forever_elf

MonogoDB的GirdFS

GirdFS是一种在MongoDB中存储大二进制文件的机制。

mongofiles内置在MongoDB发布版中，可以用来在GridFS中上传、下载、列示、查找或删除文件。

```
$ echo "Hello World" > foo.txt
$ ./mongofiles put foo.txt
connected to：127.0.0.1
added file：{ _id：ObjecteId('    '),
              filename："foo.txt",
              length：13,
              chunkSize：262144,
              uploadDate：new Date(    ),
              md5："        "
            }
done!
$ ./mongofiles list
connected to：127.0.0.1
foo.txt 13
$ rm foo.txt
$ ./mongofiles get foo.txt
connected to：127.0.0.1
done write to：foo.txt
$ cat foo.txt
Hello World
```

上面的例子中，使用了mongofiles的3个基本操作：put、list和get。put将文件系统中的一个文件添加到GridFS中，list会把所有添加到GridFS中的文件列出来，get是put的逆操作，它将GridFS中的文件写入到文件系统中。mongofiles还支持另外两个操作：search用来按文件名查找GridFS中的文件。delete则从GridFS中删除一个文件。

GridFS是建立在普通MongoDB文档基础上的轻量级文件存储规范。所有相关工作都有客户端驱动或工具完成。GridFS的一个基本思想就是可以将大文件分成很多块，每块作为一个单独的文档存储。因为MongoDB支持在文档中存储二进制数据，可以最大限度的减小块的存储开销。除了存储文件本身的块，还有一个单独的文档用来存储分块的信息和文件的元数据。GridFS的块有个单独的集合。默认情况下，块将使用fs.chunks集合。这个块集合里面文档的结构是非常简单的：

```
{
   "_id"：ObjectId("..."),
   "n"：0,            //这个块在原文件中的顺序编号
   "data"：BinData("..."),  //包含组成文件块的二进制数据
   "files_id"：ObjectId("...") //包含这个块元素的文件文档的_id
}
```

文件的元数据放在另一个集合中，默认是fs.files。这里面每个文档代表了GridFS中的一个文件，与文件相关的自定义元数据也可以存在其中。除用户自定义的键，GridFS规范还定义了一些键：

_id：　　文件的唯一id。在块中作为files_id键的值存储
length ：　文件内容总的字节数
chunkSize ：　每块的大小，以字节为单位。默认是256K
uploadDate ：文件存入GridFS的时间戳
md5 ：　　文件内容的md5校验和，有服务器端生成（用户可以校验md5键的值确保文件正确上传）

< 2018年2月 >

日	一	二	三	四	五	六
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	1	2	3
4	5	6	7	8	9	10

公告

昵称：forever_elf
园龄：2年8个月
粉丝：3
关注：0
+加关注

搜索

找找看
谷歌搜索

常用链接

- 我的随笔
- 我的评论
- 我的参与
- 最新评论
- 我的标签

随笔分类

- AmazeUI(4)
- AngularJS(2)
- AWS
- Bash(2)
- Blockchain(1)
- Bootstrap(2)
- Clean Code学习笔记(10)
- Design Patterns 读书笔记(2)
- Docker(1)
- Front-end(5)
- Git(2)

在服务器端可以通过db.eval函数执行js脚本，也可把js脚本保存在数据库中，然后在别的数据库命令中调用。
db.eval可以在MongoDB的服务器端执行任意的js脚本。这个函数先将给定的js字符串传送给MongoDB，然后返回结果。db.eval可以用来模拟多文档事务:db.eval锁住数据库，然后在执行js，再解锁。虽没有内置的回滚机制，但这的确能保证一系列操作按照指定顺序发生。发送代码有两种选择，或封装进一个函数，或不封装。
db.eval("return 1;")
等价于：db.eval("function(){ return 1 ;}")
只有传递参数的时候。才必须要封装成一个函数。参数通过db.eval的第二个参数传递，要写成一个数组的形式。
db.eval("function(u) { print ('Hello, ' +u+ '!');}",[username])

每个MongoDB的数据库库都有个特殊的集合，叫做system.js，用来存放js变量。这些变量可以在任何MongoDB的js上下文中调用，包括"$where"子句，db.eval调用和MapReduce作业。
db.system.js.insert({"_id" : "x", "value" : 1})
db.system.js.insert({"_id" : "y", "value" : 2})
db.system.js.insert({"_id" : "z", "value" : 3})

db.eval("return x+y+z ;")

安全性：
func = "function() { print('Hello,"+username+" ! ');}"
恶意注入式攻击
username = "'); db.dropDataBase();print ('"
//output
func = "function() {print('Hello,'); db.dropDatabase(); print('!');}"
因此为了避免这种情况，要限定作用域
$func = new MongoCode("function() { print('Hello,'"+username+"!');}",...array("username" => $username));

数据库引用（DBRef）像一个URL，唯一确定一个到文档的引用。它自动加载文档的方式正如网站中URL通过链接自动加载Web页面一样。
DBRef是个内嵌文档，但有些必须键。
{"$ref" : collection, "$id" : id_value}
DBRef指向一个集合。还有一个id_value用来在集合里面根据_id确定唯一的文档。这两条信息使得DBRef能唯一标识MongoDB数据库内的任何一个文档。若要引用另一个数据库中的文档，DBRef中有个可选键"$db"
{"ref" : collection, "$id" : id_value, "$db" : database} //顺序不能改变

users文档
{"_id" : "mike" "display_name" : "Mike D"}
{"_id" : "Kristina" "display_name" : "Cristina C"}
notes文档
{"_id" : 5, "author" : "mike", "text" : "MongoDB is fun!"}
{"_id" : 25, "author" : "kristina", "text" : " and DBRef are easy, too ", "references" : [{"$ref" : "users", "$id" : "mike" },{"$ref" : "notes", "$id" : 5}]}

var note = db.notes.findOne({"_id" : 20});
note,references.forEach(function(ref){
    printjson(db[ref.$ref].findOne({"_id" : ref.$id}));})
//output
{"_id" : "mike" "display_name" : "Mike D"}
{"_id" : 5, "author" : "mike", "text" : "MongoDB is fun!"}

MongoDB支持从文件获取配置信息。当需要的配置非常多或要自动化MongoDB的启动时就会用到这个。指定配置文件可以用-f或--config选项。
mogod --config ~/.mongodb.conf

让MongoDB停下来的最基本方法是向MongoDB服务器发送一个SIGNT或SIGTERM信号。若服务器是作为前台进程运行在终端，就直接Ctrl C，否则就用kill这种命令发出信号。若mongod的PID是10014，则kill -2 10014 或 kill 10014。另一种稳妥的方式就是使用shutdown命令，这是管理员命令，再要admin数据库下使用。
use admin
//itched to db admin
db.shutdownServer
//server should be down
切记 不要向运行中的MongoDB发送SIGKILL，这会导致数据库直接关闭，不会等当前运行的操作或文件与分配完成，会是数据库文件损坏。要是真的发生不幸，一定要在启动备份前修复数据库。

默认情况下，启动mongod时还会启动一个基本的HTTP服务器，该服务器监听的窗口号比主服务的端口号大1000.这个服务提供了Http接口，可以查看MongoDB的一些基本信息。

随笔档案

阅读排行榜

要获取运行中的MongoDB服务器统计信息，最基本的工具就是serverStatus命令。

mongostat输出一些serverStatus提供的重要信息。它会每秒钟输出新的一行。

向MongoDB添加管理员
db.addUser("root", "abc", "true"); //将第三个参数设为true后即可，该数据库变为只读。

--auth 开启安全检查

数据库的用户帐号以文档的形式存储在syste.users集合里面。文档的结构是{"user"：username, "readOnly"：true, "pwd"：password hash} // password hash是根据用户名和密码生成的散列。

MongoDB将所有数据都存放在数据目录下。默认目录是/data/db。若想要备份MongoDB，只要简单创建数据库目录中所有文件的副本就可以了。在运行MongoDB时复制数据目录不太安全，所以要把服务器关了在再复制数据库。

mongodump是一种能在运行时备份的方法。mongodump对运行的MongoDB做查询，然后将所有查到的文档写入到磁盘。

mongorestore获取mongodump的输出结果，并将备份的数据插入到运行的MongoDB实例中。
```
$ ./mongodump -d test -o backup
connected to :127.0.0.1
DATABASE：test to  backuo/test
  test.x to backup/text/x.bson
      1 objects
$ ./mongorestore -d foo --drop backuo.test/
connected to：127.0.0.1
backuo/test/x.bson
     going into namespace [foo.x]
     1 objects
```

MongoDB的fsync命令能在MongoDB运行时复制数据目录还不会损坏数据。fsync命令会强制服务器将所有缓存区写入磁盘。还可以选择上锁阻止对数据库的进一步写入，直到释放锁为止。
```
use admin
db.runCommad({"fsync"：1, "lock"：1});  //上锁
db.$cmd.sys.unlock.findOne();  //解锁
db.currentOp();  //确保已经解锁
```

在从服务器上备份是MongoDB推荐的备份方式。

修复所有数据库的方式是 --repair：mongod --repair来启动服务器。修复数据库的实际过程实际上非常简单：将所有的文档导出然后马上导入、忽略那些无效的文档。完成以后，会重新建立索引。
修复运行中的服务器上的数据库，要在shell中用repairDatabase。
```
use test
db.repairDatebase
```

主从复制是MongoDB最常用的复制方式。最基本的设置方式是建立一个主节点和一个或多个从节点，每个从节点要知道主节点的地址。运行mongod --master就启动了主服务器。运行mongod --slave --source master_address（主节点地址）则启动了从服务器。
主从复制的一些有用的选项：
```
--only            在从节点上指定只复制特定某个数据库
--slavedelay      在从节点上，当应用主节点的操作时增加延时
--fastsync        以主节点的数据快照为基础启动从节点。
--autoresync      若从节点与主节点不同步了，则自动更新同步
--oplogSize       主节点iplog的大小
```

启动从节点时可以用--source指定主节点。若主节点绑定了localhost:27017，启动从节点时可以不添加源，而是随后向source集合添加主节点信息。
```
$ ./mongod --slave --dbpaht ~/dbs/slave --port 27018
```
也可以在shell中运行命令
```
use local
db.sources.insert({"host"："localhost：27017"})
db.sources.remove({"host"："localhost：27017"})
```


1. Spring反射机制(5227)
2. AmazeUI布局(2650)
3. Maven属性、profile和资源过滤(2639)
4. 数据库和集合的基本操作(649)
5. AmazeUI基本样式(647)

推荐排行榜

1. AmazeUI布局(1)
2. Maven属性、profile和资源过滤(1)

副本集(Replica Set)有自动故障恢复功能的主从集群。主从集群和副本集最为明显的区别是副本集没有固定的"主节点"：整个集群会选举出一个“主节点”，当其不能工作时则变更到其他节点。副本集总会有一个活跃节点(primary)和一个或多个备份节点(secondary)。

设置副本集

不能使用localhost地址作为成员，所以得找到机器的主机名。
两个服务器的情况下：
为每个服务器创建数据目录，选择端口
$ mkdir -p ~/dbs/nodel ~/dbs/node2 //之后要启服
$ ./mongod --dbpath !/db/node1 --port 10001 --replSet blort/morton:10002   //replSet的作用是让服务器知晓在这个blort副本集中还有别的同伴 位置在morton:10002   之后要启服
$ ./mongod --dbpath ~/dbs/node2 --port 10002 --replSet blort/morton:10001
若想添加第三台服务器，则
$ ./mongod --dbpath ~/dbs/node3 --port 10003 --replSet blort/morton:10001
等价于: $ ./mongod --dbpath ~/dbs/node3 --port 10003 --replSet blort/morton:10001,morton : 10002
在shell中初始化副本集
$ ./mongo morton : 10001/admin
```
db.runCommand({"relSetInitiate" : {
   "_id" : "blort",
   "members" :[
      {
        "_id" : 1,
        "host" : "morton : 10001"
       },{
        "_id" : 2,
        "host" : "morton : 10002"
       },{
        "_id" : 3,
        "host" : "morton : 10003"
       }
    ]
  }
})
//output
{
   "info" : "Config now saved locally. Shoule come online in about a minute."
    "ok" :true
}
```

副本集中的节点：任何时间里，集群只有一个活跃节点，其他的都为备份节点。有几种不同类型的节点可以存在于副本集中:
standard:常规节点。它存储一份完整的数据副本，参与选举投票，有可能成为活跃节点
passive:存储了完整的数据副本，参与投票，不能成为活跃节点
arbiter：仲裁者只参与投票，不接收复制的数据，也不能成为活跃节点
标准节点和被动节点之间的区别仅仅是数量的差别：每个参与节点有个优先权。优先权为0则是被动的，不能成为活跃节点。优先值部位0，则按照由大到小选出活跃节点，优先值一样的化则看谁的数据比较新。
默认优先级为1，可以是0~1000。
```
members.push({
   "_id" : 4,
   "host" : "morton : 10004",
   "priority" : 40 ,     //设置优先级
   "arbiterOnly" : true  //是否为仲裁节点
})
```

备份节点会从活跃节点抽取oplog，并执行操作。活跃节点也会写操作到自己的本地oplog，这样就成活跃节点了。

活跃节点使用心跳来跟踪集群中有多少节点对其可见。若不够半数，活跃节点会自动降为备份节点。不论活跃节点何时变化，新活跃节点的数据都被假定为系统的最新数据。对其他节点的操作都会回滚，即便是之前的活跃节点已恢复工作。为了完成回滚，所有节点链接新的活跃节点后要更新同步。这些节点会查看自己的oplog，找出其中活跃节点没有执行过的操作，然后向活跃节点请求这些操作影响文档的最新副本。正在执行重新同步的节点被视为恢复中，在完成这个过程之前不能成为活跃节点候选者。

用MongoDB扩展读取的一种方式就是将查询放在从节点上。扩展读取本身很简单：像以往一样设置主从复制，连接从服务器处理请求。唯一的技巧就是 有个特殊的查询选项，告诉从服务器是够可以处理请求。这

个选项叫做slaveOkay。
从节点的另外一个用途就是作为一种机制来减轻密集型处理的负载，或作为聚合，避免影响主节点的性能。用--master参数启动一个普通的从节点。同时使用--slave和--master有点矛盾。

MongoDB的复制至少需要两个服务器或者节点。其中一个是主节点，负责处理客户端请求，其他的都是从节点，负责映射主节点的数据。主节点记录在骑上执行的所有操作。从节点定期轮询主节点获得这些操作，然后对自己的数据副本执行这些操作。由于和主节点执行了相同的操作，主从节点就能保持与主节点的数据同步。

主节点的操作记录成为oplog（operation log的简写）。oplog存储在一个特殊的数据库中，叫做local。oplog就在其中的oplog.$main集合里面。oplog中的每个文档都代表主节点上执行一个操作。文档包含的键如下：
ts ： 操作的时间戳。时间戳是一种内部类型，用于跟踪操作执行的事件。由4字节的时间戳和4字节的递增计数器构成。
op ： 操作类型，只有1字节代码
ns ： 执行操作的命名空间
o ： 进一步制定要执行的操作文档。对插入来说，就是要插入的文档。
oplog只记录改变数据库状态的操作。

同步：从节点第一次启动时，会对主节点数据进行完整的同步。从节点复制主节点上的每个文档，耗费的资源可想而知。同步完成后，从节点开始查询主节点的oplog并执行这些操作，以保证数据是最新的。若从节点的操作已经被主节点落下很远了，从节点就跟不上同步了。跟不上同步的从节点无法一直不断的追赶主节点，因为主节点oplog的所有操作都太“新”了。从节点发生了宕机或疲于应付读取时就会出现这种情况。也会在执行完完整同步以后发生类似的事，因为只要同步时间太长，同步完成时，oplog就可能已经滚一圈了。
从节点跟不上同步时，复制就会停下来，从节点需要重新做完整的同步。可以用{"resync"：1 }命令手动执行重新同步，也可以在启动从节点时使用--autoresync选项让其自动重新同步。为了避免从节点跟不上，一定要确保主节点的oplog足够大，能存放相当长时间的操作记录。

本地数据库用来存放所有内部复制状态，主节点和从节点都有。本地数据库的名字就是local，其内容不会被复制。

主节点和从节点都跟踪从节点的更新状况，这是通过存放在"syncedTo"中的时间戳来完成的。每次从节点查询主节点的oplog时，都会用"syncedTo"来确定那些操作需要执行，或者查看是否已经跟不上同步了。

开发者可以用getLastError的'w'参数来确保数据的同步性。这里运行getLastError会进入阻塞状态，知道N个服务器复制了最新的写入操作为止。
db.runCommand({getLastError：1, w：N });

db.printReplicationInfo()可以看到oplog的大小和oplog中操作的时间范围。
```
db.printReplicationInfo();
//output
configured oplog size：10.45MB
log length start to end：34secs
oplog first event time：
oplog laser evebt time：
now：
```
oplog的长度至少要能满足一次完成的重新同步。
日志的长度是通过oplog中最早的操作事件和最后的操作事件的差值得到的。

db.printSlaveReplicationInfo()可以显示从节点的数据源列表，其中有数据滞后时间。
```
db.printSlaveReplicationInfo();
//output
source：localhost：27017
syncedTo：                = 12 secs ago  //数据滞后时间
```

若发现oplog大小不合适，最简单的做法就是停掉主节点，删除local数据库的文件，用新的设置重新启动。
```
$ rm/data/db/local.*
$ ./mongod --master --oplogSize size
size is specified in megabytes
```

重启主节点之后，所有从节点得用--autoresync重启，否则需要手动重新同步。

若在复制的过程中使用了认证，还需要做些配置，是的主从节点能够访问主节点的数据。在主节点和从节点上都需要在本地数据库添加用户，每个节点的用户名和口令都是相同的。

分片是MongoDB的扩展方式，通过分片能够增加更多的机器来应对不断增加的负载和数据。分片(sharding)是指将数据拆分，将其分散存在不同的机器上的过程。有时也用分区(partitioning)来表示这个概念。

MongoDB分片的基本思想就是将集合切分成小块。这些块分散到若干片里面，每个片只负责总数居的一部分。在分片之前要运行一个路由进程，该进程名为mongos。这个路由器知道所有数据的存放位置，所以应用可以连接他来正常发送请求。

在以下几种情形下开始使用分片：
机器的磁盘不够用了
单个mongod已经不能满足写数据的性能需要了
想将大量数据放在内存中提高性能

设置分片时，需要从结合里面选一个键，用该键的值作为数据拆分的依据。这个键称为片键(shard key)。当片键建好并运行后，MongoDB就会把集合拆分成两半，成为块。每个块中包括片键值在一定范围内的所有文档。片键的选择决定了插入操作在片之间的分布。

分片一般会有3个组成部分：
片 就是保存子集合数据的容器。片可是单个mongod服务器，也可以是副本集。
mongos 就是MongoDB各版本中都匹配的路由器进程。它路由所有请求，然后将结果聚合。
配置服务器 存储了集群的配置信息：数据和片的对应关系。mongos不永久存放数据。

启动mongos
$ mkdir -p ~/dbs/config
$ ./mongod --dbpath ~/dbs/config --port 20000
建立mongos进程，以供应应用程序连接。
$ ./mongos --port 30000 --configdb localhost:20000
添加片
$ mkdir -p ~/dbs/shard1
$ ./mongod --dbpath ~/dbs/shard1 --port 10000
启动mongos，为集群添加一个片
$ ./mongo localhost:30000/admin
//output
MongoDB shell version：1.6.0
url：localhost:30000/admin
connecting to localhost:30000/admin
type "help" for help

db.runCommand({addshard ："localhost:10000",allowLocal：true})
//output
{
   "added"："localhost:10000",
   "ok"：true
}

开启foo的分片功能：
db.runCommand({"enablesharding"："foo"})
对集合进行分片
db.runCommand({"shardcollection"："foo.bar", "key"：{"_id"：1}})

配置多个服务器
$ mkdir -p ~/dbs/config1 ~/dbs/config2 ~/dbs/config3
$ ./mongod --dbpath ~/dbs/config1 --port 20001
$ ./mongod --dbpath ~/dbs/config2 --port 20002
$ ./mongod --dbpath ~/dbs/config3 --port 20003
$ ./mongos --confidb localhost：20001, localhost：20002, localhost：20003

在生产环境中，每个片都应是副本集。
db.runCommand({"addshard"："foo/prod.example/com:27017"})

查找所有的片
db.shared.find();

db.printShardingStatus()

db.runCommand({"removedshard"："localhost：10000" })

MongoDB权威指南看完了，感觉看的快，忘得也快。看的都是理论上的东西，感觉懵懵懂懂的，还没有实践。回去要做几个例子才行。不过Maven还没有看，think in java也没有看，Hibernate还没有看完，还有Spring想要深入了解一下...我的任务有点多啊..唯一不忙的就是工作..倒是很希望工作可以忙起来，这样学的会更快、更多。继续加油吧！

分类: MongoDB

forever_elf
关注 - 0
粉丝 - 3
+加关注

0 0

« 上一篇：MongoDB索引、聚合
» 下一篇：AngularJS基本指令

发表于 2015-08-12 14:09 FOREVER_ELF 阅读(103) 评论(0) 编辑 收藏

· 作为一个程序员，数学对你到底有多重要
· 领域驱动设计在互联网业务开发中的实践
» 更多知识库文章…