第四十一章：MongoDB 介绍与部署应用

**一、Mongodb简介；**

**二、Mongodb特点；**

**三、Mongo应用场景；**

**四、Mongodb存储结构；**

**五、Mongodb数据类型；**

**六、案例：搭建Mongodb数据库应用；**

**七、Mongodb中语句操作；**

**八、Mongodb数据库的导入导出、备份恢复、复制数据库；**

**九、Mongodb数据库的用户角色权限管理；**

**十、Mongodb连接php程序；**

**十一：自主学习：Mongodb如何占用内存，占用内存过高如何处理？**

**一、Mongodb简介；**

**概述：**MongoDB 是由C++语言编写的，是一个基于分布式文件存储的非关系型开源数据库系统。其优势在于可以存放海量数据，具备强大的查询功能，是一个独立的面向集合文档形式的；

**起源与发展：**2007年10月，MongoDB由10gen团队所发展。2009年2月首度推出。

**应用平台：**MonggoDB支持Unix、linux、windows等系统平台。

**二、Mongodb特点：**

**存储性：**

1.面向集合：Collection-Oriented，数据被分组存储在数据集中，被称为一个集合（Collection)。每个集合在数据库中都有一个唯一的标识名，并且可以包含无限数目的文档。集合的概念类似关系型数据库里的表，不同的是它不需要定义任何模式。

2.面向文档：存储在集合中的文档，被存储为键-值对的形式。键用于唯一标识一个文档，为字符串类型，而值则可以是各种复杂的文件类型。我们称这种存储形式为BSON（Binary Serialized Document Format）；

3.高效二进制数据存储：包括大型对象（如视频）。使用二进制格式存储，可以保存任何类型的数据对象；

**操作性：**

1.完全索引：可以在任意属性上建立索引，包含内部对象。以提高查询的速度；

2.强大的聚合工具：MongoDB 除了提供丰富的查询功能外，还提供强大的聚合工具，如count、group 等，支持使用MapReduce 完成复杂的聚合任务；

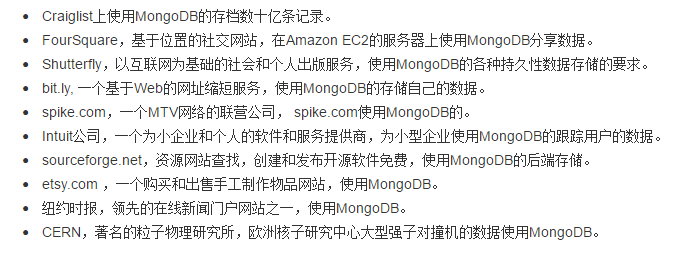
3.支持Perl、PHP、Java、C#、JavaScript、Ruby、C 和C++语言的驱动程序：MongoDB 提供了当前所有主流开发语言的数据库驱动包，开发人员使用任何一种主流开发语言都可以轻松编程，实现访问MongoDB 数据库；

**可用性：**

1.支持复制和数据恢复：MongoDB 支持主从复制机制，可以实现数据备份、故障恢复、读扩展等功能。而基于副本集的复制机制提供了自动故障恢复的功能，确保了集群数据不会丢失；

2.自动处理分片：MongoDB 支持集群自动切分数据，对数据进行分片可以使集群存储更多的数据，实现更大的负载，也能保证存储的负载均衡；

**三、Mongodb应用场景：**



**适用场景：**

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搭建的持久化缓存可以避免下层的数据源过载；

3.密度高、价值低的数据：使用传统的关系数据库存储一些数据时可能会比较贵，在此之前，很多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非常适合由数十或者数百台服务器组成的数据库。

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格式化的存储及查询；

**非适用场景：**

1.高度事物性的系统：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；

**四、Mongodb存储结构：**

**逻辑结构：**

文档（ document）：是 mongodb 核心概念也是逻辑存储的最小单元，相当于行；

集合（ collection）：多个文档组成集合，相当于表，但不同于表的是无固定结构；

数据库（ database）：多个集合组成数据库；

总结：

一个MongoDB 实例可以包含一组数据库，一个DataBase 可以包含一组Collection，一个集合可以包含一组Document。一个Document包含一组field（字段），每一个字段都是一个key/value pair；

**物理结构：**

1. .ns文件：每个表或索引对应一个命名空间，数据量增加，文件数量增多，存储了分配和正在使用的磁盘空间；

2. 数据文件：存放数据的实体，并且使用预分配空间机制；

注解：预分配空间机制：mongodb在当前数据文件的基础上，会预分配文件（用0进行填充），数据文件每次新生成的文件，大小是上个文件的两倍，这样可以保证数据文件既不浪费又保证了可用性；

3. 日志文件：系统日志文件、journal日志文件（用于mongodb崩溃恢复的保障）、oplog复制操作日志文件（相当于mysql的binary log文件）、慢查询日志：查询操作超出超时时间的慢动作；

注：在mongodb 3.0版本后，默认的存储引擎已经改变，成为WT（wiredTiger）存储引擎，无以前的.ns文件；

  wiredTiger引擎：3.0新增引擎，官方宣称在read、insert和复杂的update下具有更高的性能。可以支撑更高的读写负载和并发量。因为对于production环境，更多的CPU可以有效提升wireTiger的性能，因为它是的IO是多线程的。wiredTiger不像MMAPV1引擎那样尽可能的耗尽内存，它可以通过在配置文件中指定“cacheSizeGB”参数设定引擎使用的内存量；

    journal就是一个预写事务日志，来确保数据的持久性，wiredTiger每隔60秒（默认）或者待写入的数据达到2G时，mongodb将对journal文件提交一个checkpoint（检测点，将内存中的数据变更flush到磁盘中的数据文件中，并做一个标记点，表示此前的数据表示已经持久存储在了数据文件中，此后的数据变更存在于内存和journal日志）。对于write操作，首先被持久写入journal，然后在内存中保存变更数据，条件满足后提交一个新的检测点，即检测点之前的数据只是在journal中持久存储，但并没有在mongodb的数据文件中持久化，延迟持久化可以提升磁盘效率；

    为了确保数据的安全性，mongodb将所有的变更操作写入journal并间歇性的持久到磁盘上，对于实际数据文件将延迟写入，和wiredTiger一样journal也是用于数据恢复；



https://blog.csdn.net/xiaoql520/article/details/76973887

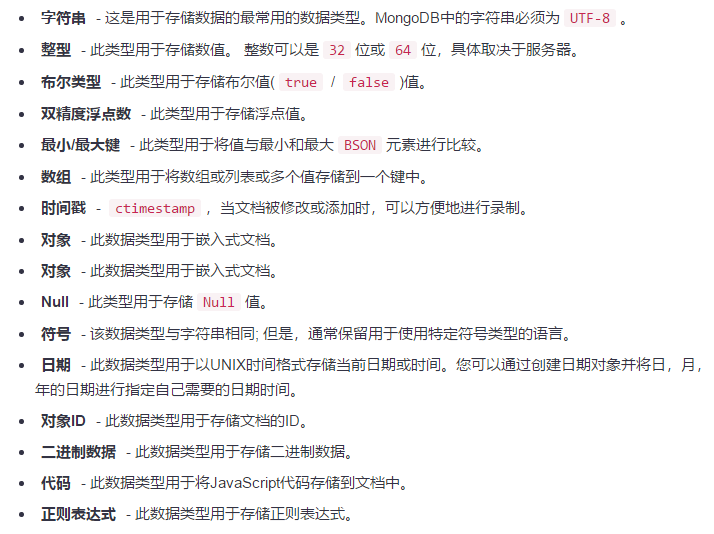
**内存映射存储引擎MMAP：**

1.mongodb会把数据文件映射到内存中，如果是读操作，内存中的数据起到缓存的作用，如果是写操作，内存还可以把随机的写操作转换成顺序的写操作，总之可以大幅度提升性能；

2.MongoDB并不干涉内存管理工作，而是把这些工作留给操作系统的虚拟内存管理器去处理，这样做的好处是简化了MongoDB的工作，但坏处是你没有方法很方便的控制MongoDB占多大内存，幸运的是虚拟内存管理器的存在让我们多数时候并不需要关心这个问题；

3.MongoDB的内存使用机制（持久化）让它在缓存重建方面更有优势，简而言之：如果重启进程，那么缓存依然有效；

**五、Mongodb数据类型；**



**六、案例：搭建Mongodb数据库应用；**

**案例环境：**

|  |  |  |  |
| --- | --- | --- | --- |
| 系统 | 主机名 | IP地址 | 所需软件 |
| Centos 7.4 | mongodb.linuxfan.cn | 192.168.100.101 | mongodb-linux-x86\_64-rhel70-3.6.3.tgz |

**案例步骤：**

* 准备系统环境；
* 下载解压mongodb软件包；
* 创建mongodb存储目录以及配置文件；
* 编写服务启动脚本；
* 测试访问mongodb数据库；
* **准备系统环境；**

[root@mongodb ~]# ulimit -n 25000 ##同一时间最多开启的文件数

[root@mongodb ~]# ulimit -u 25000 ##用户最多开启的程序数目

[root@mongodb ~]# echo 0 >/proc/sys/vm/zone\_reclaim\_mode ##设置内核参数，当某个节点内存不足，可以借用其他节点的内存

[root@mongodb ~]# sysctl -w vm.zone\_reclaim\_mode=0

[root@mongodb ~]# echo never >/sys/kernel/mm/transparent\_hugepage/enabled

[root@mongodb ~]# echo never >/sys/kernel/mm/transparent\_hugepage/defrag

* **下载解压mongodb软件包；**

[root@mongodb ~]# tar zxvf mongodb-linux-x86\_64-rhel70-3.6.3.tgz

[root@mongodb ~]# mv mongodb-linux-x86\_64-rhel70-3.6.3 /usr/local/mongodb

[root@mongodb ~]# echo "export PATH=/usr/local/mongodb/bin:\$PATH" >>/etc/profile

[root@mongodb ~]# source /etc/profile

* **创建mongodb存储目录以及配置文件；**

[root@mongodb ~]# mkdir /usr/local/mongodb/mongodb1

[root@mongodb ~]# mkdir /usr/local/mongodb/logs/

[root@mongodb ~]# touch /usr/local/mongodb/logs/mongodb1.log

[root@mongodb ~]# chmod 777 /usr/local/mongodb/logs/mongodb1.log

[root@mongodb ~]# cat <<END >>/usr/local/mongodb/bin/mongodb1.conf

bind\_ip=192.168.100.101

port=27017

dbpath=/usr/local/mongodb/mongodb1/

logpath=/usr/local/mongodb/logs/mongodb1.log

logappend=true

fork=true

maxConns=5000

END

* **编写服务启动脚本；**

[root@mongodb ~]# mongod -f /usr/local/mongodb/bin/mongodb1.conf ##启动实例命令

[root@mongodb ~]# mongo --port 27017 --host IP地址 ##连接服务命令，ctrl+D退出

[root@mongodb ~]# mongod -f /usr/local/mongodb/bin/mongodb1.conf --shutdown ##关闭实例命令

[root@mongodb ~]# echo /usr/local/mongodb/bin/mongod -f /usr/local/mongodb/bin/mongodb1.conf >>/etc/rc.local ##设置开机自启

[root@mongodb ~]# chmod +x /etc/rc.local

[root@mongodb ~]# cat <<END >>/etc/init.d/mongodb

#!/bin/bash

INSTANCE=\$1

ACTION=\$2

case "\$ACTION" in

'start')

/usr/local/mongodb/bin/mongod -f /usr/local/mongodb/bin/"\$INSTANCE".conf;;

'stop')

/usr/local/mongodb/bin/mongod -f /usr/local/mongodb/bin/"\$INSTANCE".conf --shutdown;;

'restart')

/usr/local/mongodb/bin/mongod -f /usr/local/mongodb/bin/"\$INSTANCE".conf --shutdown

/usr/local/mongodb/bin/mongod -f /usr/local/mongodb/bin/"\$INSTANCE".conf;;

esac

END

[root@mongodb ~]# chmod +x /etc/init.d/mongodb

[root@mongodb ~]# /etc/init.d/mongodb mongodb1 start

[root@mongodb ~]# /etc/init.d/mongodb mongodb1 stop

* **测试访问mongodb数据库；**

[root@mongodb ~]# mongo --port 27017 --host 192.168.100.101

> show dbs

admin 0.000GB ##管理数据库，存档用户密码等信息；

config 0.000GB ##配置管理mongodb数据库所用

local 0.000GB ##主要存储副本集的配置信息、oplog信息，这些信息是每个Mongod进程独有的，不需要同步到副本集种其他节点；

> exit

**七、Mongodb中语句操作；**



**注：可以补全、可以不加 ;**

[root@mongodb ~]# mongo --port 27017 --host 192.168.100.101

> help ##查看帮助

> db.help

> show dbs ##查看数据库

admin 0.000GB

config 0.000GB

local 0.000GB

> show databases

admin 0.000GB

config 0.000GB

local 0.000GB

> use local ##进入数据库，没有且创建

switched to db local

> show tables ##查看集合

startup\_log

> show collections ##查看集合

startup\_log

> use linuxfan ##创建数据库linuxfan

switched to db linuxfan

> db.user.insert({"id":1,"name":"tom"}); ##创建集合user，并设置键为"id":1，值为"name":"tom"

WriteResult({ "nInserted" : 1 })

> show dbs ##查看数据库

admin 0.000GB

config 0.000GB

linuxfan 0.000GB

local 0.000GB

> show collections ##查看集合

user

> db.user.find() ##查看集合中的数据记录

{ "\_id" : ObjectId("5b700cfad455035330c2dc82"), "id" : 1, "name" : "tom" }

> db.user.count() ##统计集合的条目

1

> db.stats(); ##查看集合状态

{

"db" : "linuxfan",

"collections" : 1,

"views" : 0,

"objects" : 1,

"avgObjSize" : 48,

"dataSize" : 48,

"storageSize" : 16384,

"numExtents" : 0,

"indexes" : 1,

"indexSize" : 16384,

"fsUsedSize" : 2697973760,

"fsTotalSize" : 19815989248,

"ok" : 1

}

>db.user.insert({"id":2,"name":"jack","isadmin":true,"gender":null,"favorite":["apple","banana" ,"orange",1,2,3],"regtime":new Date()}); ##集合中插入条目，并设置多种数据类型

WriteResult({ "nInserted" : 1 })

> db.user.find() ##查看集合中内容

{ "\_id" : ObjectId("5b700cfad455035330c2dc82"), "id" : 1, "name" : "tom" }

{ "\_id" : ObjectId("5b701638d4ea3f227a919f4e"), "id" : 2, "name" : "jack", "isadmin" : true, "gender" : null, "favorite" : [ "apple", "banana", "orange", 1, 2, 3 ], "regtime" : ISODate("2018-08-12T11:12:56.056Z") }

> db.user.findOne({"id":"2"}) ##根据key值搜索集合中内容

{

"\_id" : ObjectId("5b701638d4ea3f227a919f4e"),

"id" : 2,

"name" : "jack",

"isadmin" : true,

"gender" : null,

"favorite" : [

"apple",

"banana",

"orange",

1,

2,

3

],

"regtime" : ISODate("2018-08-12T11:12:56.056Z")

}

> a = db.user.findOne({'id':2}) ##查看集合中值的类型

{

"\_id" : ObjectId("5b701638d4ea3f227a919f4e"),

"id" : 2,

"name" : "jack",

"isadmin" : true,

"gender" : null,

"favorite" : [

"apple",

"banana",

"orange",

1,

2,

3

],

"regtime" : ISODate("2018-08-12T11:12:56.056Z")

}

> typeof(a.id) ##查看id值类型为数字

number

> typeof(a.name) ##查看name值类型为字符串

string

> db.user.insert({"id":1,"id":"2"}); ##后插入的key值会覆盖先存在的key值

WriteResult({ "nInserted" : 1 })

> db.user.find()

...

{ "\_id" : ObjectId("5b70173ed4ea3f227a919f50"), "id" : "2" }

> db.user.insert({"id":1,"ID":"2"}); ##默认区分大小写

WriteResult({ "nInserted" : 1 })

> db.user.find()

...

{ "\_id" : ObjectId("5b70173ed4ea3f227a919f50"), "id" : "2" }

{ "\_id" : ObjectId("5b70176ad4ea3f227a919f51"), "id" : 1, "ID" : "2" }

> db.user.insert({"":"20"}); ##key值可以为空，但是通常不会用

WriteResult({ "nInserted" : 1 })

> db.user.find()

...

{ "\_id" : ObjectId("5b7017c0d4ea3f227a919f52"), "" : "20" }

> db.user.insert({"$id":"20"}); ##key值不可以以$开头

WriteResult({

"nInserted" : 0,

"writeError" : {

"code" : 2,

"errmsg" : "Document can't have $ prefixed field names: $id"

}

})

> db.user.insert({"i$d":"20"}); ##特殊符号可以作为key值中的一部分，但不可为开头

WriteResult({ "nInserted" : 1 })

> db.user.find()

...

{ "\_id" : ObjectId("5b7017f9d4ea3f227a919f54"), "i$d" : "20" }

> db.user.update({"id":1},{$set:{ "name":"xiaohong"}}); ##更新集合中的内容

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.user.find()

{ "\_id" : ObjectId("5b700cfad455035330c2dc82"), "id" : 1, "name" : "xiaohong" }

> for(i=1;i<=1000;i++){db.user.insert({"id":i,"name":"haha"})}; ##批量增加集合条目

WriteResult({ "nInserted" : 1 })

> db.user.remove({"id":1}) ##删除集合中的某条数据

WriteResult({ "nRemoved" : 3 })

> db.version() ##查看mongodb的版本

3.6.3 ##删除当前所在的数据库

> db.user.drop() ##删除集合

true

> db.dropDatabase() ##删除当前所在的数据库

{ "dropped" : "linuxfan", "ok" : 1 }

> ctrl + d

bye

**八、Mongodb数据库的导入导出、备份恢复、复制数据库；**

**导入导出：**

导入语法： mongoimport –d database\_name –c collection\_name –-file source\_name

导出语法： mongoexport –d database\_name –c collection\_name [–f list\_name] –o backup.json ##有些选项为可选，若进行数据过滤条件判断指定选项-q

**案例：**

[root@mongodb ~]# yum -y install mysql mariadb-server

[root@mongodb ~]# systemctl start mariadb

[root@mongodb ~]# mysql

MariaDB [(none)]> create database linuxfan;

Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> use linuxfan;

Database changed

MariaDB [linuxfan]> create table t1(id int,name varchar(20));

Query OK, 0 rows affected (0.01 sec)

MariaDB [linuxfan]> insert into t1 values(1,"Jack");

Query OK, 1 row affected (0.00 sec)

MariaDB [linuxfan]> insert into t1 values(2,"Rose");

Query OK, 1 row affected (0.00 sec)

MariaDB [linuxfan]> select \* from t1;

+------+------+

| id | name |

+------+------+

| 1 | Jack |

| 2 | Rose |

+------+------+

MariaDB [linuxfan]> select \* from linuxfan.t1 into outfile '/tmp/t1\_mysql.csv' fields terminated by ",";

Query OK, 2 rows affected (0.00 sec)

MariaDB [linuxfan]> exit

[root@mongodb ~]# cat /tmp/systemd-private-a1a3e9c7f04f4ae6809b13f327d3d907-mariadb.service-D8WK2p/tmp/t1\_mysql.csv

1,Jack

2,Rose

[root@mongodb ~]# /usr/local/mongodb/bin/mongoimport --port 27017 --host 192.168.100.101 -d benet -c tt1 -f id,name --file /tmp/systemd-private-a1a3e9c7f04f4ae6809b13f327d3d907-mariadb.service-D8WK2p/tmp/t1\_mysql.csv --type csv

2018-08-12T20:21:45.557+0800 connected to: 192.168.100.101:27017

2018-08-12T20:21:45.569+0800 imported 2 documents

[root@mongodb ~]# mongo --port 27017 --host 192.168.100.101

> show dbs

admin 0.000GB

benet 0.000GB

cloud 0.000GB

config 0.000GB

local 0.000GB

> use benet

switched to db benet

> show tables;

tt1

> db.tt1.find()

{ "\_id" : ObjectId("5b7026598e7970c2106ebc79"), "id" : 1, "name" : "Jack" }

{ "\_id" : ObjectId("5b7026598e7970c2106ebc7a"), "id" : 2, "name" : "Rose" }

> exit

[root@mongodb ~]# mongo --port 27017 --host 192.168.100.101

> use benet

switched to db benet

> for(var i=1;i<=100000;i++)db.tt1.insert({"id":i,"name":"name"+i});

WriteResult({ "nInserted" : 1 })

> db.tt1.find({"id":{"$gt":50000}}).limit(3);

{ "\_id" : ObjectId("5b702764deb0bd591420d9e5"), "id" : 50001, "name" : "name50001" }

{ "\_id" : ObjectId("5b702764deb0bd591420d9e6"), "id" : 50002, "name" : "name50002" }

{ "\_id" : ObjectId("5b702764deb0bd591420d9e7"), "id" : 50003, "name" : "name50003" }

> db.tt1.find({"id":{"$gt":50000}}).count();

50000

> exit

[root@mongodb ~]# /usr/local/mongodb/bin/mongoexport --host 192.168.100.101 --port 27017 -d benet -c tt1 -q '{"id":{"$gt":50000}}' -o /tmp/test.json

2018-08-12T22:41:56.942+0800 connected to: 192.168.100.101:27017

2018-08-12T22:41:57.413+0800 exported 50000 records

[root@mongodb ~]# ls /tmp/

mongodb-27017.sock systemd-private-a1a3e9c7f04f4ae6809b13f327d3d907-mariadb.service-D8WK2p test.json

[root@mongodb ~]# wc -l /tmp/test.json

50000 /tmp/test.json

**备份恢复：**

逻辑备份： mongodump –h server\_ip –d database\_name –o dbdirectory

物理备份： 冷备

恢复语法： mongorestore -d database\_name --dir=dbdirectory

**案例：**

[root@mongodb ~]# mkdir /backup

[root@mongodb ~]# /usr/local/mongodb/bin/mongodump --host 192.168.100.101 --port 27017 -d benet -o /backup/

2018-08-12T23:20:51.629+0800 writing benet.tt1 to

2018-08-12T23:20:51.752+0800 done dumping benet.tt1 (100012 documents)

[root@mongodb ~]# ls /backup/

benet

[root@mongodb ~]# ls /backup/benet/

tt1.bson tt1.metadata.json

[root@mongodb ~]# /usr/local/mongodb/bin/mongorestore --host 192.168.100.101 --port 27017 -d bdqn --dir=/backup/benet/

2018-08-12T23:23:36.298+0800 the --db and --collection args should only be used when restoring from a BSON file. Other uses are deprecated and will not exist in the future; use --nsInclude instead

2018-08-12T23:23:36.298+0800 building a list of collections to restore from /backup/benet dir

2018-08-12T23:23:36.299+0800 reading metadata for bdqn.tt1 from /backup/benet/tt1.metadata.json

2018-08-12T23:23:36.340+0800 restoring bdqn.tt1 from /backup/benet/tt1.bson

2018-08-12T23:23:36.754+0800 no indexes to restore

2018-08-12T23:23:36.754+0800 finished restoring bdqn.tt1 (100012 documents)

2018-08-12T23:23:36.754+0800 done

[root@mongodb ~]# mongo --port 27017 --host 192.168.100.101

> show dbs;

admin 0.000GB

bdqn 0.002GB

benet 0.002GB

cloud 0.000GB

config 0.000GB

local 0.000GB

> exit

**复制数据库：**

复制数据库： db.copyDatabase("source\_db","des\_db","192.168.100.102")

克隆集合： db.runCommand({cloneCollection:"accp.t1",from:"192.168.100.102:27017

"}) ##将远程ip服务器的某个数据库中的某个集合克隆到本地

**案例：**

[root@lwh ~]# mongo --port 27017 --host 192.168.100.101

> db.copyDatabase("bdqn","bdqn2","192.168.100.101");

{ "ok" : 1 }

> show dbs

admin 0.000GB

bdqn 0.002GB

bdqn2 0.002GB

benet 0.002GB

cloud 0.000GB

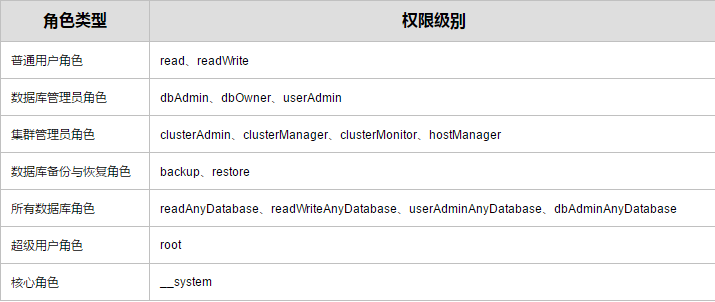
config 0.000GB

local 0.000GB

> exit

**九、Mongodb数据库的用户角色权限管理；**

**角色：**



注意：

创建用户时，需要在默认test数据库中，不use到任何数据库；

查看扫描用户时，需要在admin数据库中，执行use admin切换进去；

撤销修改其权限时，需要在用户所属的db数据库中进行；

[root@lwh ~]# mongo --host 192.168.100.105 --port 27017

> db.getUsers() ##查看所有用户

[ ]

> db.createUser({user: "root", pwd: "abc123", roles: [ { role: "root", db: "admin" } ] }) ##创建超级管理员root

Successfully added user: {

"user" : "root",

"roles" : [

{

"role" : "root",

"db" : "admin"

}

]

}

> db.getUsers() ##查看所有用户

[

{

"\_id" : "test.root",

"user" : "root",

"db" : "test",

"roles" : [

{

"role" : "root",

"db" : "admin"

}

]

}

]

> exit

[root@lwh ~]# mongod -f /usr/local/mongodb/bin/mongodb1.conf --shutdown

killing process with pid: 6999

[root@lwh ~]# mongod -f /usr/local/mongodb/bin/mongodb1.conf --auth ##开启认证

about to fork child process, waiting until server is ready for connections.

forked process: 7608

child process started successfully, parent exiting

[root@lwh ~]# mongo --host 192.168.100.105 --port 27017

> show dbs ##无法认证，所以失败

2018-08-13T00:34:53.745+0800 E QUERY [thread1] Error: listDatabases failed:{

"ok" : 0,

"errmsg" : "not authorized on admin to execute command { listDatabases: 1.0, $db: \"admin\" }",

"code" : 13,

"codeName" : "Unauthorized"

} :

\_getErrorWithCode@src/mongo/shell/utils.js:25:13

Mongo.prototype.getDBs@src/mongo/shell/mongo.js:65:1

shellHelper.show@src/mongo/shell/utils.js:816:19

shellHelper@src/mongo/shell/utils.js:706:15

@(shellhelp2):1:1

> db.auth('root','abc123') ##认证用户

1

> show dbs ##验证查看权限

admin 0.000GB

bdqn 0.002GB

bdqn2 0.002GB

benet 0.002GB

cloud 0.000GB

config 0.000GB

local 0.000GB

> use haha

switched to db haha

> db.haha.insert({"id":"1","num":"101"}) ##测试插入内容

WriteResult({ "nInserted" : 1 })

> db.haha.find()

{ "\_id" : ObjectId("5b70629599b86a58e43b7b8c"), "id" : "1", "num" : "101" }

> exit

[root@lwh ~]# mongo --host 192.168.100.105 --port 27017

> db.auth('root','abc123') ##认证用户

1

>db.createUser({user:"hahaadmin",pwd:"123123",roles:[{role:"read",db:"haha"},{role:"readWrite",db:"hehe"}]}) ##创建普通用户

Successfully added user: {

"user" : "hahaadmin",

"roles" : [

{

"role" : "read",

"db" : "haha"

},

{

"role" : "readWrite",

"db" : "hehe"

}

]

}

> db.getUsers() ##查看所有用户

[

{

"\_id" : "haha.hahaadmin",

"user" : "hahaadmin",

"db" : "haha",

"roles" : [

{

"role" : "read",

"db" : "haha"

},

{

"role" : "readWrite",

"db" : "hehe"

}

]

}

]

> exit

[root@lwh ~]# mongo --host 192.168.100.105 --port 27017

> db.auth('hahaadmin','123123') ##登录普通用户，无法认证

Error: Authentication failed.

0

> use haha ##切换至有权限的数据库

switched to db haha

> db.auth('hahaadmin','123123') ##认证账户，测试访问数据

1

> show tables;

haha

> db.haha.find()

{ "\_id" : ObjectId("5b70629599b86a58e43b7b8c"), "id" : "1", "num" : "101" }

> exit

[root@lwh ~]# mongo --host 192.168.100.105 --port 27017

> db.auth('root','abc123')

1

> use haha

switched to db haha

> db.grantRolesToUser("hahaadmin",[{role:"readWrite",db:"haha"}]) ##现有权限基础上增加权限

> use admin

switched to db admin

> db.system.users.find() ##扫描所有用户及权限

> use haha

switched to db haha

> db.revokeRolesFromUser("hahaadmin",[{role:"readWrite",db:"hehe"}]) ##现有权限基础上撤销权限

> use admin

switched to db admin

> db.system.users.find() ##扫描所有用户及权限

> db.system.users.remove({user:"hahaadmin"}) ##删除用户

WriteResult({ "nRemoved" : 1 })

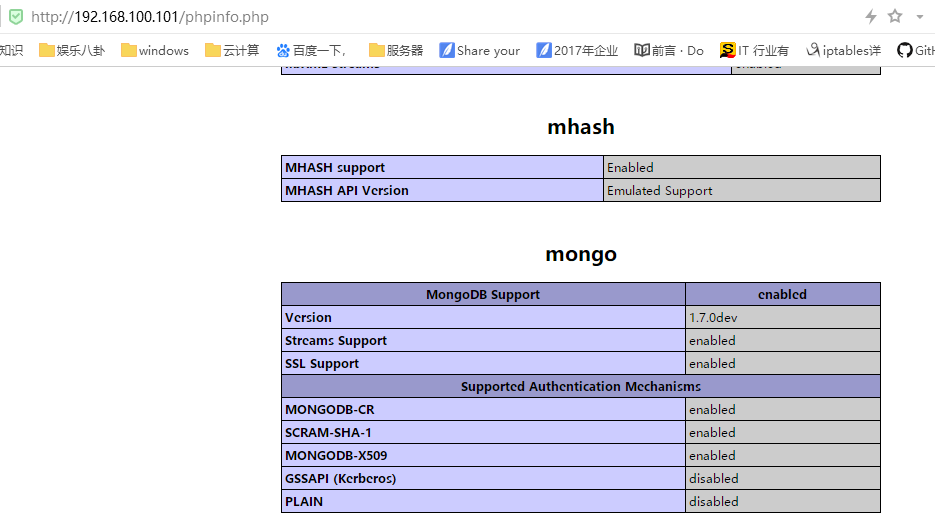
> db.system.users.find() ##扫描系统中所有用户，注意当前必须在admin库中

...

> exit

**十、Mongodb连接php程序；**





**十一：自主学习：Mongodb如何占用内存，占用内存过高如何处理？**

https://blog.csdn.net/liqfyiyi/article/details/73613669

https://www.cnblogs.com/luo-mao/p/5872388.html

https://blog.csdn.net/top\_gun\_1/article/details/50777329