火龙果•整理
uml.org.cn

MongoDB入门实战手册

技术研究部

目录

1 MongoDB简介 3
2 MongoDB服务器的安装 4
2.1 源码版本的安装 4
2.2 编译版本安装 6
2.2.1 MongoDB版本下载 6
2.2.2 Windows版本安装 7
2.2.3 Linux版安装 9
3 MongoDB客户端访问 10
3.1 Shell方式 10
3.2 JAVA方式 11
4 MySQL与MongoDB设计实例对比 14
5 性能测试 32
5.1 测试案例一 32
5.2 测试案例二：Mongodb亿级数据量的性能测试 33

1 MongoDB简介

2011 年将被记住，因为这一年 SQL 将死；这一年，关系数据库从一线退下；这一年开发人员发现他们没必要为了持久化数据，而将每个对象转化为表格结构。

2011 年是文档数据库的一年，尽管一直在稳步发展势头，通过过去八年多的发展，现在有各种稳定的文档数据库——从基于亚马逊和谷歌的云，到各种开放源码工具，尤其是 MongoDB。

那么，MongoDB 是什么?这里的五件事是每个开发人员应该知道的：

- MongoDB 是一个独立的服务器；
- 它是基于文档的，而不是基于表格的；
- 它是非结构化的；
- 不必去学习另一种查询语言；
- 它具有强大的主流开发语言支持，如 C#、C++、Java、PHP、Perl、Python、Ruby。

1） MongoDB 是一个独立的服务器

如 MySQL 或 PostgreSQL 一样，MongoDB 提供侦听端口以便接入。它提供了用于查询，创建，更新和删除的工具。从理论上讲，你使用它的工作方式与你使用 MySQL 或 PostgreSQL 的工作方式相同：连接，执行任务，并关闭连接。

2） MongoDB 是非结构化的

MongoDB 没有结构化语言。如果你想创建一个新的文档类型，你不用做任何事来告诉数据库关于这些数据的结构，而仅仅是存到数据库中即可。

简单的说，MongoDB 使用类似 JavaScript 或 PHP 的类型处理方式。也就是说，数据库是灵活的弱类型。

虽然有一些数据是有限制条件的(大块的数据可能需要一些明确的处理)，但在大多数情况下，你可以像写 PHP 代码一样编写你的 MongoDB 代码。

3） 不必去学习另一种查询语言

还记得这些你写的数据库抽象层吗?还记得那些你处理过的 ORM 层吗?现在，你可以将它们全部丢弃。在 MongoDB 中你不需要他们。MongoDB 没有很多查询语句。在大多数情况下，只需给它一个数组指定你想要的信息，然后它会给你返回文档的数组。如果你想运行一些非常复杂的查询(如 Map-Reduce 操作)，可以向 MongoDB 传递 JavaScript，其内部的 JavaScript 引擎可以解析这个脚本。

4） MongoDB 是神速的

开发时间也短，因为没有结构需要管理和很少(如果有的话)的数据映射。

学习曲线很平滑，因为没有新的查询语言学习。代码是简洁的。毕竟，无须任何其他 ORM，封装可以非常简单。你的代码是未来的保证。向你的对象增加更多的字段是很轻松的。因此，需求变化了，你可以很快修改代码以便适应。

MongoDB 足以让我意识到它有改变游戏规则的潜力。这也是让大家主张使用新一代的文档数据库代替基于 SQL 的关系数据库的原因。将关系数据库留在尘土里，更可能的是让它们做它们能做好的事情：存储属于行和表的数据。

■ **补充说明：**

mongodb 是用 C++开发的面向文档的数据库，也就是反传统的数据库范式来设计的，把相关的对象都记录到一个文档里，每个文档内是 schema-free 的，也就是列名可以自由定义，比较灵活，特别是面对业务逻辑多变的应用场景十分给力。数据以 BSON(类似 JSON)的格式二进制存储。不好的地方就是可能带来一定的数据冗余和存储开销。

很明显，MongoDB 这种面向文档的数据库和传统的关系型数据库的设计思路是差别很大的，因为每个文档都包含了所有信息，和其他文档是没有关联的，这样传统的 Join 操作就完全没必要了，也正是因为去除了这种“关系”，使得 MongoDB 的水平拆分更加容易，这也是面对海量数据的一个很好的处理思路。另外，MongoDB 的索引机制和 MySQL 等数据库是一样的，可以利用传统的关系型数据库的经验来使用 MongoDB 的索引。

不像其他很多 NoSQL 产品由个别工程师根据应用场景开发出来，MongoDB 是有一个专门的公司 10gen 来维护。有一点要注意的是，MongoDB 自己是不管理内存的，无法指定内存大小，完全交给操作系统来管理，因此有时候是不可控的，在生产环境使用必须在 OS 层面监控内存使用情况。

2 MongoDB服务器的安装

2.1 源码版本的安装

mongodb 目前最新的版本是.8.2-rc3，其源码安装用了很多第三方的东西，比如 JS 引擎(目前官方推荐的是 mozilla 的 Spider Monkey，以后可能改成 google 的 V8，和 node.js 一样，呵呵)、正则表达式引擎(pcre)、安装构建工具 scons(这东西还要用 python 来安装)、boost C++库等等。下面是安装过程：

1、下载需要的源文件和相关软件包：

```
[root@localhost mongodb]# wget http://downloads.mongodb.org/src/mongodb-src-r1.8.2-rc3.tar.gz

[root@localhost mongodb]# wget http://sourceforge.net/projects/scons/files/scons/2.1.0.alpha.20101125/scons-2.1.0.alpha.20101125.tar.gz/download

[root@localhost mongodb]# wget http://ftp.mozilla.org/pub/mozilla.org/js/js-1.7.0.tar.gz

[root@localhost mongodb]# wget http://sourceforge.net/projects/pcre/files/pcre/7.4/pcre-7.4.tar.gz/download
```

2、安装 scons：

```
[root@localhost mongodb]# tar zxvf scons-2.1.0.alpha.20101125.tar.gz
```

```
[root@localhost mongodb]# cd scons-2.1.0.alpha.20101125

[root@localhost scons-2.1.0.alpha.20101125]# python setup.py install
```

3、安装 pcre：

```
[root@localhost mongodb]# tar zxvf pcre-7.4.tar.gz

[root@localhost mongodb]# cd pcre-7.4

[root@localhost pcre-7.4]# ./configure

[root@localhost pcre-7.4]# make

[root@localhost pcre-7.4]# make install
```

4、安装 Spider Monkey：

```
[root@localhost mongodb]# tar zxvf js-1.7.0.tar.gz

[root@localhost mongodb]# cd js/src

[root@localhost src]# export CFLAGS="-DJS_C_STRINGS_ARE_UTF8"

[root@localhost src]# make -f Makefile.ref

[root@localhost src]# JS_DIST=/usr make -f Makefile.ref export
```

5、安装 boost，yum 方式比较偷懒：

```
[root@localhost src]# yum -y install boost boost-devel
```

6、安装 mongodb：

```
[root@localhost mongodb]# tar zxvf mongodb-src-r1.8.2-rc3.tar.gz

[root@localhost mongodb]# cd mongodb-src-r1.8.2-rc3

[root@localhost mongodb-src-r1.8.2-rc3]# scons all

[root@localhost mongodb-src-r1.8.2-rc3]# scons --prefix=/usr/local/mongodb --full install
```

这样就安装完毕了，可以简单的启动 mongod 进程来验证一下：

这样就安装完毕了，可以简单的启动 mongod 进程来验证一下：

```
[root@localhost bin]# ./mongod --dbpath /tmp

Wed Jun 8 11:57:38 [initandlisten] MongoDB starting : pid=29700 port=27017 db path=/tmp 64-bit

Wed Jun 8 11:57:38 [initandlisten] db version v1.8.2-rc3, pdfile version 4.5

Wed Jun 8 11:57:38 [initandlisten] git version: nogitversion

Wed Jun 8 11:57:38 [initandlisten] build sys info: Linux localhost.localdomain 2.6.18-128.el5 #1 SMP Wed Jan 21 10:41:14 EST 2009 x86_64 BOOST_LIB_VERSION=1_33_1

Wed Jun 8 11:57:38 [initandlisten] waiting for connections on port 27017

Wed Jun 8 11:57:38 [websvr] web admin interface listening on port 28017
```

可见 mongod 默认在 27017 端口监听，而 28017 端口是 web 管理的端口，可通过 http 方式来访问。为了规范，我们用以下命令启动一个 mongod 进程：

```
[root@localhost data]# /usr/local/mongodb/bin/mongod --fork --dbpath /home/mongo/data/ --logpath /home/mongo/mongo.log --logappend --directoryperdb --journal --rest
```

这样一个 mongod 进程就启动了，它监听 27017 端口来提供服务，可以在应用程序中进行建立数据库等操作，它不像传统的 Oracle 等关系数据库那样，建库是个很慎重的工作。要了解更详细的使用 MongoDB 的信息，读者可以参看官方文档，这里就不提及了。

【关闭】的方法很简单：Killall mongod 或者是 kill [pid]

2.2 编译版本安装

2.2.1 MongoDB版本下载

MongoDB 的官方下载站是 http://www.mongodb.org/downloads，可以去上面下载最新的程序下来。在下载页面可以看到，对操作系统支持很全面，OS X、Linux、Windows、Solaris 都支持，而且都有各自的 32 位和 64 位版本。目前的稳定版本是 1.8.1 版本。

MongoDB Downloads

你的·技术开发频道 tech.it168.com

This table lists MongoDB distributions by platform and version. There are also packages available for various package managers.

	OS X 32-bit note	OS X 64-bit	Linux 32-bit note	Linux 64-bit	Windows 32-bit note	Windows 64-bit	Solaris i86pc note	Solaris 64	Source
Production Release (Recommended)									
1.8.1 4/6/2011 Changelog Release Notes	download	download	download *legacy-static	download *legacy-static	download	download	download	download	tgz zip
Nightly Changelog	download	download	download *legacy-static	download *legacy-static	download	download	download	download	tgz zip

注意：

1. MongoDB 1.8.1 Linux 版要求 glibc 必须是 2.5 以上，所以需要先确认操作系统的 glibc 的版本，笔者最初用 Linux AS 4 安装不上，最后用的是 RHEL5 来安装才成功的；

2. 在 32 位平台 MongoDB 不允许数据库文件(累计总和)超过 2G，而 64 位平台没有这个限制。本文都是采用 32 位系统。

2.2.2 Windows版本安装

(1)、下载 MongoDB

url 地址：http://downloads.mongodb.org/win32/mongodb-win32-i386-1.8.1.zip

(2)、设置 MongoDB 目录

将其解压到 d:\，再重命名为 mongodb，路径为 d:\mongodb

(3)、设置数据文件路径

在 d:盘建一个 db 文件夹，路径 d:\db

(4)、启动 MongoDB 服务

进入 cmd 提示符控制台，D:\mongodb\bin\mongod.exe --dbpath=d:\data\db

```
D:\mongodb\bin>D:\mongodb\bin\mongod --dbpath=d:\data\db
Sun Apr 10 22:34:09 [initandlisten] MongoDB starting : pid=5192 port=27017 dbpat
h=d:\data\db 32-bit
** 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
**      with --dur, the limit is lower
Sun Apr 10 22:34:09 [initandlisten] db version v1.8.1, pdfile version 4.5
Sun Apr 10 22:34:09 [initandlisten] git version: a429cd4f535b2499cc4130b06ff7c2
6
f41c00f04
Sun Apr 10 22:34:09 [initandlisten] build sys info: windows (5, 1, 2600, 2, 'Ser
vice Pack 3') BOOST_LIB_VERSION=1_35
Sun Apr 10 22:34:09 [initandlisten] waiting for connections on port 27017
```

```
Sun Apr 10 22:34:09 [websvr] web admin interface listening on port 28017
```

MongoDB 服务端的默认连接端口是 27017

(5)、将 MongoDB 作为 Windows 服务随机启动

先创建 D:\mongodb\logs\mongodb.log 文件，用于存储 MongoDB 的日志文件，再安装系统服务：

```
D:\mongodb\bin\mongod --dbpath=d:\data\db --logpath=d:\mongodb\logs\mongodb.log --install
D:\mongodb\bin>D:\mongodb\bin\mongod --dbpath=d:\data\db --logpath=d:\mongodb\lo
gs\mongodb.log --install
all output going to: d:\mongodb\logs\mongodb.log
Creating service MongoDB.
Service creation successful.
Service can be started from the command line via 'net start "MongoDB"'.
D:\mongodb\bin>net start mongodb
Mongo DB 服务已经启动成功。
D:\mongodb\bin>
```

(6)、客户端连接验证

新打开一个 CMD 输入：d:\mongodb\bin\mongo，如果出现下面提示，那么您就可以开始 MongoDB 之旅了：

```
D:\mongodb\bin>d:\mongodb\bin\mongo
MongoDB shell version: 1.8.1
connecting to: test
>
```

(7)、查看 MongoDB 日志

查看 D:\mongodb\logs\mongodb.log 文件，即可对 MongoDB 的运行情况进行查看或排错了：

这样就完成了 Windows 平台的 MongoDB 安装。

2.2.3 Linux版安装

(1)、下载 MongoDB

curl -O http://fastdl.mongodb.org/linux/mongodb-linux-i686-1.8.1.tgz

(2)、设置 MongoDB 目录

将其解压到/Apps，再重命名为 mongodb，路径为/Apps/mongodb

(3)、设置数据文件路径

建立/data/db 的目录，mkdir -p /data/db

(4)、启动 MongoDB 服务

/Apps/mongodb/bin/mongod --dbpath=/data/db

```
[root@localhost ~]# /Apps/mongodb/bin/mongod --dbpath=/data/db
Sun Apr 8 22:41:06 [initandlisten] MongoDB starting : pid=13701 port=27017 dbpa
th=/data/db 32-bit
** 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
**    with --dur, the limit is lower
Sun Apr 8 22:41:06 [initandlisten] db version v1.8.1, pdfile version 4.5
Sun Apr 8 22:41:06 [initandlisten] git version: a429cd4f535b2499cc4130b06ff7c26
f41c00f04
Sun Apr 8 22:41:06 [initandlisten] build sys info: Linux bs-linux32.10gen.cc 2.
6.21.7-2.fc8xen #1 SMP Fri Feb 15 12:39:36 EST 2008 i686 BOOST_LIB_VERSION=1_37
Sun Apr 8 22:41:06 [initandlisten] waiting for connections on port 27017
Sun Apr 8 22:41:06 [websvr] web admin interface listening on port 28017
```

MongoDB 服务端的默认连接端口是 27017

(5)、将 MongoDB 作为 Linux 服务随机启动

先创建/Apps/mongodb/logs/mongodb.log 文件，用于存储 MongoDB 的日志文件

vi /etc/rc.local，使用 vi 编辑器打开配置文件，并在其中加入下面一行代码

/Apps/mongodb/bin/mongod --dbpath=/data/db
--logpath=/Apps/mongodb/logs/mongodb.log

(6)、客户端连接验证

新打开一个 Session 输入：/Apps/mongodb/bin/mongo，如果出现下面提示，那么您就可以开始 mongo 之旅了

```
[root@localhost ~]# /Apps/mongodb/bin/mongo
MongoDB shell version: 1.8.1
connecting to: test
>
```

(7)、查看 MongoDB 日志

查看/Apps/mongodb/logs/mongodb.log 文件，即可对 MongoDB 的运行情况进行查看或排错了

```
[root@localhost logs]# pwd
```

```
/Apps/mongodb/logs
[root@localhost logs]# ll
总计 0
-rw-r--r-- 1 root root 0 04-08 20:15 mongodb.log
[root@localhost logs]#
```

这样就完成了 Linux 平台的 MongoDB 安装。

- 也可以将 mongoDB 服务加入随机启动
  vi /etc/rc.local
  使用 vi 编辑器打开配置文件，并在其中加入下面一行代码
  /home/liwei/mongodb/mongodb-linux-i686-1.6.5/bin/mongod -dbpath=/data/mongodb/db -logpath=/data/mongodb/log/MongoDB.log --logappend

3 MongoDB客户端访问

3.1 Shell方式

MongoDB 是 MongoDB 自带的交互式 Javascript shell，用来对 MongoDB 进行操作和管理的交互式环境。

使用 "./mongo --help" 可查看相关连接参数，下面将从常见的操作，如插入，查询，修改，删除等几个方面阐述 MongoDB shell 的用法

1、插入记录

```
> use my_mongodb
switched to db my_mongodb
> db.user.insert({uid:1,username:"Tom",age:25});
> db.user.insert({uid:2,username:"Jerry",age:25});
>
```

本例向数据库 my_mongodb 的表 user 中插入了 2 条记录。MongoDB 会隐式的创建数据库 my_mongodb 和表 user，所以这个例子没有建库和建表的过程，可以通过 show dbs 和 show collections 来查看数据库及表，具体如下：

```
> show dbs
admin  (empty)
local  (empty)
my_mongodb   0.0625GB   ---隐式创建的数据库
> show collections
system.indexes
```

```
user                ---隐式创建的表
>
```

2、查询记录

查询表中的全部记录：

```
> db.user.find();
{ "_id" : ObjectId("4f81a49b779282ca68fd8a59"), "uid" : 1, "username" : "Tom",
"age" : 25 }
{ "_id" : ObjectId("4f81a4a1779282ca68fd8a5a"), "uid" : 2, "username" : "Jerry
", "age" : 25 }
>
```

查询用户名是 ”Jerry” 记录：

```
> db.user.find({username:"Jerry"});
{ "_id" : ObjectId("4f81a4a1779282ca68fd8a5a"), "uid" : 2, "username" : "Jerry
", "age" : 25 }
>
```

3、修改记录

将用户 ID 是 2 的记录的年龄修改为 100：

```
> db.user.update({uid:2},{$set:{age:100}}) ;
>
```

查询一下是否改过来了：

```
> db.user.find({uid:2});
{ "_id" : ObjectId("4f81a4a1779282ca68fd8a5a"), "uid" : 2, "username" : "Jerry
", "age" : 100 }
>
```

4、删除记录

将用户 ID 是 1 的记录从表 user 中删除：

```
> db.user.remove({uid:1});
> db.user.find();
{ "_id" : ObjectId("4f81a4a1779282ca68fd8a5a"), "uid" : 2, "username" : "Jerry
", "age" : 100 }
>
```

经验证，该记录确实被删除了。

3.2 JAVA方式

【IT168 技术】MongoDB 官网：http://www.mongodb.org/

MongoDB J 驱动：http://www.mongodb.org/display/DOCS/Drivers

MongoDB Java 文档：http://www.mongodb.org/display/DOCS/Java+Tutorial

1、MongoDB 连接：

连接数据库如果不存在，会自动生成数据库。

```
import com.mongodb.Mongo;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.BasicDBObject;
import com.mongodb.DBObject;
import com.mongodb.DBCursor;
Mongo m = new Mongo();
// or
Mongo m = new Mongo("localhost");
// or
Mongo m = new Mongo("localhost", 27017);
DB db = m.getDB("mydb");
```

设置用户名密码连接：有就返回 true，否则返回 false。

```
boolean auth = db.authenticate(myUserName, myPassword);
```

2、MongoDB 获取所有的数据库

```
Mongo m = new Mongo();
for (String s : m.getDatabaseNames()){
System.out.println(s);
}
```

删除数据库：

```
Mongo m = new Mongo();
m.dropDatabase("myDatabaseName");
```

3、MongoDB 表

获取数据库的所有表：

```
Set<String> colls = db.getCollectionNames();
for (String s : colls) {
System.out.println(s);
}
```

获取表的索引：

```
List<DBObject> list = coll.getIndexInfo();
for (DBObject o : list) {
System.out.println(o);
}
```

获取表的所有记录：

```
DBCursor cur = coll.find();
while(cur.hasNext()) {
System.out.println(cur.next());
```

```
}
```

获取表的记录数量：

```
coll.find().count();
coll.find(new BasicDBObject("i", 26)).count(); //（可以在find里添加条件 ）
```

获取查询的第一条记录：

```
DBObject firstDoc = coll.findOne();
//findOne() 返回一个记录，而find()返回的是DBCursor游标对象。
```

4、查询

条件查询：

```
BasicDBObject condition = new BasicDBObject();
condition.put("name", "bruce");
condition.put("age", 26);
coll.find(condition);
```

分页查询：

```
DBCursor cursor = coll.find().skip(0).limit(10);
while(cursor.hasNext()) {
System.out.println(cursor.next());
}
```

条件查询：

```
BasicDBObject condition = new BasicDBObject();
condition.put("age", new BasicDBObject("$gt", 50));
coll.find(condition);
比较符
"$gt"：  大于
"$gte"：大于等于
"$lt"：  小于
"$lte"：小于等于
"$in"：  包含
//以下条件查询 20<age<=30
condition.put("age", new BasicDBObject("$gt", 20).append("$lte", 30));
```

$exists

用来判断一个元素是否存在，如：

```
db.things.find( {a:{$exists:true}}); // 如果存在元素 a,就返回 true
```

$type

基于 bson type 来匹配一个元素的类型，像是按照类型 ID 来匹配，不过我没找到 bson 类型和 id 对照表。

```
db.things.find({a:{$type:2}}); // matches if a is a string
```

正则表达式

查询所有名字匹配 /joh?n/i 的记录。

```
Pattern pattern = Pattern.compile("joh?n", CASE_INSENSITIVE);
BasicDBObject query = new BasicDBObject("name", pattern);
DBCursor cursor = coll.find(query);
```

5、数据插入

批量插入：

```
List datas = new ArrayList();
for (int i=0; i < 100; i++) {
BasicDBObject bo = new BasicDBObject();
bo.put("name", "bruce");
bo.append("age", i);
datas.add(bo);
}
coll.insert(datas);
```

单个插入：

```
BasicDBObject doc = new BasicDBObject();
doc.put("name", "MongoDB");
doc.put("type", "database");
doc.put("count", 1);
BasicDBObject info = new BasicDBObject();
info.put("x", 203);
info.put("y", 102);
doc.put("info", info);
coll.insert(doc);
```

6、索引

```
coll.createIndex(new BasicDBObject("i", 1)); // create index on "i", ascending -1
 descending
```

获取所有的索引：

```
List<DBObject> list = coll.getIndexInfo();
for (DBObject o : list) {
System.out.println(o);
}
```

3.3 索引的用法

索引能提高检索数据的速度，你可以想像成在 MySQL 中创建索引一样，同样索引也是用 B-Tree 也实现的。

1.单列索引

在字段 x 上创建索引，1 (ascending) or -1 (descending)

```
> db.data.ensureIndex({x:1})
```

显示表 data 里面的所有索引

```
1.  > db.data.getIndexes()
2.  [
3.  {
4.  "name" : "_id_",
5.  "ns" : "recommender.data",
6.  "key" : {
7.  "_id" : 1
8.  }
9.  },
10. {
11. "_id" : ObjectId("4befb146b0e29ba1ce20e0bb"),
12. "ns" : "recommender.data",
13. "key" : {
14. "x" : 1
15. },
16. "name" : "x_1"
17. }
18. ]
```

查找字段 x 为 6 的值，此时已经用到索引了

```
> db.data.find({x:6})
{"_id":ObjectId("4bee804ba23d558eb6687117"), "x":6, "name":"caihuafeng1"}
{"_id":ObjectId("4bee804ba23d558eb6687118"), "x":6, "name":"caihuafeng2"}
{"_id":ObjectId("4bee804ba23d558eb6687119"), "x":6, "name":"caihuafeng3"}
{"_id":ObjectId("4bee804ba23d558eb668711a"), "x":6, "name":"caihuafeng4"}
{"_id":ObjectId("4bee804ba23d558eb668711b"), "x":6, "name":"caihuafeng5"}
{"_id":ObjectId("4bee804ba23d558eb668711c"), "x":6, "name":"caihuafeng6"}
{"_id":ObjectId("4bee804ba23d558eb668711d"), "x":6, "name":"caihuafeng7"}
{"_id":ObjectId("4bee804ba23d558eb668711e"), "x":6, "name":"caihuafeng8"}
{"_id":ObjectId("4bee804ba23d558eb668711f"), "x":6, "name":"caihuafeng9"}
{"_id":ObjectId("4bee804ba23d558eb6687120"), "x":6, "name":"caihuafeng10"}
```

2.默认索引

上述 1 中 db.data.getIndexes()显示出来的一共有 2 个索引，其中_id 是创建表的时候自动创建的索引，此索引是不能够删除的。

An index is always created on _id. This index is special and cannot be deleted. The _id index enforces uniqueness for its keys.

3.文档作为索引的键值

a.单列索引

MongoDB 的官方文档上面是这样说的:

Documents as Keys

Indexed fields may be of any type, including documents:

往数据库 recommender 的表 data 中插入三条记录

```
> db.data.insert({name:"1616",info:{url:"http://www.1616.net/",city:"beijing"}});
> db.data.insert({name:"hao123",info:{url:"http://www.hao123.com/",city:"beijing"}});
> db.data.insert({name:"114la",info:{url:"http://www.114la.com/",city:"dongguan"}});
```

对字段 info 创建索引

```
> db.data.ensureIndex({info: 1});
```

显示表 data 上的所有索引

```
> db.data.getIndexes();
[
{
"name" : "_id_",
"ns" : "recommender.data",
"key" : {
"_id" : 1
}
```

```
},
{
"_id" : ObjectId("4befb146b0e29ba1ce20e0bb"),
"ns" : "recommender.data",
"key" : {
"x" : 1
},
"name" : "x_1"
},
{
"_id" : ObjectId("4befb76bb0e29ba1ce20e0bf"),
"ns" : "recommender.data",
"key" : {
"info" : 1
},
"name" : "info_1"
}
]
```

查找指定的记录，此时会用到索引

```
> db.data.find({info: {url:"http://www.1616.net/",city:"beijing"}});
{ "_id" : ObjectId("4befb711b0e29ba1ce20e0bc"), "name" : "1616", "info
" : { "url" : "http://www.1616.net/", "city" : "beijing" } }
```

b.组合索引

建立组合索引

```
> db.data.ensureIndex({"info.url":1, "info.city":1});
> db.data.getIndexes();
[
{
"name" : "_id_",
"ns" : "recommender.data",
"key" : {
"_id" : 1
}
},
```

```
{
"_id" : ObjectId("4befb146b0e29ba1ce20e0bb"),
"ns" : "recommender.data",
"key" : {
"x" : 1
},
"name" : "x_1"
},
{
"_id" : ObjectId("4befb76bb0e29ba1ce20e0bf"),
"ns" : "recommender.data",
"key" : {
"info" : 1
},
"name" : "info_1"
},
{
"_id" : ObjectId("4befb9d1b0e29ba1ce20e0c0"),
"ns" : "recommender.data",
"key" : {
"info.url" : 1,
"info.city" : 1
},
"name" : "info.url_1_info.city_1"
}
]
```

下面几个操作均会用到索引

```
> db.data.find({"info.url": "http://www.1616.net/", "info.city": "beijing
"});
{ "_id" : ObjectId("4befb711b0e29ba1ce20e0bc"), "name" : "1616", "info
" : { "url" : "http://www.1616.net/", "city" : "beijing" } }

> db.data.find({"info.url": "http://www.1616.net/"});
{ "_id" : ObjectId("4befb711b0e29ba1ce20e0bc"), "name" : "1616", "info
" : { "url" : "http://www.1616.net/", "city" : "beijing" } }
```

1 表示升序(asc)，-1 表示降序(desc)

```
> db.data.find({"info.url": /http:*/i}).sort({"info.url": 1, "info.city
": 1});
{ "_id" : ObjectId("4befb740b0e29ba1ce20e0be"), "name" : "114la", "info
" : { "url" : "http://www.114la.com/", "city" : "dongguan" } }
{ "_id" : ObjectId("4befb711b0e29ba1ce20e0bc"), "name" : "1616", "info
" : { "url" : "http://www.1616.net/", "city" : "beijing" } }
{ "_id" : ObjectId("4befb723b0e29ba1ce20e0bd"), "name" : "hao123", "info
" : { "url" : "http://www.hao123.com/", "city" : "beijing" } }

> db.data.find({"info.url": /http:*/i}).sort({"info.url": 1});
{ "_id" : ObjectId("4befb740b0e29ba1ce20e0be"), "name" : "114la", "info
" : { "url" : "http://www.114la.com/", "city" : "dongguan" } }
{ "_id" : ObjectId("4befb711b0e29ba1ce20e0bc"), "name" : "1616", "info
" : { "url" : "http://www.1616.net/", "city" : "beijing" } }
{ "_id" : ObjectId("4befb723b0e29ba1ce20e0bd"), "name" : "hao123", "info
" : { "url" : "http://www.hao123.com/", "city" : "beijing" } }

> db.data.find({"info.url": /http:*/i}).sort({"info.url": -1});
{ "_id" : ObjectId("4befb723b0e29ba1ce20e0bd"), "name" : "hao123", "info
" : { "url" : "http://www.hao123.com/", "city" : "beijing" } }
{ "_id" : ObjectId("4befb711b0e29ba1ce20e0bc"), "name" : "1616", "info
" : { "url" : "http://www.1616.net/", "city" : "beijing" } }
{ "_id" : ObjectId("4befb740b0e29ba1ce20e0be"), "name" : "114la", "info
" : { "url" : "http://www.114la.com/", "city" : "dongguan" } }
```

4.组合索引

注意，这里的组合索引与上述 3 中的 b 中的组合索引是有点不同的，4 里面是对一级字段建立组合索引，而上述 3 中是对二级字段建立组合索引。

在字段 name 及 info 上面创建组合索引

```
> db.data.ensureIndex({name: 1, info: -1});
```

当创建组合索引时，字段后面的 1 表示升序，-1 表示降序，是用 1 还是用-1 主要是跟排序的时候或指定范围内查询的时候有关的，具体看下面的英文原文的说明。

When creating an index, the number associated with a key specifies the direction of the index, so it should always be 1 (ascending) or -1 (descending). Direction doesn't matter for single key indexes or for random access retrieval but is important if you are doing sorts or range queries on compound indexes.

显示所有的索引

```
1.  > db.data.getIndexes();
2.  [
3.  {
4.  "name" : "_id_",
5.  "ns" : "recommender.data",
6.  "key" : {
7.  "_id" : 1
8.  }
9.  },
10. {
11. "_id" : ObjectId("4befb146b0e29ba1ce20e0bb"),
12. "ns" : "recommender.data",
13. "key" : {
14. "x" : 1
15. },
16. "name" : "x_1"
17. },
18. {
19. "_id" : ObjectId("4befb76bb0e29ba1ce20e0bf"),
20. "ns" : "recommender.data",
21. "key" : {
22. "info" : 1
23. },
24. "name" : "info_1"
25. },
26. {
27. "_id" : ObjectId("4befb9d1b0e29ba1ce20e0c0"),
28. "ns" : "recommender.data",
29. "key" : {
30. "info.url" : 1,
31. "info.city" : 1
32. },
33. "name" : "info.url_1_info.city_1"
34. },
35. {
```

```
36.  "_id" : ObjectId("4befbfcfb0e29ba1ce20e0c1"),
37.  "ns" : "recommender.data",
38.  "key" : {
39.  "name" : 1,
40.  "info" : -1
41.  },
42.  "name" : "name_1_info_-1"
43.  }
44.  ]
```

下面的排序将用到上面的索引

最后一行的“name”：“ll4la”实际上是”name”：“114la”(就是将数字一写成了字母l)，但是我录入的时候写成了“name”：“ll4la”，是我写错了，但是排序的结果是对的。

```
> db.data.find({"info.url": /http:*/i}).sort({name:1, info: -1});
{ "_id" : ObjectId("4befb711b0e29ba1ce20e0bc"), "name" : "1616", "info
" : { "url" : "http://www.1616.net/", "city" : "beijing" } }
{ "_id" : ObjectId("4befb723b0e29ba1ce20e0bd"), "name" : "hao123", "info
" : { "url" : "http://www.hao123.com/", "city" : "beijing" } }
{ "_id" : ObjectId("4befb740b0e29ba1ce20e0be"), "name" : "ll4la", "info
" : { "url" : "http://www.114la.com/", "city" : "dongguan" } }
```

MongoDB 组合索引规则

If you have a compound index on multiple fields, you can use it to query on the beginning subset of fields. So if you have an index on

a,b,c

you can use it query on

a

a,b

a,b,c

如果用过 MySQL 的话，看起来是不是很熟悉，原理跟 MySQL 是一样的。

5.唯一索引

往表 data 中插入一条记录。

```
1.    > db.data.insert({firstname: "cai", lastname: "huafeng"});
```

由于表 data 中只有一记录有字段 firstname 及 lastname，其它的行均没有相应的值，也就是均为 null，为 null 就说明是相同的，而唯一索引是不允许有相同的值的，所以下面创建唯一组合索引时报错了。

所以建立唯一索引时，不管是对单个字段还是多个字段建立索引，则最好每一行均有此字段，否则会报错。

```
> db.data.find();
{ "_id" : ObjectId("4bee745a0863b1c233b8b7ea"), "name" : "caihuafeng" }
{ "_id" : ObjectId("4bee745f0863b1c233b8b7eb"), "website" : "1616.net" }
{ "_id" : ObjectId("4bee804ba23d558eb6687117"), "x" : 6, "name" : "caihuafeng1" }

{ "_id" : ObjectId("4bee804ba23d558eb6687118"), "x" : 6, "name" : "caihuafeng2" }

{ "_id" : ObjectId("4bee804ba23d558eb6687119"), "x" : 6, "name" : "caihuafeng3" }

{ "_id" : ObjectId("4bee804ba23d558eb668711a"), "x" : 6, "name" : "caihuafeng4" }

{ "_id" : ObjectId("4bee804ba23d558eb668711b"), "x" : 6, "name" : "caihuafeng5" }

{ "_id" : ObjectId("4bee804ba23d558eb668711c"), "x" : 6, "name" : "caihuafeng6" }

{ "_id" : ObjectId("4bee804ba23d558eb668711d"), "x" : 6, "name" : "caihuafeng7" }

{ "_id" : ObjectId("4bee804ba23d558eb668711e"), "x" : 6, "name" : "caihuafeng8" }

{ "_id" : ObjectId("4bee804ba23d558eb668711f"), "x" : 6, "name" : "caihuafeng9" }

{ "_id" : ObjectId("4bee804ba23d558eb6687120"), "x" : 6, "name" : "caihuafeng10" }

{ "_id" : ObjectId("4befb711b0e29ba1ce20e0bc"), "name" : "1616", "info" : { "url
" : "http://www.1616.net/", "city" : "beijing" } }
```

```
{ "_id" : ObjectId("4befb723b0e29ba1ce20e0bd"), "name" : "hao123", "info" : { "url
" : "http://www.hao123.com/", "city" : "beijing" } }
{ "_id" : ObjectId("4befb740b0e29ba1ce20e0be"), "name" : "114la", "info" : { "url
" : "http://www.114la.com/", "city" : "dongguan" } }
{ "_id" : ObjectId("4befc51ab0e29ba1ce20e0c2"), "firstname" : "cai", "lastname
" : "huafeng" }

> db.data.ensureIndex({firstname: 1, lastname: 1}, {unique: true});
E11000 duplicate key error index: recommender.data.$firstname_1_lastname_1 dup ke
y: { : null, : null }
```

下面我们用另外一个表 person 来进行测试

```
1.   > db.person.ensureIndex({firstname:1, lastname: 1},{unique: true});
2.   > db.person.insert({firstname: 'cai', lastname: 'huafeng'});
```

第二次插入同样值的时候报错了，说明唯一索引生效了，其实跟 MySQL 里面是一样的。

```
> db.person.insert({firstname: 'cai', lastname: 'huafeng'});
E11000 duplicate key error index: recommender.person.$firstname_1_lastnam
e_1 dup key: { : "cai", : "huafeng" }
```

6.唯一索引中的重复值处理

删除上述 5 中的索引，插入两行一样的记录

```
> db.person.dropIndexes();
{
"nIndexesWas" : 2,
"msg" : "non-_id indexes dropped for collection",
"ok" : 1
}
> db.person.find();
{ "_id" : ObjectId("4befcda6b0e29ba1ce20e0cf"), "firstname" : "cai", "las
tname" : "huafeng" }
> db.person.insert({firstname: 'cai', lastname: 'huafeng'});
> db.person.find();
```

```
{ "_id" : ObjectId("4befcda6b0e29ba1ce20e0cf"), "firstname" : "cai", "las
tname" : "huafeng" }
{ "_id" : ObjectId("4befcef0b0e29ba1ce20e0d1"), "firstname" : "cai", "las
tname" : "huafeng" }
```

如果现在直接在字段 firstname 及 lastname 上面创建唯一组合索引的时候肯定会报错，我们来试一试:

```
> db.person.ensureIndex({firstname: 1, lastname: 1}, {unique: true});
E11000 duplicate key error index: recommender.person.$firstname_1_lastname_1 du
p key: { : "cai", : "huafeng" }
```

查看表 person 的索引，我们可以看到，新创建的索引没有生成。

```
> db.person.getIndexes();
[
{
"name" : "_id_",
"ns" : "recommender.person",
"key" : {
"_id" : 1
}
}
]
```

可以在第二个 json 对象加入一项 dropDups: true，这样在创建唯一组合索引的时候不会报错，保留文档中第一个重复的值，其它重复的值均删除。

再次测试一下，加入 dropDups 选项，虽然报错了，但是唯一组合索引已经建立了。

```
> db.person.ensureIndex({firstname: 1, lastname: 1}, {unique: true, dropDups: true});

E11000 duplicate key error index: recommender.person.$firstname_1_lastname_1 dup ke
y: { : "cai", : "huafeng" }
> db.person.getIndexes();
[
{
```

```
"name" : "_id_",
"ns" : "recommender.person",
"key" : {
"_id" : 1
}
},
{
"_id" : ObjectId("4befcfd9b0e29ba1ce20e0d3"),
"ns" : "recommender.person",
"key" : {
"firstname" : 1,
"lastname" : 1
},
"name" : "firstname_1_lastname_1",
"unique" : true,
"dropDups" : true
}
]
```

再次查询表 person 中的记录，发现重复的记录已经自动删除了。

```
> db.person.find();
{ "_id" : ObjectId("4befcda6b0e29ba1ce20e0cf"), "firstname" : "cai", "las
tname" : "huafeng" }
```

MongoDB 官方文档的说明

A unique index cannot be created on a key that has duplicate values. If you would like to create the index anyway, keeping the first document the database indexes and deleting all subsequent documents that have duplicate values, add the dropDups option.

```
db.things.ensureIndex({firstname : 1}, {unique : true, dropDups : true})
```

7.删除索引

a.删除某个表中的所有索引

To delete all indexes on the specified collection:

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

> db.data.dropIndex({firstname: 1, lastname: 1});
{ "nIndexesWas" : 6, "ok" : 1 }
```

Running directly as a command without helper:

```
// note: command was "deleteIndexes", not "dropIndexes", before MongoDB v
1.3.2
// remove index with key pattern {y:1} from collection foo
db.runCommand({dropIndexes:'foo', index : {y:1}})
// remove all indexes:
db.runCommand({dropIndexes:'foo', index : '*'})

> db.person.ensureIndex({firstname: 1, lastname: 1});

> db.runCommand({dropIndexes:'person', index:{firstname:1, lastname:1}});

{ "nIndexesWas" : 2, "ok" : 1 }
```

4 MySQL与MongoDB设计实例对比

【IT168 技术】MySQL 是关系型数据库中的明星，MongoDB 是文档型数据库中的翘楚。下面通过一个设计实例对比一下二者：假设我们正在维护一个手机产品库，里面除了包含手机的名称，品牌等基本信息，还包含了待机时间，外观设计等参数信息，应该如何存取数据呢？

如果使用 MySQL 的话，应该如何存取数据呢？

如果使用 MySQL 话，手机的基本信息单独是一个表，另外由于不同手机的参数信息差异很大，所以还需要一个参数表来单独保存。

```
CREATE TABLE IF NOT EXISTS `mobiles` (

`id` int(10) unsigned NOT NULL AUTO_INCREMENT,

`name` VARCHAR(100) NOT NULL,

`brand` VARCHAR(100) NOT NULL,
```

```
PRIMARY KEY (`id`)
);
CREATE TABLE IF NOT EXISTS `mobile_params` (
`id` int(10) unsigned NOT NULL AUTO_INCREMENT,
`mobile_id` int(10) unsigned NOT NULL,
`name` varchar(100) NOT NULL,
`value` varchar(100) NOT NULL,
PRIMARY KEY (`id`)
);
INSERT INTO `mobiles` (`id`, `name`, `brand`) VALUES
(1,'ME525','摩托罗拉'),
(2,'E7' ,'诺基亚');
INSERT INTO `mobile_params` (`id`, `mobile_id`, `name`, `value`) VALUES
(1, 1,'待机时间','200'),
(2, 1,'外观设计','直板'),
(3, 2,'待机时间','500'),
(4, 2,'外观设计','滑盖');
```

注：为了演示方便，没有严格遵守关系型数据库的范式设计。

如果想查询待机时间大于 100 小时，并且外观设计是直板的手机，需要按照如下方式查询：

```
SELECT * FROM `mobile_params` WHERE name = '待机时间' AND value > 100;

SELECT * FROM `mobile_params` WHERE name = '外观设计' AND value = '直板';
```

注：参数表为了方便，把数值和字符串统一保存成字符串，实际使用时，MySQL 允许在字符串类型的字段上进行数值类型的查询，只是需要进行类型转换，多少会影响一点性能。

两条 SQL 的结果取交集得到想要的 MOBILE_IDS，再到 mobiles 表查询即可：

```
SELECT * FROM `mobiles` WHERE mobile_id IN (MOBILE_IDS)
```

如果使用 MongoDB 的话，应该如何存取数据呢?

如果使用 MongoDB 的话，虽然理论上可以采用和 MySQL 一样的设计方案，但那样的话就显得无趣了，没有发挥出 MongoDB 作为文档型数据库的优点，实际上使用 MongoDB 的话，和 MySQL 相比，形象一点来说，可以合二为一：

```
db.getCollection("mobiles").ensureIndex({
"params.name": 1,
"params.value": 1
});

db.getCollection("mobiles").insert({
"_id": 1,
"name": "ME525",
"brand": "摩托罗拉",
"params": [
{"name": "待机时间", "value": 200},
{"name": "外观设计", "value": "直板"}
]
});

db.getCollection("mobiles").insert({
"_id": 2,
"name": "E7",
"brand": "诺基亚",
"params": [
{"name": "待机时间", "value": 500},
{"name": "外观设计", "value": "滑盖"}
]
});
```

如果想查询待机时间大于 100 小时，并且外观设计是直板的手机，需要按照如下方式查询：

```
db.getCollection("mobiles").find({
"params": {
$all: [
{$elemMatch: {"name": "待机时间", "value": {$gt: 100}}},
{$elemMatch: {"name": "外观设计", "value": "直板"}}
]
}
});
```

注：查询中用到的$all，$elemMatch 等高级用法的详细介绍请参考官方文档中相关说明。

MySQL 需要多个表，多次查询才能搞定的问题，MongoDB 只需要一个表，一次查询就能搞定，对比完成，相对 MySQL 而言，MongoDB 显得更胜一筹，至少本例如此

5 内部结构分析

对于大多数的 MongoDB 的用户来说，MongoDB 就像是一个大黑盒，但是如果你能够了解到 MongoDB 内部一些构造的话，将有利于你更好地理解和使用 MongoDB。

5.1 BSON

在 MongoDB 中，文档是对数据的抽象，它被使用在 Client 端和 Server 端的交互中。所有的 Client 端（各种语言的 Driver）都会使用这种抽象，它的表现形式就是我们常说的 BSON（Binary JSON ）。

BSON 是一个轻量级的二进制数据格式。MongoDB 能够使用 BSON，并将 BSON 作为数据的存储存放在磁盘中。

当 Client 端要将写入文档，使用查询等等操作时，需要将文档编码为 BSON 格式，然后再发送给 Server 端。同样，Server 端的返回结果也是编码为 BSON 格式再放回给 Client 端的。

使用 BSON 格式出于以下 3 种目的：

1） 效率

BSON 是为效率而设计的，它只需要使用很少的空间。即使在最坏的情况下，BSON 格式也比 JSON 格式再最好的情况下存储效率高。

2） 传输性

在某些情况下，BSON 会牺牲额外的空间让数据的传输更加方便。比如，字符串的传输的前缀会标识字符串的长度，而不是在字符串的末尾打上结束的标记。这样的传输形式有利于 MongoDB 修改传输的数据。

3） 性能

最后，BSON 格式的编码和解码都是非常快速的。它使用了 C 风格的数据表现形式，这样在各种语言中都可以高效地使用。

更多关于 BSON 的介绍，可以参考：http://www.bsonspec.org。

5.2 写入协议

Client 端访问 Server 端使用了轻量级的 TCP/IP 写入协议。这种协议在 MongoDB Wiki 中有详细介绍，它其实是在 BSON 数据上面做了一层简单的包装。比如说，写入数据的命令中包含了 1 个 20 字节的消息头（由消息的长度和写入命令标识组成），需要写入的 Collection 名称和需要写入的数据。

5.3 数据文件

在 MongoDB 的数据文件夹中（默认路径是/data/db）由构成数据库的所有文件。每一个数据库都包含一个.ns 文件和一些数据文件，其中数据文件会随着数据量的增加而变多。所以如果有一个数据库名字叫做 foo，那么构成 foo 这个数据库的文件就会由 foo.ns，foo.0，foo.1，foo.2 等等组成。

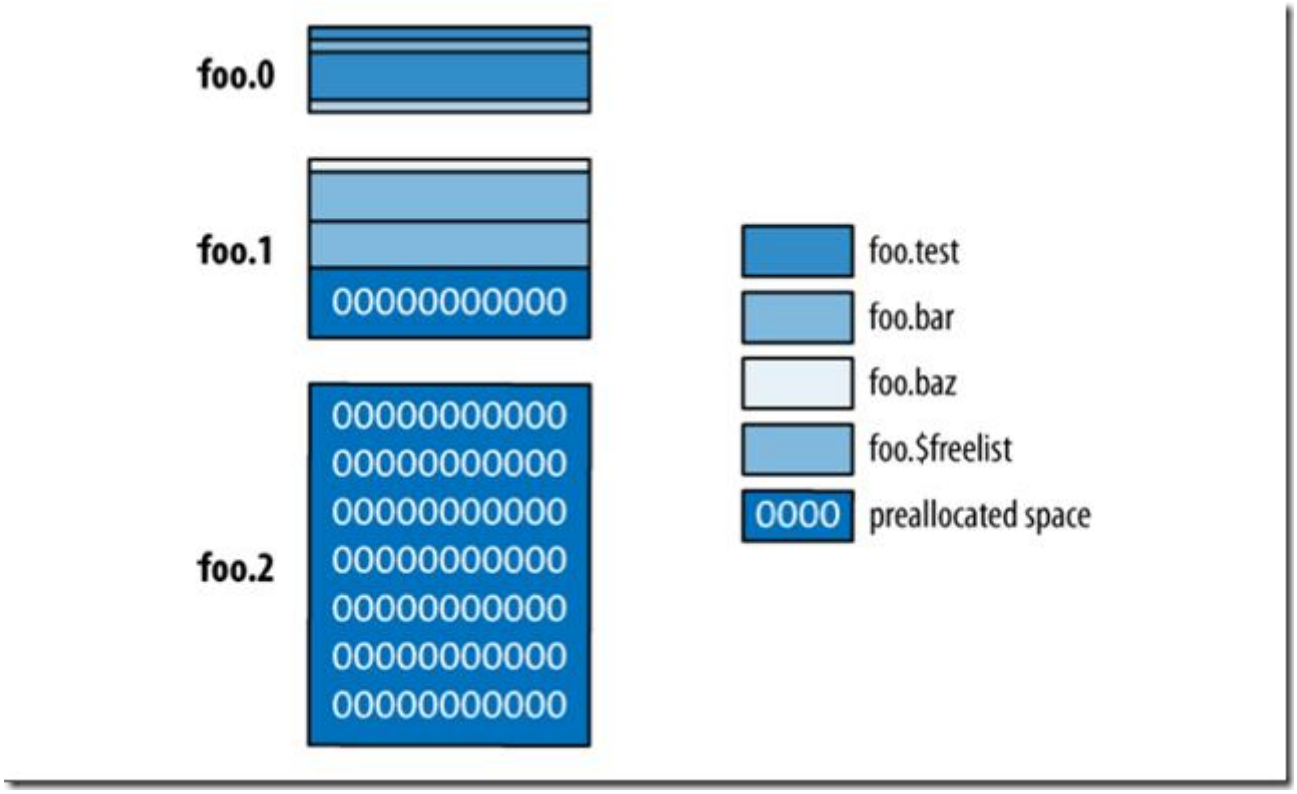
数据文件每新增一次，大小都会是上一个数据文件的 2 倍，每个数据文件最大 2G。这样的设计有利于防止数据量较小的数据库浪费过多的空间，同时又能保证数据量较大的数据库有相应的空间使用。

MongoDB 会使用预分配方式来保证写入性能的稳定（这种方式可以使用–noprealloc 关闭）。预分配在后台进行，并且每个预分配的文件都用 0 进行填充。这会让 MongoDB 始终保持额外的空间和空余的数据文件，从而避免了数据增长过快而带来的分配磁盘空间引起的阻塞。

5.4 名字空间和盘区

每一个数据库都由多个名字空间组成，每一个名字空间存储了相应类型的数据。数据库中的每一个 Collection 都有各自对应的名字空间，索引文件同样也有名字空间。所有名字空间的元数据都存储在.ns 文件中。

名字空间中的数据在磁盘中分为多个区间，这个叫做盘区。在下图中，foo 这个数据库包含 3 个数据文件，第三个数据文件属于空的预分配文件。头两个数据文件被分为了相应的盘区对应不同的名字空间。



上图显示了名字空间和盘区的相关特点。每一个名字空间可以包含多个不同的盘区，这些盘区并不是连续的。与数据文件的增长相同，每一个名字空间对应的盘区大小的也是随着分配的次数不断增长的。这样做的目的是为了平衡名字空间浪费的空间与保持某一个名字空间中数据的连续性。上图中还有一个需要注意的名字空间：$freelist，这个名字空间用于记录不再使用的盘区（被删除的 Collection 或索引）。每当名字空间需要分配新的盘区的时候，都会先查看$freelist 是否有大小合适的盘区可以使用。

5.5 内存映射存储引擎

MongoDB 目前支持的存储引擎为内存映射引擎。当 MongoDB 启动的时候，会将所有的数据文件映射到内存中，然后操作系统会托管所有的磁盘操作。这种存储引擎有以下几种特点：

* MongoDB 中关于内存管理的代码非常精简，毕竟相关的工作已经有操作系统进行托管。

* MongoDB 服务器使用的虚拟内存将非常巨大，并将超过整个数据文件的大小。不用担心，操作系统会去处理这一切。要注意的是，MongoDB 自己是不管理内存的，无法指定内存大小，完全交给操作系统来管理，因此有时候是不可控的，在生产环境使用必须在 OS 层面监控内存使用情况。

* MongoDB 无法控制数据写入磁盘的顺序，这样将导致 MongoDB 无法实现 writeahead 日志的特性。所以，如果 MongoDB 希望提供一种 durability 的特性（这一特性可以参考我写的关于 Cassandra 文章：http://www.cnblogs.com/gpcuster/tag/Cassandra/），需要实现另外一种存储引擎。

* 32 位系统的 MongoDB 服务器每一个 Mongod 实例只能使用 2G 的数据文件。这是由于地址指针只能支持 32 位。

1) 在引入mongoDB过程中，Yottaa的收获是：必须提前考虑分片；升级时复查所有慢查询，并添加合适的索引；在生产环境中，必须小心添加索引；避免慢写操作等。

6 性能测试

6.1 测试案例一

1、测试环境

MongoDB 部署在一台 PC 服务器上，配置如下：

CPU 为 Xeon 2.80GHz *4

内存为 4G

硬盘为一块 400G SATA 盘

操作系统为 64 位 CentOS 5.3 版本

2、测试方法

这里仍然采用 PHP 客户端进行测试，MongoDB 官方为 PHP 开发了一个扩展包可以操作 MongoDB，网址为 http://pecl.php.net/package/mongo，可以编译到 PHP 运行环境中来使用。

为了不对测试服务器产生额外的影响，测试客户端部署在另外一台独立的服务器上，运行的 PHP 的版本是 5.3.5，web server 是 Nginx 0.8.54，通过 fastcgi 的方式调用 PHP 服务。使用 apache ab 工具实现多个请求和并发操作。

测试过程中就使用上文提到的已经启动的数据库实例，首先是进行写操作，通过 500 个请求，每个请求写入 10000 条记录，并发度为 2 来共写入 500 万条数据，MongoDB 的数据格式是 BSON 方式，不同于其他的 NoSQL 是 key value 的方式，因此这里写入的数据格式为：{id,data}的形式，ID 为数字 1 到 5000000，data 大小为 100 个字节。然后是读操作，也是用 500 个请求，每个请求随机根据 ID 值读出 10000 条记录，并发度为 10 共读出 500 万条记录，评测的重点是写入和读出数据的时间，以及在此过程中服务器的资源使用情况。

需要说明的是，MongoDB 默认会为每个记录建立一个名字为_id 的索引，在没有索引的情况下 MongoDB 读数据都要进行全表扫描，效率还是很低的，因此在写完 500 万条记录后，我在 id 字段上建立了一个索引来加快读的过程。

6.2 测试案例二：Mongodb亿级数据量的性能测试

导读：近日，博客园作者 lovecindywang 撰写了一篇题为"Mongodb 亿级数据量的性能测试"的文章，全文如下

Mongodb 亿级数据量的性能测试，分别测试如下几个项目：

（所有插入都是单线程进行，所有读取都是多线程进行）

1） 普通插入性能 （插入的数据每条大约在 1KB 左右）

2） 批量插入性能 （使用的是官方 C#客户端的 InsertBatch），这个测的是批量插入性能能有多少提高

3） 安全插入功能 （确保插入成功，使用的是 SafeMode.True 开关），这个测的是安全插入性能会差多少

4） 查询一个索引后的数字列，返回 10 条记录（也就是 10KB）的性能，这个测的是索引查询的性能

5） 查询两个索引后的数字列，返回 10 条记录（每条记录只返回 20 字节左右的 2 个小字段）的性能，这个测的是返回小数据量以及多一个查询条件对性能的影响

6） 查询一个索引后的数字列，按照另一个索引的日期字段排序（索引建立的时候是倒序，排序也是倒序），并且 Skip100 条记录后返回 10 条记录的性能，这个测的是 Skip 和 Order 对性能的影响

7） 查询 100 条记录（也就是 100KB）的性能（没有排序，没有条件），这个测的是大数据量的查询结果对性能的影响

8） 统计随着测试的进行，总磁盘占用，索引磁盘占用以及数据磁盘占用的数量

并且每一种测试都使用单进程的 Mongodb 和同一台服务器开三个 Mongodb 进程作为 Sharding（每一个进程大概只能用 7GB 左右的内存）两种方案

其实对于 Sharding，虽然是一台机器放 3 个进程，但是在查询的时候每一个并行进程查询部分数据，再有运行于另外一个机器的 mongos 来汇总数据，理论上来说在某些情况下性能会有点提高

基于以上的种种假设，猜测某些情况性能会下降，某些情况性能会提高，那么来看一下最后的测试结果怎么样？

备注：测试的存储服务器是 E5620 @ 2.40GHz，24GB 内存，CentOs 操作系统，打压机器是 E5504 @ 2.0GHz，4GB 内存，Windows Server 2003 操作系统，两者千兆网卡直连。

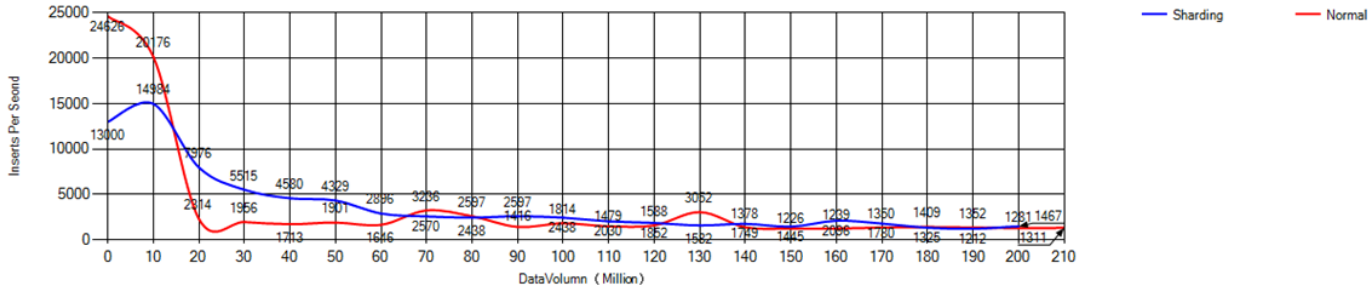
参考：

http://special.csdn.net/mongodb/index.html

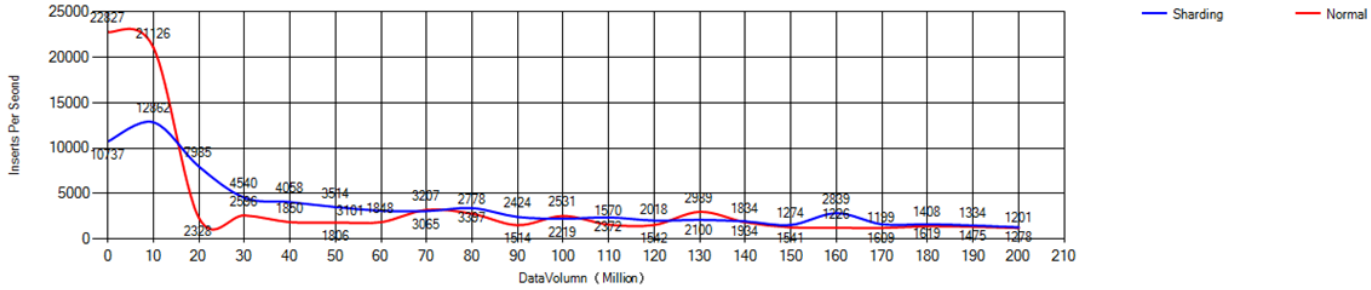
性能测试

查看Mongodb测试结果 | 查看Redis测试结果 | 查看Kt测试结果 | 查看测试进度

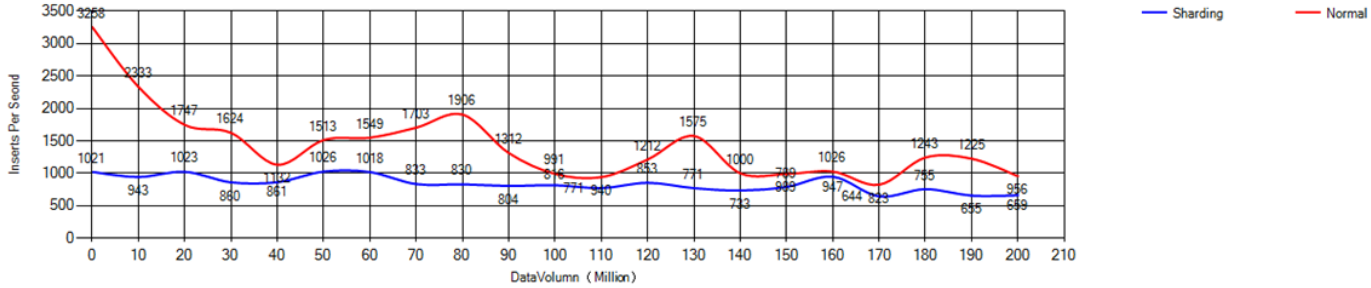
Mongodb - Performance of Normal Insert



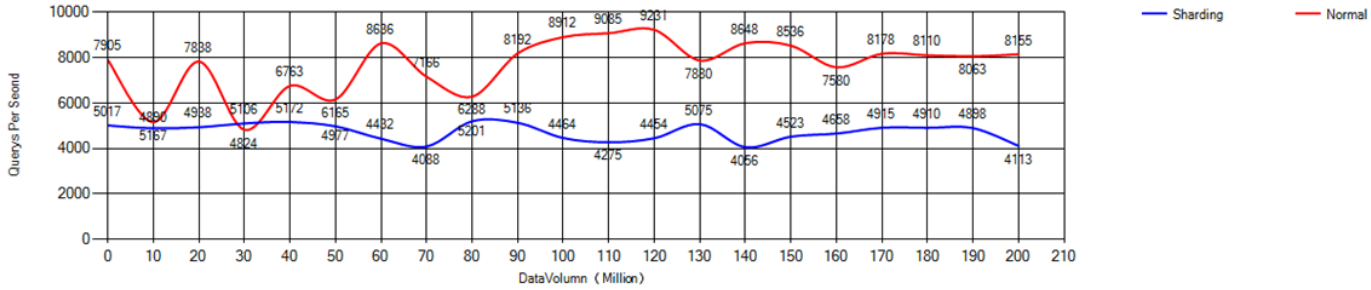
Mongodb - Performance of Batch Insert



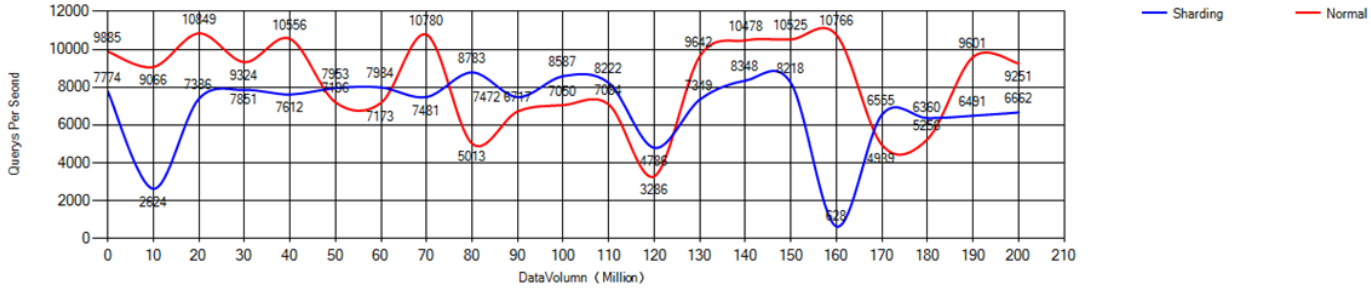
Mongodb - Performance of Safe Insert



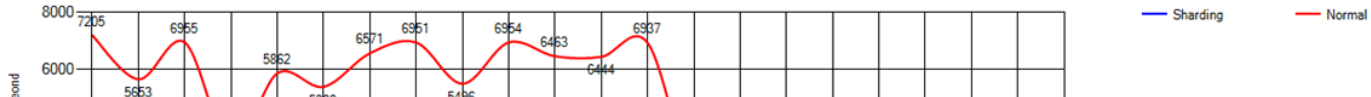
Mongodb - Performance of Query (10 Rows) with One Condition



Mongodb - Performance of Query (10 Rows) with Two Conditions and Returning Small Fields



Mongodb - Performance of Query (10 Rows) With One Condition and Skipping / Sorting



点击查看大图

从这个测试可以看出，对于单进程的方式：

1） Mongodb 的非安全插入方式，在一开始插入性能是非常高的，但是在达到了两千万条数据之后性能骤减，这个时候恰巧是服务器 24G 内存基本占满的时候（随着测试的进行 mongodb 不断占据内存，一直到操作系统的内存全部占满），也就是说 Mongodb 的内存映射方式，使得数据全部在内存中的时候速度飞快，当部分数据需要换出到磁盘上之后，性能下降很厉害。（这个性能其实也不算太差，因为我们对三个列的数据做了索引，即使在内存满了之后每秒也能插入 2MB 的数据，在一开始更是每秒插入 25MB 数据）

2） 对于批量插入功能，其实是一次提交一批数据，但是相比一次一条插入性能并没有提高多少，一来是因为网络带宽已经成为了瓶颈，二来我想写锁也会是一个原因。

3） 对于安全插入功能，相对来说比较稳定，不会波动很大，我想可能是因为安全插入是确保数据直接持久化到磁盘的，而不是插入内存就完事。

4） 对于一列条件的查询，性能一直比较稳定，别小看，每秒能有 8000-9000 的查询次数，每次返回 10KB，相当于每秒查询 80MB 数据，而且数据库记录是 2 亿之后还能维持这个水平，性能惊人。

5） 对于二列条件返回小数据的查询，总体上性能会比 4）好一点，可能返回的数据量小对性能提高比较大，但是相对来说性能波动也厉害一点，可能多了一个条件就多了一个从磁盘换页的机会。

6） 对于一列数据外加 Sort 和 Skip 的查询，在数据量大了之后性能明显就变差了（此时是索引数据量超过内存大小的时候，不知道是否有联系），我猜想是 Skip 比较消耗性能，不过和 4）相比性能也不是差距特别大。

7） 对于返回大数据的查询，一秒瓶颈也有 800 次左右，也就是 80M 数据，这就进一步说明了在有索引的情况下，顺序查询和按条件搜索性能是相差无几的，这个时候是 IO 和网络的瓶颈。

8） 在整个过程中索引占的数据量已经占到了总数据量的相当大比例，在达到 1 亿 4 千万数据量的时候，光索引就可以占据整个内存，此时查询性能还是非常高，插入性能也不算太差，mongodb 的性能确实很牛。

那么在来看看 Sharding 模式有什么亮点：

1） 非安全插入和单进程的配置一样，在内存满了之后性能急剧下降。安全插入性能和单进程相比慢不少，但是非常稳定。

2） 对于一个条件和两个条件的查询，性能都比较稳定，但条件查询性能相当于单进程的一半，但是在多条件下有的时候甚至会比单进程高一点。我想这可能是某些时候数据块位于两个 Sharding，这样 Mongos 会并行在两个 Sharding 查询，然后在把数据进行合并汇总，由于查询返回的数据量小，网络不太可能成为瓶颈了，使得 Sharding 才有出头的机会。

3） 对于 Order 和 Skip 的查询，Sharding 方式的差距就出来了，我想主要性能损失可能在 Order，因为我们并没有按照排序字段作为 Sharding 的 Key，使用的是_id 作为 Key，这样排序就比较难进行。

4） 对于返回大数据量的查询，Sharding 方式其实和单进程差距不是很大，我想数据的转发可能是一个性能损耗的原因（虽然 mongos 位于打压机本机，但是数据始终是转手了一次）。

5） 对于磁盘空间的占用，两者其实是差不多的，其中的一些差距可能是因为多个进程都会多分配一点空间，加起来有的时候会比单进程多占用点磁盘（而那些占用比单进程少的地方其实是开始的编码错误，把实际数据大小和磁盘文件占用大小搞错了）。

虽然在最后由于时间的关系，没有测到 10 亿级别的数据量，但是通过这些数据已经可以证明 Mongodb 的性能是多么强劲了。另外一个原因是，在很多时候可能数据只达到千万我们就会对库进行拆分，不会让一个库的索引非常庞大。在测试的过程中还发现几个问题需要值得注意：

1） 在数据量很大的情况下，对服务进行重启，那么服务启动的初始化阶段，虽然可以接受数据的查询和修改，但是此时性能很差，因为 mongodb 会不断把数据从磁盘换入内存，此时的 IO 压力非常大。

2） 在数据量很大的情况下，如果服务没有正常关闭，那么 Mongodb 启动修复数据库的时间非常可观，在 1.8 中退出的-dur 貌似可以解决这个问题，我简单测试了一下，开启 dur 对插入和查询性能影响都不是很大。

3） 在使用 Sharding 的时候，Mongodb 时不时会对数据拆分搬迁，这个时候性能下降很厉害，虽然从测试图中看不出（因为我每一次测试都会测试比较多的迭代次数），但是我在实际观察中可以发现，在搬迁数据的时候每秒插入性能可能会低到几百条。

4） 对于数据的插入，如果使用多线程并不会带来性能的提高，反而还会下降一点性能（并且可以在 http 接口上看到，有大量的线程处于等待）。

5） 在整个测试过程中，批量插入的时候遇到过几次连接被远程计算机关闭的错误，怀疑是有的时候 Mongodb 不稳定关闭了连接，或是官方的 C#客户端有 BUG，但是也仅仅是在数据量特别大的时候遇到几次。