入 help

```
> help  
HELP  
    show dbs                      show database names  
    show collections               show collections in current database  
    show users                     show users in current database  
    show profile                   show most recent system.profile entries with  
time >= 1ms  
    use <db name>                 set current database to <db name>  
    db.help()                      help on DB methods  
    db.foo.help()                  help on collection methods  
    db.foo.find()                  list objects in collection foo  
    db.foo.find( { a : 1 } )        list objects in foo where a == 1  
    it                            result of the last line evaluated; use to  
further iterate
```

如果想知道当前数据库支持哪些方法：

```
> db.help();  
DB methods:
```

```

db.addUser(username, password) 添加数据库授权用户
db.auth(username, password)           访问认证
db.cloneDatabase(fromhost) 克隆数据库
db.commandHelp(name) returns the help for the command
db.copyDatabase(fromdb, todb, fromhost) 复制数据库
db.createCollection(name, { size : ..., capped : ..., max : ... } ) 创建表
db.currentOp() displays the current operation in the db
db.dropDatabase()      删除当前数据库
db.eval(func, args) run code server-side
db.getCollection(cname) same as db['cname'] or db.cname
db.getCollectionNames()       获取当前数据库的表名
db.getLastErrorMessage() - just returns the err msg string
db.getLastErrorMessageObj() - return full status object
db.getMongo() get the server connection object
db.getMongo().setSlaveOk() allow this connection to read from the
nonmaster member of a replica pair
db.getName()
db.getLastError()
db.getProfilingLevel()
db.getReplicationInfo()
db.getSiblingDB(name) get the db at the same server as this one
db.killOp() kills the current operation in the db
db.printCollectionStats() 打印各表的状态信息
db.printReplicationInfo()      打印主数据库的复制状态信息
db.printSlaveReplicationInfo() 打印从数据库的复制状态信息
db.printShardingStatus()       打印分片状态信息
db.removeUser(username) 删除数据库用户
db.repairDatabase() 修复数据库
db.resetError()
db.runCommand(cmdObj) run a database command. if cmdObj is a string,
turns it into { cmdObj : 1 }
db.setProfilingLevel(level) 0=off 1=slow 2=all
db.shutdownServer()
db.version() current version of the server
  
```

如果想知道当前数据库下的表或者表 collection 支持哪些方法，可以使用一下命令如：

```

> db.user.help(); user 为表名
DBCollection help
    db.foo.count()          统计表的行数
    db.foo.dataSize()        统计表数据的大小
    db.foo.distinct( key ) - eg. db.foo.distinct('x')      按照
给定的条件除重
    db.foo.drop() drop the collection 删除表
  
```

```

db.foo.dropIndex(name)    删除指定索引
db.foo.dropIndexes()    删除所有索引
db.foo.ensureIndex(keypattern, options) - options should be an object with
these possible fields: name, unique, dropDups 增加索引
    db.foo.find( [query] , [fields]) - first parameter is an optional query
filter. second parameter is optional set of fields to return. 根据条件查找数据
        e.g. db.foo.find( { x : 77 } , { name :
1 , x : 1 } )

db.foo.find(...).count()
db.foo.find(...).limit(n) 根据条件查找数据并返回指定记录数
db.foo.find(...).skip(n)
db.foo.find(...).sort(...) 查找排序
db.foo.findOne([query]) 根据条件查询只查询一条数据
db.foo.getDB() get DB object associated with collection 返回表所属的库
db.foo.getIndexes() 显示表的所有索引
db.foo.group( { key : ..., initial: ... , reduce : ... [, cond: ... ] } ) 根
据条件分组
db.foo.mapReduce( mapFunction , reduceFunction , <optional params> )
db.foo.remove(query) 根据条件删除数据
db.foo.renameCollection( newName ) renames the collection 重命名表
db.foo.save(obj) 保存数据
db.foo.stats() 查看表的状态
db.foo.storageSize() - includes free space allocated to this collection
查询分配到表空间大小
db.foo.totalIndexSize() - size in bytes of all the indexes 查询所有索引
的大小
db.foo.totalSize() - storage allocated for all data and indexes 查询表
的总大小
db.foo.update(query, object[, upsert_bool]) 根据条件更新数据
db.foo.validate() - SLOW 验证表的详细信息
db.foo.getShardVersion() - only for use with sharding
  
```

## Mongodb 的备份工具 mongodump

如果想备份数据库 test 如:

```
[falcon@www.fwphp.cn ~]/mongodb/bin]$ ./mongodump --help
options:
--help          produce help message
-h [ --host ] arg      mongo host to connect to
-d [ --db ] arg       database to use
-c [ --collection ] arg collection to use (some commands)
-u [ --username ] arg   username
-p [ --password ] arg   password
--dbpath arg        directly access mongod data files in this path,
```

```

        instead of connecting to a mongod instance
-v [ --verbose ]      be more verbose (include multiple times for more
                      verbosity e.g. -vvvvv)
-o [ --out ] arg (=dump) output directory
[falcon@www. fwp.php.cn ~]/mongodb/bin]$ [color=Blue]. ./mongodump -d test -o
test/[/color]
connected to: 127.0.0.1
DATABASE: test          to      test/test
            test.user to test/test/user.bson
            100000 objects
            test.system.indexes to test/test/system.indexes.bson
            1 objects
[falcon@www. fwp.php.cn ~]/mongodb/bin]$ ls
2      mongo  mongodump  mongofiles  mongorestore  mongosniff
dump  mongod  mongoexport  mongoimport  mongos      test

```

### MongoDB 的数据恢复工具 mongorestore

#### 查看 test 库中的表

```

> show collections
system.indexes
User

```

#### 删除 user 表

```

> db.user.drop();
True

> show collections
System.indexes

```

#### 现在利用 mongorestore 表恢复刚才利用 mongodump 备份的数据

```

[falcon@www. fwp.php.cn ~]/mongodb/bin]$ ./mongorestore --help
usage: ./mongorestore [options] [directory or filename to restore from]
options:
  --help                  produce help message
  -h [ --host ] arg       mongo host to connect to
  -d [ --db ] arg         database to use
  -c [ --collection ] arg collection to use (some commands)
  -u [ --username ] arg   username
  -p [ --password ] arg   password
  --dbpath arg            directly access mongod data files in this path,
                        instead of connecting to a mongod instance
  -v [ --verbose ]         be more verbose (include multiple times for more
                        verbosity e.g. -vvvvv)

```

```
[falcon@www.fwphp.cn ~]/mongodb/bin]$ ./mongorestore -d test -c user
test/test/user.bson
connected to: 127.0.0.1
test/test/user.bson
      going into namespace [test.user]

100000 objects
```

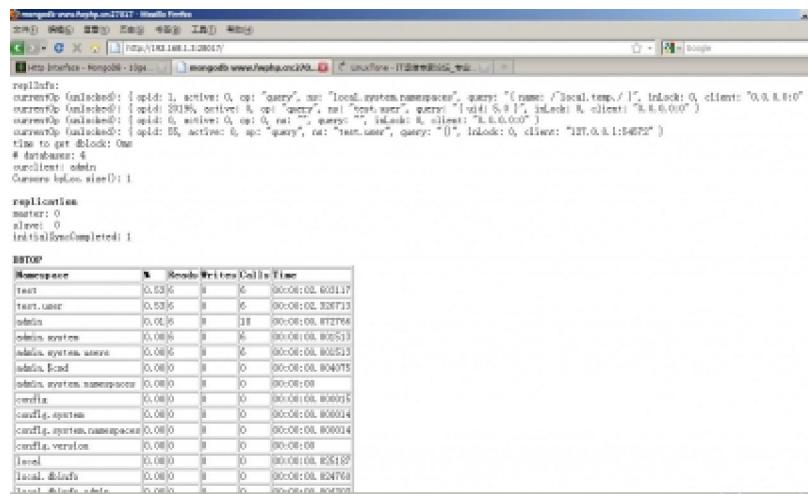
User 表中的 10w 条记录已经恢复

```
> show collections
system.indexes
user
> db.user.find();
{ "_id" : ObjectId("4b9c8db08ead0e3347000000"), "uid" : 1, "username" :
  "Falcon.C-1" }
{ "_id" : ObjectId("4b9c8db08ead0e3347010000"), "uid" : 2, "username" :
  "Falcon.C-2" }
{ "_id" : ObjectId("4b9c8db08ead0e3347020000"), "uid" : 3, "username" :
  "Falcon.C-3" }
{ "_id" : ObjectId("4b9c8db08ead0e3347030000"), "uid" : 4, "username" :
  "Falcon.C-4" }
{ "_id" : ObjectId("4b9c8db08ead0e3347040000"), "uid" : 5, "username" :
  "Falcon.C-5" }

has more
```

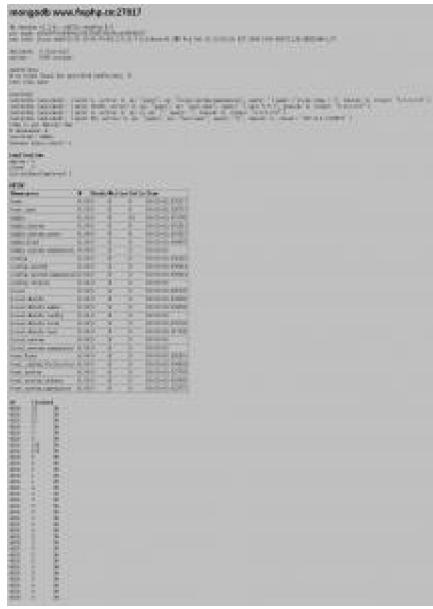
mongodb 还提供了 HTTP 查看运行状态及 restfull 的接口

默认的访问端口是 28017



The screenshot shows the MongoDB 28017 Health Profile page. It includes a log section with detailed information about operations like insert, update, and query, and a BIRTOP section showing the distribution of reads, writes, and calls over time across different namespaces.

Namespace	R	W	Calls	Time
test	0.00	0	0	00:00:02.62117
test.user	0.00	0	0	00:00:02.35873
admin	0.00	0	18	00:00:08.872768
admin.system	0.00	0	6	00:00:08.867813
admin.system.users	0.00	0	6	00:00:08.86513
admin.land	0.00	0	0	00:00:08.864973
admin.system.namespaces	0.00	0	0	00:00:08.864973
config	0.00	0	0	00:00:08.860015
config.system	0.00	0	0	00:00:08.860014
config.system.namespaces	0.00	0	0	00:00:08.860014
config.version	0.00	0	0	00:00:08.860014
local	0.00	0	0	00:00:08.86169
local.databases	0.00	0	0	00:00:08.864763
local.diagnostics	0.00	0	0	00:00:08.864763



## rest 的访问接口

```
{
  "offset": 0,
  "rows": [
    {
      "_id": "4b9c8db08ead0e3347000000",
      "uid": 1,
      "username": "Falcon.C-1"
    },
    {
      "_id": "4b9c8db08ead0e3347010000",
      "uid": 2,
      "username": "Falcon.C-2"
    },
    {
      "_id": "4b9c8db08ead0e3347020000",
      "uid": 3,
      "username": "Falcon.C-3"
    },
    {
      "_id": "4b9c8db08ead0e3347030000",
      "uid": 4,
      "username": "Falcon.C-4"
    },
    {
      "_id": "4b9c8db08ead0e3347040000",
      "uid": 5,
      "username": "Falcon.C-5"
    },
    {
      "_id": "4b9c8db08ead0e3347050000",
      "uid": 6,
      "username": "Falcon.C-6"
    },
    {
      "_id": "4b9c8db08ead0e3347060000",
      "uid": 7,
      "username": "Falcon.C-7"
    },
    {
      "_id": "4b9c8db08ead0e3347070000",
      "uid": 8,
      "username": "Falcon.C-8"
    },
    {
      "_id": "4b9c8db08ead0e3347080000",
      "uid": 9,
      "username": "Falcon.C-9"
    },
    {
      "_id": "4b9c8db08ead0e3347090000",
      "uid": 10,
      "username": "Falcon.C-10"
    }
  ],
  "total_rows": 10,
  "query": {},
  "rallies": 0
}
```

参考地址: <http://www.mongodb.org/display/DOCS/Http+Interface>

## MongoDB 的 sharding 功能 (五)

MongoDB 的 auto-sharding 功能是指 mongodb 通过 mongos 自动建立一个水平扩展的数据库集群系统，将数据库分表存储在 sharding 的各个节点上。

一个 mongodb 集群包括一些 shards (包括一些 mongod 进程)，mongos 路由进程，一个或多个 config 服务器

### Shards

每一个 shard 包括一个或多个服务和存储数据的 mongod 进程 (mongod 是 MongoDB 数据的核心进程)

典型的每个 shard 开启多个服务来提高服务的可用性。这些服务/mongod 进程在 shard 中组成一个复制集

### Chunks

Chunk 是一个来自特殊集合中的一个数据范围，(collection, minKey, maxKey) 描述一个 chunk，它介于 minKey 和 maxKey 范围之间。

例如 chunks 的 maxsize 大小是 100M，如果一个文件达到或超过这个范围时，会被切分到 2 个新的 chunks 中。当一个 shard 的数据过量时，chunks 将会被迁移到其他的 shards 上。

同样，chunks 也可以迁移到其他的 shards 上

### Config Servers

Config 服务器存储着集群的 metadata 信息，包括每个服务器，每个 shard 的基本信息和 chunk 信息

Config 服务器主要存储的是 chunk 信息。每一个 config 服务器都复制了完整的 chunk 信息。

配置：（模拟 2 个 shard 服务和一个 config 服务）

Shard1: 27020

Shard2: 27021

Config: 27022

Mongos 启动时默认使用的 27017 端口

### 新建存放数据的目录

```
[falcon@www. fwphp. cn ~]/mongodata]$ mkdir 27020 27021 27022
```

```
[falcon@www. fwphp. cn ~]/mongodata]$ ls
```

```
27020 27021 27022
```

```
[falcon@www. fwphp. cn ~]/mongodb/bin]$ ./mongod --dbpath  
/home/falcon/mongodata/27020 --port 27020 > /home/falcon/mongodata/27020.log &  
[falcon@www. fwphp. cn ~]/mongodb/bin]$ ./mongod --dbpath  
/home/falcon/mongodata/27021 --port 27021 > /home/falcon/mongodata/27021.log &  
[falcon@www. fwphp. cn ~]/mongodb/bin]$ ./mongod --dbpath  
/home/falcon/mongodata/27022 --port 27022 > /home/falcon/mongodata/27022.log &
```

启动 mongos 时，默认开启了 27017 端口

```
[falcon@www. fwphp. cn ~]/mongodb/bin]$ ./mongos --configdb localhost:27022 >  
/home/falcon/mongodata/config.log &
```

### 检查是否启动

```
[falcon@www. fwphp. cn ~]/mongodb/bin]$ ps -ef|grep mongo  
falcon 2612 1 0 20:15 ? 00:00:00 ./mongod --dbpath
```

```
/home/falcon/mongodata/27020 --port 27020
falcon 2619 1 0 20:15 ? 00:00:00 ./mongod --dbpath
/home/falcon/mongodata/27021 --port 27021
falcon 2625 1 0 20:15 ? 00:00:00 ./mongod --dbpath
/home/falcon/mongodata/27022 --port 27022
falcon 2658 1 0 20:15 ? 00:00:00 ./mongos --configdb
localhost:27022
falcon 2804 2772 0 20:31 pts/0 00:00:00 bin/mongo
falcon 2847 2812 0 20:55 pts/2 00:00:00 grep mongo
[falcon@www.fwphp.cn ~/mongodb/bin]$
```

```
[falcon@www.fwphp.cn ~/mongodb/bin]$ netstat -an -t
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign
Address State
tcp 0 0
0.0.0.0:10022 0.0.0.0:*
LISTEN
tcp 0 0
0.0.0.0:27017 0.0.0.0:*
LISTEN
tcp 0 0
0.0.0.0:587 0.0.0.0:*
LISTEN
tcp 0 0
0.0.0.0:27020 0.0.0.0:*
LISTEN
tcp 0 0
0.0.0.0:27021 0.0.0.0:*
LISTEN
tcp 0 0
0.0.0.0:27022 0.0.0.0:*
LISTEN
.....
tcp 0 0
0.0.0.0:28020 0.0.0.0:*
LISTEN
tcp 0 0
0.0.0.0:28021 0.0.0.0:*
LISTEN
tcp 0 0
0.0.0.0:28022 0.0.0.0:*
LISTEN
tcp 0 0
127.0.0.1:631 0.0.0.0:*
LISTEN
.....
```

看到以上信息证明 mongodb 启动完整，对于开启的 28020、28021、28022 是对于的 http 接口

```
[falcon@www.fwphp.cn ~/mongodb/bin]$ ./mongo 默认连接到 mongos 上
MongoDB shell version: 1.2.4-
url: test
connecting to: test
```

```
type "help" for help
> show dbs
admin
config
Local
```

### 加入 shard 节点

```
> use admin
switched to db admin

> db.runCommand( { addshard : "localhost:27020", allowLocal : true } )
{ "ok" : 1 , "added" : "localhost:27020" }
```

```
> db.runCommand( { addshard : "localhost:27021", allowLocal : true } )
{ "ok" : 1 , "added" : "localhost:27021" }
```

```
> db.runCommand({listshards:1});    查看 shard 节点列表
```

```
{
  "shards" : [
    {
      "_id" : ObjectId("4b9cd380c33000afad27718e"),
      "host" : "localhost:27020"
    },
    {
      "_id" : ObjectId("4b9cd381c33000afad27718f"),
      "host" : "localhost:27021"
    }
  ],
  "ok" : 1
}
```

### 新建自动切片的库 user001:

```
> config = connect("localhost:27022")
> config = config.getSiblingDB("config")
> user001=db.getSiblingDB("user001");
user001
> db.runCommand({enablesharding:"user001"})
{ "ok" : 1 }

> db.printShardingStatus();
--- Sharding Status ---
  sharding version: { "_id" : ObjectId("4b9cd354c33000afad27718d"), "version" :
2 }
  shards:
```

```
{
  "_id": ObjectId("4b9cd380c33000afad27718e"), "host": "localhost:27020"
  {"_id": ObjectId("4b9cd381c33000afad27718f"), "host": "localhost:27021"}
  databases:
    { "name": "admin", "partitioned": false, "primary": "localhost:27022",
    "_id": ObjectId("4b9cd3776693dcfa468dec13") }
    { "name": "user001", "partitioned": true, "primary": "localhost:27021",
    "_id": ObjectId("4b9cde866693dcfa468dec17") }
      my chunks
```

我们来在 user001 中新建表，插入数据

```
> use user001
switched to db user001
> db.createCollection("user_001")
{ "ok" : 1 }
> show collections
system.indexes
user_001
> db.user_001.insert({uid:1,username:"Falcon.C",sex:"男",age:25});
> db.user_001.find();
{ "_id": ObjectId("4b9ce1a6c84d7f20576c4df1"), "uid": 1, "username": "Falcon.C",
"sex": "男", "age": 25 }
```

我们来看看 user001 库被分配到了哪个 shard 上

```
[falcon@www.fwphp.cn ~]/mongodata]$ ls -R
.:
27020 27021 27022 mongos.log

./27020:
27020.log mongod.lock test.0 test.1 test.ns _tmp

./27020/_tmp:

./27021:
27021.log mongod.lock _tmp user.0 user001.0 user001.1 user001.ns user.1
user.ns

./27021/_tmp:

./27022:
27022.log config.0 config.ns mongod.lock mongos.log _tmp

./27022/_tmp:
[falcon@www.fwphp.cn ~]/mongodata]$
```

从以上的文件可以看出，user001 被分配到了 27021 的 shard 上了，但是通过 mongos 路由，我们并感觉不到是数据存放在哪个 shard 的 chunk 上

### Sharding 的管理命令

```
>
db.$cmd.findOne({isdbgrid:1});

{ "isdbgrid" : 1, "hostname" : "www.fwphp.cn", "ok" : 1 }
> db.$cmd.findOne({ismaster:1});
{ "ismaster" : 1, "msg" : "isdbgrid", "ok" : 1 }
> printShardingStatus(db.getSiblingDB("config"))
--- Sharding Status ---
  sharding version: { "_id" : ObjectId("4b9cd354c33000afad27718d"), "version" :
2 }

shards:
  { "_id" : ObjectId("4b9cd380c33000afad27718e"), "host" : "localhost:27020" }
  { "_id" : ObjectId("4b9cd381c33000afad27718f"), "host" : "localhost:27021" }

databases:
  { "name" : "admin", "partitioned" : false, "primary" : "localhost:27022",
"_id" : ObjectId("4b9cd3776693dcfa468dec13") }

      my chunks
  { "name" : "user001", "partitioned" : true, "primary" : "localhost:27021",
"_id" : ObjectId("4b9cde866693dcfa468dec17") }

      my chunks

> use admin
switched to db admin
> db.runCommand({netstat:1})
{ "configserver" : "localhost:27022", "isdbgrid" : 1, "ok" : 1 }
>
```

参考信息：<http://www.mongodb.org/display/DOCS/Sharding>

### MongoDB 数据库的索引操作（六）

Mongodb 数据库的索引操作很简单，只需要把作为条件的字段设置为索引即可

```
> use user
switched to db user
> show collections
system.indexes
u_info
u_setting
> db.system.indexes.find();   这是默认的索引（默认为_id 为索引）
{ "name" : "_id_", "ns" : "user.u_info", "key" : { "_id" :
```

```

ObjectId("00000000000000000000000000000000") } }
{ "name" : "_id_", "ns" : "user.u_setting", "key" : { "_id" :
ObjectId("00000000000000000000000000000000") } }
>
> db.u_info.insert({uid:1, name:"Falcon.C", address:"Beijing"});
> db.u_info.insert({uid:2, name:"sexMan", address:"Wuhan"});
> db.u_info.find();
{ "_id" : ObjectId("4b9cf280c84d7f20576c4df2"), "uid" : 1, "name" : "Falcon.C",
"address" : "Beijing" }
{ "_id" : ObjectId("4b9cf284c84d7f20576c4df3"), "uid" : 2, "name" : "sexMan",
"address" : "Wuhan" }

```

插入了 2 条记录，我们来把 uid 设置为索引字段：

```

> db.u_info.ensureIndex({uid:1});
> db.u_info.ensureIndex({name:1});
> db.system.indexes.find();
{ "name" : "_id_", "ns" : "user.u_info", "key" : { "_id" :
ObjectId("00000000000000000000000000000000") } }
{ "name" : "_id_", "ns" : "user.u_setting", "key" : { "_id" :
ObjectId("00000000000000000000000000000000") } }
{ "ns" : "user.u_info", "key" : { "uid" : 1 }, "name" : "uid_1" }
{ "ns" : "user.u_info", "key" : { "name" : 1 }, "name" : "name_1" }
>

```

这时我们看到多了刚才我们设置的那个字段，这样在查询的时候，如果查询条件有 uid 字段或 name 字段，则走索引来进行查询

有索引：

```

> db.u_info.find({name:"Falcon.C"});
{ "_id" : ObjectId("4b9cf280c84d7f20576c4df2"), "uid" : 1, "name" : "Falcon.C",
"address" : "Beijing" }
> db.u_info.find({name:"Falcon.C"}).explain();
{
    "cursor" : "BtreeCursor name_1",
    "startKey" : {
        "name" : "Falcon.C"
    },
    "endKey" : {
        "name" : "Falcon.C"
    },
    "nscanned" : 1,
    "n" : 1,
    "millis" : 0,
    "allPlans" : [

```

```
{
    "cursor" : "BtreeCursor name_1",
    "startKey" : {
        "name" : "Falcon.C"
    },
    "endKey" : {
        "name" : "Falcon.C"
    }
}
]
```

删除索引后：

```
> db.system.indexes.find();
{ "name" : "_id_", "ns" : "user.u_info", "key" : { "_id" :
ObjectId("00000000000000000000000000000000") } }
{ "name" : "_id_", "ns" : "user.u_setting", "key" : { "_id" :
ObjectId("00000000000000000000000000000000") } }
{ "ns" : "user.u_info", "key" : { "uid" : 1 }, "name" : "uid_1" }
{ "ns" : "user.u_info", "key" : { "name" : 1 }, "name" : "name_1" }
> db.u_info.dropIndex("name_1")
{ "nIndexesWas" : 3, "ok" : 1 }
> db.u_info.find({name:"Falcon.C"}).explain();
{
    "cursor" : "BasicCursor",
    "startKey" : {

    },
    "endKey" : {

    },
    "nscanned" : 2,
    "n" : 1,
    "millis" : 0,
    "allPlans" : [
        {
            "cursor" : "BasicCursor",
            "startKey" : {

            },
            "endKey" : {

            }
        }
    ]
}
```

```

        ]
}

> db.system.indexes.find();
{
  "name" : "_id_",
  "ns" : "user.u_info",
  "key" : { "_id" :
  ObjectId("00000000000000000000000000000000") }
}
{
  "name" : "_id_",
  "ns" : "user.u_setting",
  "key" : { "_id" :
  ObjectId("00000000000000000000000000000000") }
}
{
  "ns" : "user.u_info",
  "key" : { "uid" : 1 },
  "name" : "uid_1"
}

```

通过以上可以看出，查询的条件中有索引时，查询走 [BtreeCursor](#) 的索引，而没有索引时走 [BasicCursor](#)

通常需要索引的字段是：

1. 唯一键 `_id` 是默认被设置为索引的
2. 需要被查找的字段应该建立索引，比如在 `find()` 里面的字段
3. 需要被排序的字段应该建立索引。比如在 `sort()` 里面的字段

以上就是 MongoDB 的索引操作

## MongoDB 数据库的 MapReduce 简单操作（七）

MongoDB 也简单的实现了 MapReduce 的功能来提供分布式的数据查询服务，MapReduce 的分布是功能主要用在 Shard 上

```

db.runCommand(
  {
    mapreduce : <collection>,
    map : <mapfunction>,
    reduce : <reducefunction>
    [, query : <query filter object>]
    [, sort : <sort the query. useful for optimization>]
    [, limit : <number of objects to return from collection>]
    [, out : <output-collection name>]
    [, keeptemp: <true|false>]
    [, finalize : <finalizefunction>]
    [, scope : <object where fields go into javascript global scope >]
    [, verbose : true]
  }
);

```

下面是对 MapReduce 的简单测试

此例子来源于：<http://www.mongodb.org/display/DOCS/MapReduce>

```

>
db.things.insert({_id:1, tags:['dog','cat']});

```

```
> db.things.insert({_id:2,tags:['cat']});
> db.things.insert({_id:3,tags:['mouse','cat','dog']});
> db.things.insert({_id:4,tags:[]});
> m = function() {
...   this.tags.forEach(
...     function(z) {
...       emit(z, {count:1});
...     }
...   );
};
function () {
  this.tags.forEach(function (z) {emit(z, {count:1});});
}
> r=function(key,values) {
...   var total = 0;
...   for(var i=0;i<values.length;i++)
...     total += values[i].count;
...   return {count:total};
... };
function (key, values) {
  var total = 0;
  for (var i = 0; i < values.length; i++) {
    total += values[i].count;
  }
  return {count:total};
}
> res=db.things.mapReduce(m,r);
{
  "result": "tmp.mr.mapreduce_1268577545_1",
  "timeMillis": 25,
  "counts": {
    "input": 4,
    "emit": 6,
    "output": 3
  },
  "ok": 1,
  "ok": 1,
}
> res
{
  "result": "tmp.mr.mapreduce_1268577545_1",
  "timeMillis": 25,
  "counts": {
    "input": 4,
```

```
    "emit" : 6,
    "output" : 3
},
"ok" : 1,
"ok" : 1,
}

> db[res.result].find()
{ "_id" : "cat", "value" : { "count" : 3 } }
{ "_id" : "dog", "value" : { "count" : 2 } }
{ "_id" : "mouse", "value" : { "count" : 1 } }
> db[res.result].drop()
true
> db[res.result].find()
>
```