    从我第一次听到Nosql这个概念到如今已经走过4个年头了，但仍然没有具体的去做过相应的实践。最近获得一段学习休息时间，购买了Nosql技术实践一书，正在慢慢的学习。在主流观点中，Nosql大体分为4类，**键值存储数据库，列存储数据库，文档型数据库，图形数据库**。今天主要快速的浏览了文档型数据库中目前市场占有率的最高的MongoDB数据库。记得初次见到和关注这个数据库还是我刚来上海的时候，公司将该数据库作为新建的项目管理系统的后台数据库，当时还是很向往的，只是无缘参与那个项目，也就一直没有和该数据库打上交道。接下来简单的介绍下该数据库的基本原理和相关应用，也算是巩固知识和加强记忆了。大体上快速学习分为两部分，第一部分为基础，第二部分为进阶。

* 优势与不足

首先，MongoDB不需要表结构，它是模式自由的(schema-free)，例如{"welcome", "Shanghai"}, {"name", "bibi"}可以放到同一个集合中。那么它是如何在存储数据的呢？MongoDB在保存数据时会使用Bson的形式，一种json的二进制化形式，并把它与特定的Key进行关联。这样将非常便于程的扩展和维护，在需要增加新字段或者修改字段时只需要修改程序，而不需要修改数据库的架构，非常的方便。

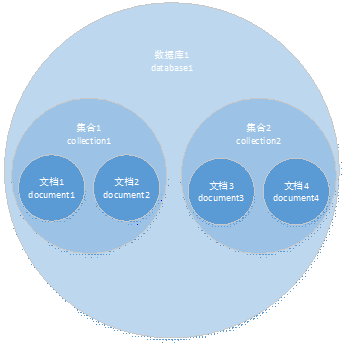
其次，MongoDB原生的提供很强的伸缩性，对于web应用，当需要存储的数据不断增加时，我们将面对一个很大的问题，如何给数据存储模块扩容。在原有的数据存储模块架构中，往往需要通过购买功能更强大的机器，给数据库服务器升级，但这存在的问题是成本很高，同时升级也受限于当时硬件技术水平的。于此同时，由于实际web应用中，访问量并不是相似的，例如在各种活动期间，会出现各种特殊峰值，例如淘宝的11节的第1分钟的访问量都已达到千万级，而在平时这个值相对小很多。所有这时增加服务器在忙时可能仍然达不到目的，而闲时又会造成大量的浪费，所以伸缩性成为上成为web架构中的最重要的技术指标之一，这也是当前Nosql技术流行的主要原因。

最后，MongoDB还提供丰富的功能，包括支持辅助索引，支持MapReduce和其他聚合工具，并提供了分布式环境下的高可用，比如自动的在集群中增加和配置节点。

  当然，MongoDB也不是万能的，实际上也存在一些不足。例如，不支持join查询和事务处理，数据也不是实时写入到磁盘的，同时存储数据时需要预留很大的空间。在实际项目中，需要根据实际的需要进行选择，当前很多主流网站均使用Sql+NoSql的形式构建数据库存储模块。

* 基本结构

MongoDB中的文档document相当于Sql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；多个集合组合在一起，就是数据库database；一个数据库服务器可以有多个数据库实例。



* 相关文档和程序

官方下载地址：<https://www.mongodb.org/>, 官方目前的版本是3.2，其实2.4以后版本都可以很.NET平台很好和整合，如果官网下载失败（常见），就直接网上搜索一个指定版本就好。

官方文档地址：<https://docs.mongodb.org/getting-started/shell/>

Mongod：数据库程序

Mongos：分片控制器

Mongo：Windows下客户端Client

Mongodump：数据库的dump工具，支持备份，快照等方式

Mongorestore：从一个dump文件恢复数据库

Mongoexport：导出单个数据集合到json、CSV等格式

Mongoimport：导出json、CSV等格式数据

Mongofiles：用于到GridFS中，设置和获取数据文件

Mongostat：显示性能统计信息

* 安装步骤（还可以参考博主懒惰的肥兔的博文<http://www.cnblogs.com/lzrabbit/p/3682510.html>，非常详细，点个赞）

1. 首先在当前目录中，建立相关目录:Data保存数据文件，log保存日志信息，etc保存配置文件(mongodb.conf)。
2. 在cmd中使用命令，命令如下所示：

|  |
| --- |
| 创建服务：sc create mongodb binPath=”d:\mongodb\bin\mongod.exe –service –config=d:\mongodb\etc\mongodb.conf”  快捷方式：mongod -f d:\mongodb\etc\mongodb.conf --install  开启服务：net start mongodb  关闭服务：net stop mongodb  删除服务：sc delete mongodb |

配置文件内容如下：

|  |
| --- |
| dbpath=D:\mongodb\data #数据库路径  logpath=D:\mongodb\log\mongodb.log #日志输出文件路径  logappend=true #错误日志采用追加模式，配置这个选项后mongodb的日志会追加到现有的日志文件，而不是从新创建一个新文件  journal=true #启用日志文件，默认启用  quiet=true #这个选项可以过滤掉一些无用的日志信息，若需要调试使用请设置为false  port=27017 #端口号 默认为27017 |

.NET下Mongodb的客户端API可以nuget中很容易的找到，mongoDB .NET 2.0 Driver是使用率最高的，其支持.NET await的异步模型、动态类型dynamic、扩展方法形式的常见Linq查询（表达式树）、简化的日志管理和静态性能的记录，使用起来非常的便捷。

在该组件中，client默认就是连接池的方式，所以直接使用单例的client即可，在插入数据时使用BsonDocument，其和json的结构完全一样，此外在构建Client的连接字符时主要加上mongodb://的协议名就OK。

详细的文档地址为：http://mongodb.github.io/mongo-csharp-driver/2.0/getting\_started/quick\_tour/

* 性能优化

Mongodb和一般关系型数据库一样，也支持查看执行计划explain，来了解系统实际对索引的使用情况，并根据该情况优化索引，提升查询性能。在执行计划结果中，包含如下属性。

Cursor:返回游标类型(BasicCursor, **BTreeCursor**)

Nscanned:被扫描的文件数量

N：返回的文件数量

Millis:耗时(毫秒)

indexBound，表示索引的使用情况，

优化器Mongodb database profiler

和关系型数据库类似，mongodb也提供慢查询（就是耗时较长的命令）日志的分析，Mysql有show Query Log与之对应。Profile有3个级别，分别是：0，不开启；1，记录慢命令（默认为>100ms）；2，记录所有命令。可以通过以下命令获取和设置profile级别和慢命令的执行时间阀值，db.getProfilingLevel()，db.setProfilingLevel(1, 100)。

MongoDb的profile是记录在数据库的系统db中的，位置在system.profile，因此可以通过如下命令获取所有执行时间大于10ms的profile记录，db.system.profile.find({millis:{$gt:5}})。结果字段中，ts表示命令的执行时间，info为命令详细信息（类似SQL语句了），reslen表示返回结果集大小，nscanned表示查询扫描的记录数，nreturned表示实际返回的结果集，millis为执行耗时。此外，profile还提供一个show profile命令用于获取最近5条执行记录。

当发现扫描的数据集数远大于返回的记录集数时，就需要考虑建立索引来加速查询了，接下来介绍几条常见的优化策略：

1. 在查询条件和排序字段上建立索引
2. 限定返回的结果集skip(),limit(),在这点上mongo真心很赞，因为在互联网场景下的查询都是数据库分页的
3. 只查询使用到字段，减少内存消耗，在find（）中第一个参数为查询条件，第二参数为所选字段，与SQL中尽量不要使用select \* 类似。例子为db.students.find({}， {name:1}).sort(age: -1).skip(2).limit(3)
4. 采用Capped Collection，类似固定大小的数组，效率高，使用方式为：db.createCollection("mycoll", {capped:true, size:100000})。需要注意的是该集合只支持insert和update操作，不支持一般的delete，只支持类似于SQL中truncate的drop操作。其数据顺序以插入顺序为准，如果超过大小，则按照循环数组的形式覆盖最先的记录（FIFO）。
5. 使用类似存储过程的Server Side Code Execution来减少网络传输开销
6. 在mongodb query optimizer不能良好工作时（极少），可以通过hint强制索引，在SQLServer， Oracle中也有相似概念，就是不知道有木有包含索引
7. 采用profiling

* 性能监控

与性能监控相关的常见命令包括：

db.serverStatus(): 查看数据库实例的运行状态，信息包括：服务器版本、启动时间、globalLock中的当前请求（读/写）队列信息、activeClients当前的连接信息、mem内存占用信息、indexCounters索引被访问命中的相关信息、服务器的数据量、添删改查等操作的信息

db.stats(): 查看当前数据库的状态，例如当前的test数据库中集合&对象的数量，数据的可用&当前大小，索引的数量和大小等

Tip:

在windows中有mongostat和mongotop工具用于查看统计信息，在Linux有mongosniff,mongostat等工具，此外还有cacti、Nagios、Zabbix等第三方监控工具。

* Replica Sets复制集

MongoDB支持在多个机器中通过异步复制达到故障转移和实现冗余，多机器中同一时刻只有一台用于写操作，其支持的高可用分为旧的Master-Slave主从复制方式和Replica Sets复制集方式，推荐使用后者。可以通过rs.status()命令查看复制集状态，members节点描述复制集相关信息，还可以使用rs.isMaster()查看相关信息。需要注意的是，在多服务器的集群中，通过一个keyFile来行进识别。

Replica Sets时通过日志oplog来存储写操作的，oplogs.rs是一个固定长度的Capped Collection，存在于local数据库中。命令db.printReplicationInfo()可以查看oplog的元数据信息，db.printSlaveReplicationInfo()可以查看slave的同步信息。此外，ReplicaSets的配置信息放在system.replset中，可以很方便的看到主从的配置信息。

实现数据的读写分离非常简单，只需要在从库中设置db.getMongo().setSlaveOk()即可。ReplicaSets的故障转移是自动的，比如我们kill primary的pid， 然后再次查看rs.status()可以看到主服务器的的转移。在windows中可以使用tasklist查看进程信息，tskill关闭指定pid的进程，netstat –aon | findstr “27020”可以找到占用指定端口的pid。

在提供高可用方案的同时，它也提供负载均衡的解决方案，增减Replica Sets节点非常常见，可以通过rs.add(“replset:27023”)增加节点，节点增加后自动与主服务器同步数据，可以通过rs.remove(“replset:27024”)减去该节点，感觉棒棒哒。



部署Replica Sets

1. 在单机多实例的实验场景下，由于次要的仲裁服务器arbiter不支持使用localhost(会提示重复)，因此在C:\Windows\System32\drivers\etc\hosts中添加一行：127.0.0.1 replset
2. 添加primary节点和两个Secondary节点（其中一个为仲裁节点），其实就是把之前的配置复制一遍，在各自的配置文件中加入replSet=rs1，并设置不同的port
3. 分别启动三个节点mongo -f XXX
4. 连接primary节点（--port 27020），并通过命令行配置，命令如下所示，当然也可以通过配置文件来设置：

|  |
| --- |
| 设置配置：config\_rs1 = {\_id : "rs1",members : [ { \_id:0, host:"replset:27020", priority:1 },{ \_id:1, host:"replset:27021", priority:1 },{ \_id:2, host:"replset:27022", priority:1, "arbiterOnly": true } ]}  启用配置：rs.initiate(config\_rs1)  这儿需要注意，这个操作可能需要很长时间，请耐心等待 |

Tip：默认情况primary支持读写，而secondary不支持，可以通过rs.slaveOk()命令使得次要节点也能读写。

此外，大家也可以查看：<http://www.cnblogs.com/jRoger/articles/4708490.html>，博主的内容很详尽。

* Sharding分片

这是一种将海量数据水平扩展的数据库集群系统，数据分别存储在Sharding的各个节点上，这就是mongodb源生支持互联网场景的特征，这部分管理不再是第三方的一个解决方案而是数据库自带的，因而更加便捷高效，这也是我们常说的分库分表。MongoDb的数据分块被称为chunk，每个chunk都是collection中的一段连续的数据记录，通常大小为200MB，超出则生成新的数据块。



图 1 Sharding Cluster简易架构

构建一个Sharding Cluster需要三种角色：

1. Shard Server即存储实际数据的分片，每一个shard可以是一个mongod实例，也可以是replicaSet，推荐后者
2. Config Server，为了将一个特定的Collection存储在多个Shade中，需要为该Collection指定一个shard key，例如{age:1}，shard key决定该条记录所属的chunk。Config Servers就是用来存储所有Shard节点的配置信息、每个chunk的shard key范围、chunk在各shard的分布情况、该集群中所有DB和Collection的Sharding配置信息。
3. Route Process是一个前端路由，客户端由此接入，然后询问Config Server需要到哪个Shard上查询或保存记录，在连接到相应的Shard进行操作。客户端只需要将原本发送给mongod的信息发送到Routing Process，而不用关系操作记录存储在哪个Shard。也就是说这个步骤对用户透明，路由算法由系统提供，比如我们常见的一致性hash算法。

搭建步骤：（也可以参照博友苏若年的博文<http://www.cnblogs.com/dennisit/archive/2013/02/18/2916159.html>，非常详细）

1. 首先构建之前之前介绍过的三个角色，route process 1个（port, 27026），config Server 1个(port, 27027)，Shard Server 2个(port, 27028, 27029)，建立相关目录和设置相关配置文件。配置文件的差异有：Config配置文件：configsvr=true；Router配置文件：configdb=localhost:27027（配置服务器地址）， chunkSize =100（chunk块的大小），其他配置基本一致。
2. 连接到Router的admin数据库, mongo admin --port 27026, 然后运行命令添加两个shard节点，db.runCommand({addshard:”localhost:27028”})，db.runCommand({addshard:”localhost:27029”})，完成Sharding集群的配置。
3. 选择指定数据库将其状态设置为可以分片db.runCommand({ enablesharding:"test" })
4. 指定分片具体集合，db.runCommand({ shardcollection:"test.users", key:{ \_id:1 }})，至此环境搭建完成。
5. 可以在该表中插入100000条测试数据，然后通过db.users.stats()查询该数据集情形，在shards中可以看到具体各个片区的数据量。

此外，该系统支持添加节点和删除节点，删除节点的命令为db.runCommand({ "removeshard":"localhost:27030" )，printShardingStatus()查看分片的生效情况，还可以通过db.runCommand({ isdbgrid:1 })命令查看当前实例是否在Sharding环境中。

* Replica Sets与Sharding的结合

通过ReplicaSet和Sharding结合，可以提供可扩展的高可用方案。当业务规模增大时，我们常见的扩展方式有两种，一种是垂直伸缩，一种是分片（水平伸缩），前者通过增加服务器的CPU和内存来实现，成本很高，而后者将数据分布到不同的服务器，不同服务器上的数据分块共同组成一个逻辑数据库。

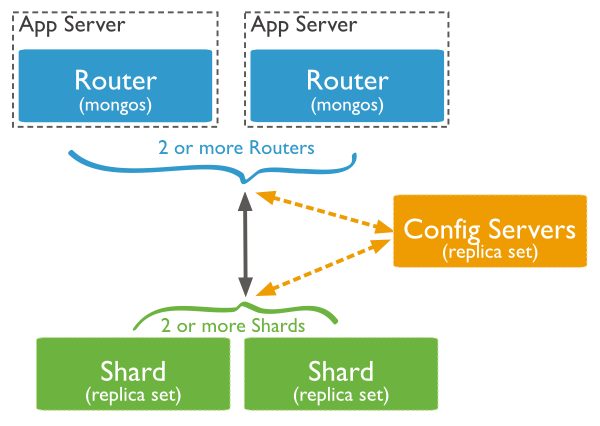


图 2 完整的mongodb高可用可扩展架构

Shards：存储数据，通过replica sets提供高可用和数据持久性。

Query Routers：当数据库服务器mongod很多时，推荐增加Router来分发大量的客户请求。Mongos是一个轻量级的进程不需要数据目录，

Config servers：存储集群元数据，包含集群数据集与各个片区的映射，在3.2版后支持将config-servers部署为replica set，避免单点故障，不再推荐原有的三镜像形式的配置服务器实例。

MongoDb通过shard key对数据进行分区，系统默认使用range based partition或hash based partition。前者通过区间分布，因而相近数据分布较近，范围查询的效率更高，于此同时由于分布不均匀，当请求集中在其中一台服务器时，将出现过量负载；后者通过hash函数分布，分布比较分散，负载均匀，但对于范围查找相对较慢。

系统提供后台运行的splitting功能，当Shard不断增大超过阀值，系统将会把它分成等量的两部分。后台balancing进程管理chunk的迁移，当负载均衡器发现某个shard中chunk过多时，会将部分chunk转移到chunk数最少的服务器，值得一提的是，只有在源shard的chunk迁移到目的shard后，才会删除源上的chunk，因此在迁移过程中出现问题并不会导致数据丢失。

在Windows上详细构建步骤可参照博友左盐的博文<http://www.cnblogs.com/spnt/archive/2012/07/26/2610070.html>，以及博友Geek\_Ma的博文<http://www.cnblogs.com/geekma/archive/2013/05/16/3081532.html>。

* 基础查询

有几点需要注意：不需要预先创建集合，在第一次插入数据时会自动创建；文档中可以存储任意类型数据，不需要类似alter table的语句来改变结构；每次插入时都有一个\_id，类型为OBjectId，其实就是GUID了，便于分布式环境下的唯一标示，当然它也可以是int或long等类型。

|  |  |  |
| --- | --- | --- |
| 操作类别 | 实例 | 备注 |
| 插入 | j={name, "bibi"};t={x : 3};  db.things.save(j); db.things.save(t);  db.things.find(); | 1. 不需要预先创建集合 2. 文档中可以存储任何结构的数据 3. 每次插入时默认添加一个\_id字段 4. 该操作等价于insert()操作 |
| 选择数据库 | Use test | 1. 默认使用test |
| 修改 | Db.things.update({name,"mongo"}, {$set:{name:"mongo\_new"}}); |  |
| 删除 | Db.things.remove({name:"mongo\_new"}); |  |
| 普通查询 | var cursor = db.things.find();  while(cursor.hasNext()) printjson(cursor.next()); | 获得游标，遍历游标。注意在数据集合很大时可能会引起内存溢出 |
|  | Db.things.find().forEach(printjson) |  |
|  | Var arr = db.things.find().toArray();  Arr[5]; |  |
| 条件查询 | Db.things.find({x:4}, {j:true}).forEach(printjson); |  |
| FindOne | Printjson(db.things.findOne({name:"mongo"})); |  |
| limit | Db.things.find().limit(3); |  |

* 高级查询

|  |  |  |
| --- | --- | --- |
| 操作符 | 实例 | 备注 |
| 条件操作符 | Db.collection.find({"field":{$gt:value}});  Db.collection.find({"field":{$lt:value}});  Db collection.find({"field":{$gte:value}});  Db.collection.find({"field":{$lte:value}}); | Field>value  Field<value  Field>=value  Field<=value |
| $all | Db.users.find({age:{$all:[6, 8]}}); | 必须满足[]内所有值 |
| $exists | Db.things.find({age:{$exists:true}});  Db.things.find({age:{$exists:false}}); | 查询存在age字段的记录  查询不存在age字段的记录 |
| Null值的处理 | ~~Db.collection.find(age:null)}~~  Db.collection.find(age:{$in:[null], $exists:true})} | 这儿要注意，在只用null作为判断条件是，还会把不存在age字段的记录找出来 |
| $mod | Db.collection.find({age:{$mod:[10, 1]}}) | 取模运算 |
| $ne | Db.things.find({x:{$ne:3}}); | 不等于 |
| $in | Db.users.find({age:{$in:[2,4,6]}}); | 包含 |
| $nin | Db.users.find({age:{$nin:[1,3]}}) | 不包含 |
| $size | {name:'bibi', age:26, luck\_number:[3,7,9]},  db.users.find({luck\_number:{$size: 3}}) | 数组元素个数 |
| 正则表达式匹配 | Db.users.find({name:{$not:/^B.\*/}}); | 查询不匹配name=B\*带头的记录 |
| Javascript查询和$where查询 | Db.collection.find({a:{$gt:3}});  Db.collection.find($where:"this.a>3");  Db.collection.find(this.a>3");  f=function(){return this.a>3}  db.collection.find(f); | 查询a大于3的数据 |
| count | Db.users.find().count();  ~~Db.user.find().skip(10).limit(5).count();~~  Db.user.find().skip(10).limit(5).count(true); | 查询记录条数  还是返回的所有记录数  加true也能限制数量 |
| Skip | Db.users.find().skip(3).limit(5) | **相当于limit(3, 5)** |
| sort | Db.colletion.find().sort({age:1});  Db.colletion.find().sort({age:-1}); | 按升序进行排序  按降序进行排序 |
| 游标 | For(var c = db.t3.find();c.hasNext();){  Printjson(c.next());}  Db.t3.find().forEach(function(u){printjson(u);}); |  |

* 统计Map/Reduce

Map/Reduce这个概念已经存在了很多年，记得有个印度工程时通过做不同口味的番茄酱的理解风趣幽默的为妻子解释了这个概念，主体的意思就是分工然后汇总。在这里Map/Reduce相当于MySQL中的"group by"，使用过程需要实现Map函数和Reduce函数。

|  |  |  |
| --- | --- | --- |
| 函数名 | 实例 | 备注 |
| 前提条件 | Db.students.insert({classid:1, age:14, name:'Tom'})  Db.students.insert({classid:2, age:27, name:'Bibi'}) | 插入班级1,2共8条记录. |
| Map | m=function(){emit(this.classid, 1)} | Map函数必须调用emit(key, value)返回键值对，使用this访问当前待处理的Document.相当于SQL的分组操作，其中的this.classid分组属性，1是用于聚合的属性或值 |
| Reduce | r=function(key, value){  var x =0;  values.forEach(function(v){x+=v});  return x;} | Reduce函数接受的参数类似Group效果，将Map返回的键值序列组合成{key, [value1, value2, value3..]}传递给reduce. 相当于SQL的聚合操作，这儿的x+=v实际就是SQL中的count(\*) |
| Result | Res=db.runcommand({  mapreduce:"students",  map:m,  reduce:r,  out:"student\_res"  }); | 相当于分组聚合操作的执行，并将结果集输出到指定的Collection  获得结果：  {"\_id": 1, "value":3} |
| finalize | F=function(key, value){return {classid:key, count:value};}  Res=db.runcommand({  …(同上)  finalize:f  }); | 利用finalize()我们可以对reduce()的结果做进一步的处理。结果变为如下形式：  {"classid": 1, "count":3}。类似于SQL中的取别名格式化输出等操作。 |
| options | Res=db.runcommand({  …(同上)  Query:{age:{$lt:10}}  }); | 可选项，例如过滤操作，只取age<10的数据。相当于where操作，注意不是having。 |

* 索引

MongoDB提供了多样性的索引支持，索引信息被保存在system.indexes中，且默认总是为\_id创建的索引。

|  |  |  |
| --- | --- | --- |
| 操作符 | 实例 | 备注 |
| 基础索引 | Db.t3.ensureIndex({age:1});  Db.t3.getIndexes(); | 按升序排序的索引。注意，1表示升序，-1表示降序  查看有哪些索引，默认情况下，\_id为创建表时自动创建的索引 |
|  | Db.t3.ensureIndex({age:1}, {background:true}); | 当系统已有大量数据时，创建索引非常耗时，我们可以在后台执行 |
| 文档索引 | Db.factories.insert({name:"SORY", addr:{city:"Shanghai", state:"China"}});  Db.factories.ensureIndex({addr:1}); | 索引可以是任何类型的字段，甚至文档。  **注意索引建立的顺序，**这点和关系型数据库一样，错误的select顺序可能造成不触发索引 |
| 组合索引 | Db.factories.ensureIndex({"addr.city":1, "addr.state":1}); |  |
| 唯一索引 | Db.users.ensureIndex({firstname:1, lastname:1}, {unique:true}); | 注意，如果建立索引所选字段的既有值有重复的，是无法建立唯一索引的。 |
| 强制使用索引 | Db.t5.find({age:{&lt: 30}}).hint(name:1, age:1).explain(); | 通过执行计划查看，SQL Server中也有相似的概念，强制走索引 |
| 删除索引 | Db.t1.dropIndexes  Db.t1.dropIndex({firstname:1}) | 删除t3表的所有索引  删除指定索引 |

Tip:

博文主要供个人基础学习使用，若有疏漏，忘见谅。文中部分图片来之于mongodb官网https://docs.mongodb.org/manual/。

参考资料：

1. 皮雄军. NoSQL数据库技术实战[M]. 北京:清华大学出版社, 2015.