MongoDB 数据库简介

ougd . ai . wangjie

MongoDB 是一个高性能，开源，无模式的文档型数据库，是当前 NoSQL 数据库产品中最热门的一种。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，MongoDB 使用 C++开发。本文介绍了 MongoDB 数据库的特点、发展史、目前应用现状、数据库存储技术、存储架构及查询、更新技术。

1 介绍

MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。他支持的数据结构非常松散，是类似 json 的 bjson 格式，因此可以存储比较复杂的数据类型。MongoDB 最大的特点是他支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。它是一个面向集合的,模式自由的文档型数据库。

1、 面向集合（Collenction-Orented）

意思是数据被分组存储在数据集中，被称为一个集合（Collenction)。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（RDBMS）里的表（table），不同的是它不需要定义任何模式（schema)。

2、 模式自由（schema-free)

意味着对于存储在 MongoDB 数据库中的文件，我们不需要知道它的任何结构定义。那么，“无模式”或“模式自由”到底是什么概念呢？例如，下面两个记录可以存在于同一个集合里面：

{"welcome" : "Beijing"}

{"age" : 25}

3、 文档型

意思是我们存储的数据是键-值对的集合，键是字符串，值可以是数据类型集合里的任意类型，包括数组和文档。我们把这个数据格式称作“BSON”即“Binary Serialized dOcument Notation.”

1.1 MongoDB 的特点

_ 面向集合存储，易于存储对象类型的数据
_ 模式自由。
_ 支持动态查询。
_ 支持完全索引，包含内部对象。
_ 支持查询。
_ 支持复制和故障恢复。
_ 使用高效的二进制数据存储，包括大型对象（如视频等）。
_ 自动处理碎片，以支持云计算层次的扩展性。
_ 支持 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。

_ 文件存储格式为 BSON（一种 JSON 的扩展）。

_ 可通过网络访问。

1.2 功能

_ 面向集合的存储：适合存储对象及 JSON 形式的数据。

_ 动态查询：MongoDB 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。

_ 完整的索引支持：包括文档内嵌对象及数组。MongoDB 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。

_ 查询监视：MongoDB 包含一系列监视工具用于分析数据库操作的性能。

_ 复制及自动故障转移：MongoDB 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。

_ 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）。

_ 自动分片以支持云级别的伸缩性：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。

1.3 适用场合

_ 网站数据：MongoDB 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。

_ 缓存：由于性能很高，MongoDB 也适合作为信息基础设施的缓存层。在系统重启之后，由 MongoDB 搭建的持久化缓存层可以避免下层的数据源过载。

_ 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。

_ 高伸缩性的场景：MongoDB 非常适合由数十或数百台服务器组成的数据库。MongoDB 的路线图中已经包含对 MapReduce 引擎的内置支持。

_ 用于对象及 JSON 数据的存储：MongoDB 的 BSON 数据格式非常适合文档化格式的存储及查询。

2 体系结构

MongoDB 是一个可移植的数据库，它在流行的每一个平台上都可以使用，即所谓的跨平台特性。在不同的操作系统上虽然略有差别，但是从整体构架上来看，MongoDB 在不同的平台上是一样的，如数据逻辑结构和数据的存储等等。

一个运行着的 MongoDB 数据库就可以看成是一个 MongoDB Server，该 Server 由实例和数据库组成，在一般的情况下一个 MongoDB Server 机器上包含一个实例和多个与之对应的数据库，但是在特殊情况下，如硬件投入成本有限或特殊的应用需求，也允许一个 Server 机器上可以有多个实例和多个数据库。

MongoDB 中一系列物理文件（数据文件，日志文件等）的集合或与之对应的逻辑结构（集合，文档等）被称为数据库，简单的说，就是数据库是由一系列与磁盘有关系的物理文件的组成。

2.1 数据逻辑结构

MongoDB 的逻辑结构是一种层次结构。主要由：文档(document)、集合

(collection)、数据库(database)这三部分组成的。逻辑结构是面向用户的，用户使用 MongoDB 开发应用程序使用的就是逻辑结构。

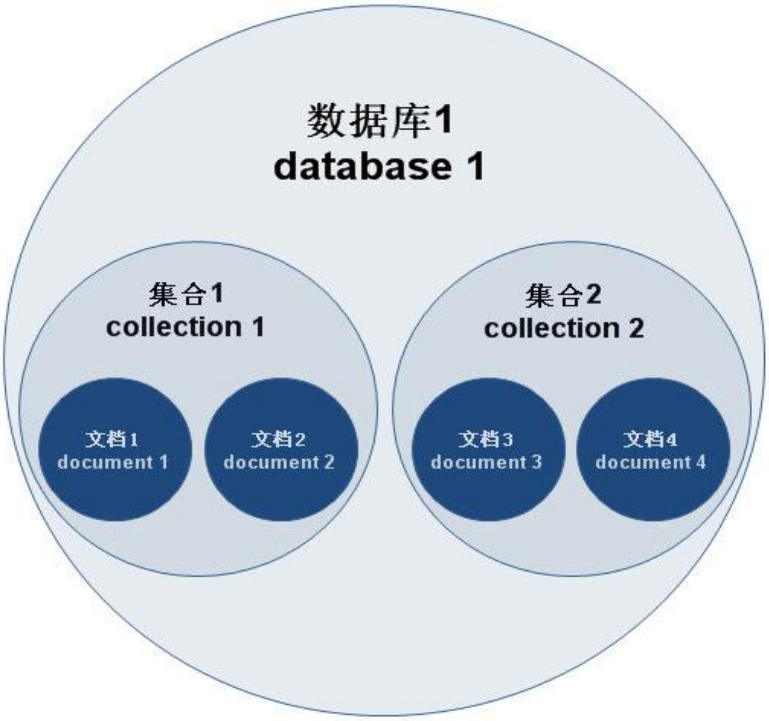
_ MongoDB 的文档（document），相当于关系数据库中的一行记录。

_ 多个文档组成一个集合（collection），相当于关系数据库的表。

_ 多个集合（collection），逻辑上组织在一起，就是数据库（database）。

_ 一个 MongoDB 实例支持多个数据库（database）。

文档(document)、集合(collection)、数据库(database)的层次结构如下图:



3.2 数据存储结构

MongoDB 对国内用户来说比较新，它就像是一个黑盒子，但是如果对于它内部的数据存储了解多一些的话，那么将会很快的理解和驾驭 MongoDB，让它发挥它更大的作用。

MongoDB 的默认数据目录是/data/db，它负责存储所有的 MongoDB 的数据文件。在 MongoDB 内部，每个数据库都包含一个.ns 文件和一些数据文件，而且这些数据文件会随着数据量的增加而变得越来越多。所以如果系统中有一个叫做 foo 的数据库，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。具体如下：

```
[root@localhost db]# ll /data/db/
总计 196844
-rw------- 1 root root 16777216 04-15 16:33 admin.0
-rw------- 1 root root 33554432 04-15 16:33 admin.1
-rw------- 1 root root 16777216 04-15 16:33 admin.ns
-rw------- 1 root root 16777216 04-21 17:30 foo.0
-rw------- 1 root root 33554432 04-21 17:30 foo.1
-rw------- 1 root root 67108864 04-21 17:30 foo.2
-rw------- 1 root root 16777216 04-21 17:30 foo.ns
```

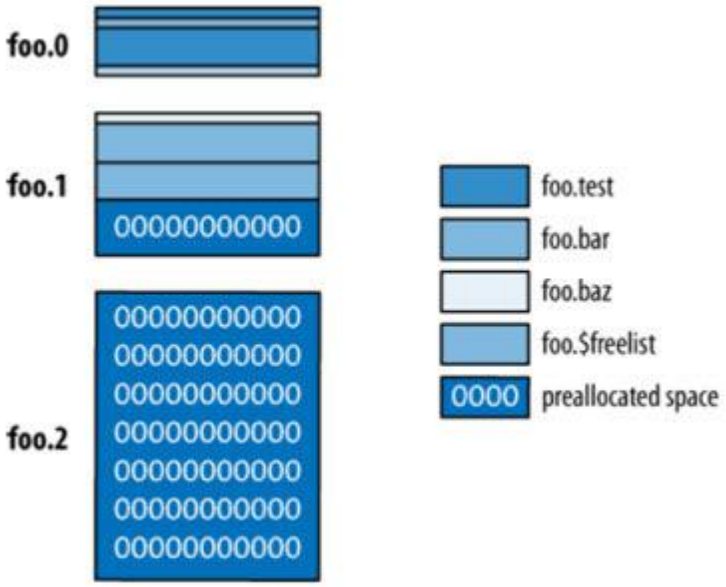
-rwxr-xr-x 1 root root 6 04-21 17:16 mongod.lock
-rw------- 1 root root 16777216 04-15 16:30 test.0
-rw------- 1 root root 33554432 04-15 16:30 test.1
-rw------- 1 root root 16777216 04-15 16:30 test.ns
drwxr-xr-x 2 root root 4096 04-21 17:30 _tmp

MongoDB 内部有预分配空间的机制，每个预分配的文件都用 0 进行填充，由于有了这个机制, MongoDB 始终保持额外的空间和空余的数据文件，从而有效避免了由于数据暴增而带来的磁盘压力过大的问题。

由于表中数据量的增加，数据文件每新分配一次，它的大小都会是上一个数据文件大小的 2 倍，每个数据文件最大 2G。这样的机制有利于防止较小的数据库浪费过多的磁盘空间，同时又能保证较大的数据库有相应的预留空间使用。

数据库的每张表都对应一个命名空间，每个索引也有对应的命名空间。这些命名空间的元数据都集中在*.ns 文件中。

在下图中，foo 这个数据库包含 3 个文件用于存储表和索引数据，foo.2 文件属于预分配的空文件。foo.0 和 foo.1 这两个数据文件被分为了相应的盘区对应不同的名字空间。



上图显示了命名空间和盘区的关系。每个命名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，每一个命名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命名空间浪费的空间与保持某一个命名空间中数据的连续性。上图中还有一个需要注意的命名空间：$freelist，这个命名空间用于记录不再使用的盘区（被删除的 Collection 或索引）。每当命名空间需要分配新的盘区的时候，都会先查看$freelist 是否有大小合适的盘区可以使用，这样就回收空闲的磁盘空间。

4 快速入门

MongoDB Shell 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。

4.1 启动数据库

MongoDB 安装、配置完后，必须先启动它，然后才能使用它。怎么启动它呢？下面分别展示了 3 种方式来启动实例。

4.1.1 命令行方式启动

MongoDB 默认存储数据目录为/data/db/ (或者 c:\data\db), 默认端口 27017，默认 HTTP 端口 28017。当然也可以修改成不同目录,只需要指定 dbpath 参数: /Apps/mongo/bin/mongod

--dbpath=/data/db

[root@localhost ~]# /Apps/mongo/bin/mongod --dbpath=/data/db

Sun Apr 8 22:41:06 [initandlisten] MongoDB starting : pid=13701 port=27017 dbpath=/data/db 32-bit

……

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

4.1.2 配置文件方式启动

如果是一个专业的 DBA，那么实例启动时会加很多的参数以便使系统运行的非常稳定，这样就可能会在启动时在 mongod 后面加一长串的参数，看起来非常混乱而且不好管理和维护，那么有什么办法让这些参数有条理呢？MongoDB 也支持同 mysql 一样的读取启动配置文件的方式来启动数据库，配置文件的内容如下:

[root@localhost bin]# cat /etc/mongodb.cnf

dbpath=/data/db/

启动时加上”-f”参数，并指向配置文件即可

[root@localhost bin]# ./mongod -f /etc/mongodb.cnf

Mon May 28 18:27:18 [initandlisten] MongoDB starting : pid=18481 port=27017 dbpath=/data/db/ 32-bit

……

Mon May 28 18:27:18 [initandlisten] waiting for connections on port 27017

Mon May 28 18:27:18 [websvr] web admin interface listening on port 28017

4.1.3 Daemon 方式启动

大家可以注意到上面的两种方式都慢在前台启动 MongoDB 进程，但当启动 MongoDB 进程的 session 窗口不小心关闭时，MongoDB 进程也将随之停止，这无疑是非常不安全的，幸好 MongoDB 提供了一种后台 Daemon 方式启动的选择，只需加上一个“--fork”参数即可，这就使我们可以更方便的操作数据库的启动，但如果用到了“--fork”参数就必须也启用“--logpath”参数，这是强制的。

[root@localhost ~]# /Apps/mongo/bin/mongod --dbpath=/data/db --fork

--fork has to be used with --logpath

[root@localhost~]#/Apps/mongo/bin/mongod--dbpath=/data/db--logpath=/data/log/r3.log--fork

all output going to: /data/log/r3.log

forked process: 19528

[root@localhost ~]#

4.1.4 mongodb 参数说明

最简单的，通过执行 mongod 即可以启动 MongoDB 数据库服务，mongod 支持很多的参数，但都有默认值，其中最重要的是需要指定数据文件路径，或者确保默认的/data/db 存在并且有访问权限，否则启动后会自动关闭服务。Ok，那也就是说，只要确保 dbpath 就可以启动 MongoDB 服务了。

Mongod 的主要参数有：

_ dbpath:数据文件存放路径，每个数据库会在其中创建一个子目录，用于防止同一个实例多次运行的 mongod.lock 也保存在此目录中。

_ logpath：错误日志文件。

_ logappend：错误日志采用追加模式（默认是覆写模式）。

_ bind_ip：对外服务的绑定 ip，一般设置为空，及绑定在本机所有可用 ip 上，如有需要可以单独指定。

_ port：对外服务端口。Web 管理端口在这个 port 的基础上+1000。

_ fork：以后台 Daemon 形式运行服务。

_ journal：开启日志功能，通过保存操作日志来降低单机故障的恢复时间，在 1.8 版本后正式加入，取代在 1.7.5 版本中的 dur 参数。

_ syncdelay：系统同步刷新磁盘的时间，单位为秒，默认是 60 秒。

_ directoryperdb：每个 db 存放在单独的目录中，建议设置该参数。与 MySQL 的独立表空间类似。

_ maxConns：最大连接数。

_ repairpath：执行 repair 时的临时目录。在如果没有开启 journal，异常 down 机后重启，必须执行 repair 操作。

在源代码中，mongod 的参数分为一般参数，windows 参数，replication 参数，replica set 参数，以及隐含参数。上面列举的都是一般参数。如果要配置 replication，replica set 等，还需要设置对应的参数，这里先不展开，后续会有专门的章节来讲述。执行 mongod --help 可以看到对大多数参数的解释，但有一些隐含参数，则只能通过看代码来获得，隐含参数一般要么是还在开发中，要么是准备废弃，因此在生产环境中不建议使用。

可能已经注意到，mongod 的参数中，没有设置内存大小相关的参数，是的，MongoDB 使用 os mmap 机制来缓存数据文件数据，自身目前不提供缓存机制。这样好处是代码简单，mmap 在数据量不超过内存时效率很高。但是数据量超过系统可用内存后，则写入的性能可能不太稳定，容易出现大起大落，不过在最新的 1.8 版本中，这个情况相对以前的版本已经有了一定程度的改善。

这么多参数，全面写在命令行中则容易杂乱而不好管理。因此，mongod 支持将参数写入到一个配置文本文件中，然后通过 config 参数来引用此配置文件：./mongod --config /etc/mongo.cnf。

4.2 查询记录

4.2.1 普通查询

在没有深入查询之前，我们先看看怎么从一个查询中返回一个游标对象. 可以简单的通过 find() 来查询，他返回一个任意结构的集合。

```
> var cursor = db.things.find();
> while (cursor.hasNext()) printjson(cursor.next());
{ "_id" : ObjectId("4c2209f9f3924d31102bd84a"), "name" : "mongo" }
{ "_id" : ObjectId("4c2209fef3924d31102bd84b"), "x" : 3 }
{ "_id" : ObjectId("4c220a42f3924d31102bd856"), "x" : 4, "j" : 1 }
{ "_id" : ObjectId("4c220a42f3924d31102bd857"), "x" : 4, "j" : 2 }
{ "_id" : ObjectId("4c220a42f3924d31102bd858"), "x" : 4, "j" : 3 }
{ "_id" : ObjectId("4c220a42f3924d31102bd859"), "x" : 4, "j" : 4 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85a"), "x" : 4, "j" : 5 }
```

上面的例子显示了游标风格的迭代输出. hasNext() 函数告诉我们是否还有数据，如果有则可以调用 next() 函数。

当我们使用的是 JavaScript shell，可以用到 JS 的特性, forEach 就可以输出游标了. 下面的例子就是使用 forEach() 来循环输出: forEach() 必须定义一个函数供每个游标元素调用。

```
> db.things.find().forEach(printjson);
{ "_id" : ObjectId("4c2209f9f3924d31102bd84a"), "name" : "mongo" }
{ "_id" : ObjectId("4c2209fef3924d31102bd84b"), "x" : 3 }
{ "_id" : ObjectId("4c220a42f3924d31102bd856"), "x" : 4, "j" : 1 }
{ "_id" : ObjectId("4c220a42f3924d31102bd857"), "x" : 4, "j" : 2 }
{ "_id" : ObjectId("4c220a42f3924d31102bd858"), "x" : 4, "j" : 3 }
{ "_id" : ObjectId("4c220a42f3924d31102bd859"), "x" : 4, "j" : 4 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85a"), "x" : 4, "j" : 5 }
```

在 MongoDB shell 里，我们也可以把游标当作数组来用:

```
> var cursor = db.things.find();
> printjson(cursor[4]);
{ "_id" : ObjectId("4c220a42f3924d31102bd858"), "x" : 4, "j" : 3 }
```

使用游标时候请注意占用内存的问题，特别是很大的游标对象，有可能会内存溢出. 所以应该用迭代的方式来输出. 下面的示例则是把游标转换成真实的数组类型:

```
> var arr = db.things.find().toArray();
> arr[5];
{ "_id" : ObjectId("4c220a42f3924d31102bd859"), "x" : 4, "j" : 4 }
```

请注意这些特性只是在 MongoDB shell 里使用，而不是所有的其他应用程序驱动都支持。MongoDB 游标对象不是没有快照，如果有其他用户在集合里第一次或者最后一次调 next()，可能得不到游标里的数据。所以要明确的锁定要查询的游标。

4.2.2 条件查询

到这里我们已经知道怎么从游标里实现一个查询并返回数据对象，下面就来看看怎么根据指定的条件来查询.

下面的示例就是说明如何执行一个类似 SQL 的查询，并演示了怎么在 MongoDB 里实现. 这是在 MongoDB shell 里查询，当然也可以用其他的应用程

序驱动或者语言来实现:

```
SELECT * FROM things WHERE name="mongo"
> db.things.find({name:"mongo"}).forEach(printjson);
{ "_id" : ObjectId("4c2209f9f3924d31102bd84a"), "name" : "mongo" }
SELECT * FROM things WHERE x=4
> db.things.find({x:4}).forEach(printjson);
{ "_id" : ObjectId("4c220a42f3924d31102bd856"), "x" : 4, "j" : 1 }
{ "_id" : ObjectId("4c220a42f3924d31102bd857"), "x" : 4, "j" : 2 }
{ "_id" : ObjectId("4c220a42f3924d31102bd858"), "x" : 4, "j" : 3 }
{ "_id" : ObjectId("4c220a42f3924d31102bd859"), "x" : 4, "j" : 4 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85a"), "x" : 4, "j" : 5 }
```

查询条件是 { a:A, b:B, … } 类似 “where a==A and b==B and …”。

上面显示的是所有的元素，当然我们也可以返回特定的元素，类似于返回表里某字段的值，只需要在 find({x:4}) 里指定元素的名字。

```
SELECT j FROM things WHERE x=4
> db.things.find({x:4}, {j:true}).forEach(printjson);
{ "_id" : ObjectId("4c220a42f3924d31102bd856"), "j" : 1 }
{ "_id" : ObjectId("4c220a42f3924d31102bd857"), "j" : 2 }
{ "_id" : ObjectId("4c220a42f3924d31102bd858"), "j" : 3 }
{ "_id" : ObjectId("4c220a42f3924d31102bd859"), "j" : 4 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85a"), "j" : 5 }
```

4.3 常用工具集

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

_ bsondump: 将 bson 格式的文件转储为 json 格式的数据。

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

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

_ mongodump/ mongorestore: 数据库备份和恢复工具。

_ mongoexport/ mongoimport: 数据导出和导入工具。

_ mongofiles: GridFS 管理工具，可实现二制文件的存取。

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

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

5 总结

Mongo 最大的特点是支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。MongoDB 服务端可运行在 Linux、Windows 或 IOS 平台，支持 32 位和 64 位应用，默认端口为 27017。推荐运行在 64 位平台，因为 MongoDB 在 32 位模式运行时支持的最大文件尺寸为 2GB。