第一天：认识MONGODB

——NewSQL中文社区系列课程

NewSQL中文社区出品
社区：http://newsql.cn/
ＱＱ：705287(ymind)
Ｑ群：79377097(NoSQL三群)

陈彦铭出品
2012.5

声明

声明一哈

- 本PPT根据《MongoDB: The Definitive Guide》原版制作而成；我很穷，但我大胆的承认我看的是盗版PDF；
- 如果您发现翻译或描述有误，请不吝赐教；
- 制作PPT时，我不一定会100%的引用书中提供的示例，看书的朋友不要见怪；
- 如果有任何疑问请通过微博或QQ反馈给我；

网鸟QQ群

- 刺客巅峰（47700865）——IT技术综合交流群组，这里神马都有。欢迎各种程序猿加入！
- NoSQL群（23152359(满)、193713524、79377097、191845335）——新建的NoSQL专用群组，来吧！来吧！欢迎各位光临！
- NewSQL中文社区：http://newsql.cn/ （注册前请到群里获取邀请码）

从NOSQL说起

NoSQL(NoSQL = Not Only SQL)，意即反SQL运动，是一项全新的数据库革命性运动，早期就有人提出，发展至2009年趋势越发高涨。NoSQL的拥护者们提倡运用非关系型的数据存储，相对于目前铺天盖地的关系型数据库运用，这一概念无疑是一种全新的思维的注入。

NoSQL不是产品，是一项运动！

NOSQL数据库特点

优点

- 高并发读写
- 海量数据存储
- 高可扩展性
- 高可用性

缺点

- 缺乏事务一致性
- 缺乏读写实时性
- 不支持复杂查询

NOSQL数据库类型

类型	简介
key-value	Key 指向 Value 的键值对，通常用hash table来实现 产品：Tokyo Cabinet/Tyrant, Redis, Voldemort, Oracle BDB
列式数据库	以列簇式存储，将同一列数据存在一起 产品：Cassandra, HBase, Riak
文档型数据库	Key-Value对应的键值对，Value为结构化数据 产品：CouchDB, MongoDB
图结构数据库	以“图”为基本存储模型 产品：Neo4J, InfoGrid, Infinite Graph

NoSQL相关的产品已经非常之多，具体到项目时应该多多评测，不要盲目追求概念！

MONGODB简介

功能丰富

- 索引
- 聚合
- 分布式文件系统
- 存储JavaScript
- 定容集合

可扩展性

- 数据分片
- 负载均衡
- 海量数据支撑

无模式

- 非关系型
- 面向文档的数据存储

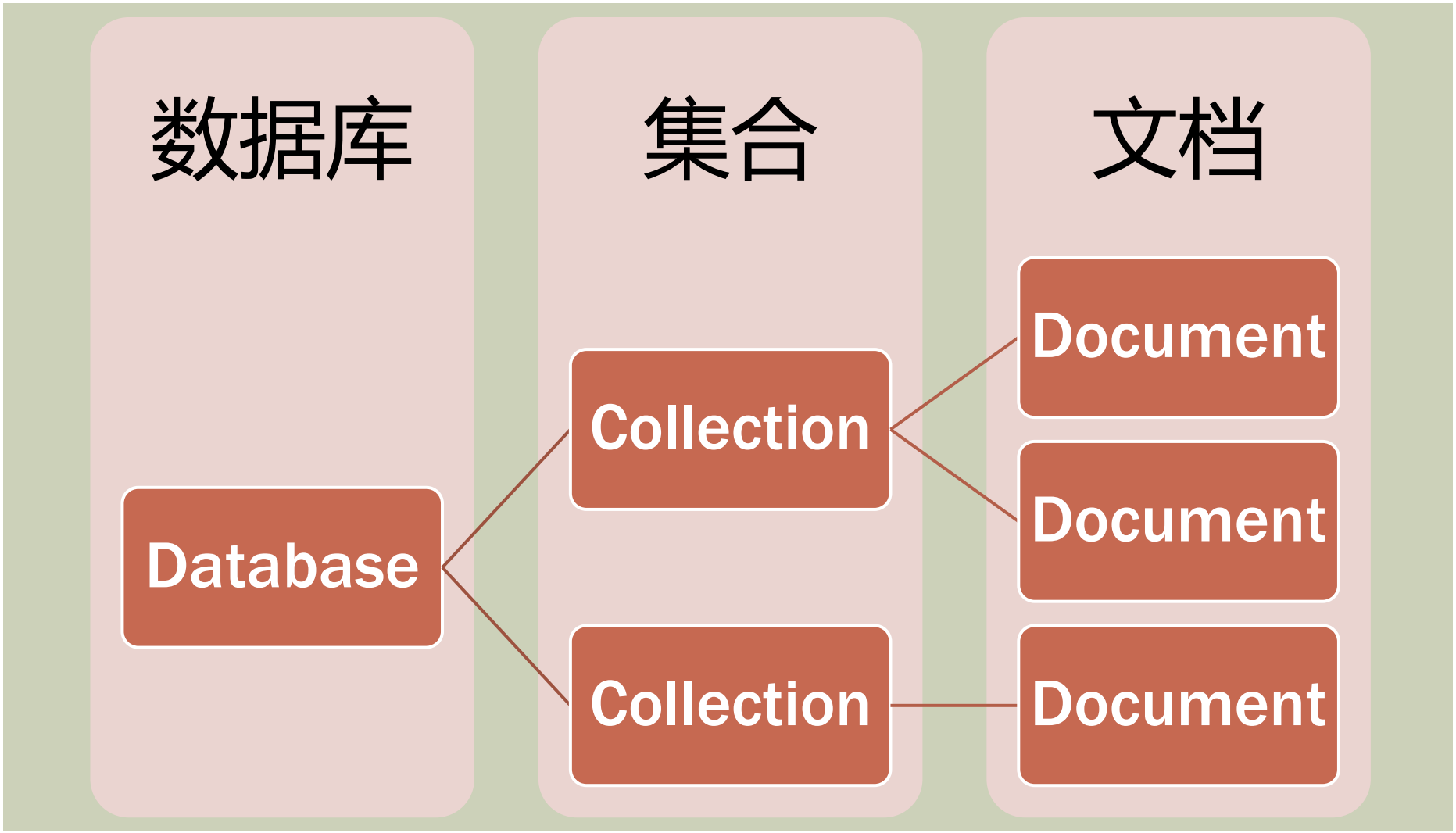
高性能

- 空间预分配
- 内存映射
- 非事务性

易管理

- Server自治
- 自动故障转移
- 动态集群

MONGODB数据架构



数据库、集合、文档命名规范

数据库

- 任意UTF-8字符串（比如：网鸟）；
- 不能是空字符串；
- 不得包含以下字符：(空格)、.、$、/、\、\0；
- 长度不得超过64字节；
- 保留字：admin、local、config等；

集合

- 任意UTF-8字符串（比如：网鸟）；
- 不能是空字符串；
- 不能包含\0 、 $；
- 不能以system.开头；

文档

- 任意UTF-8字符串（比如：网鸟）；
- 不能是空字符串；
- 不能包含\0、 $；
- 不能以_开头；

MONGODB的下载与安装

版本选择

- MongoDB的版本命名规范如：x.y.z；
- y为奇数时表示当前版本为开发版，如：1.5.1、1.7.2；
- y为偶数时表示当前版本为稳定版，如：1.6.3、1.8.2；
- 详情：http://www.mongodb.org/display/DOCS/Version+Numbers

平台选择

- 生产环境使用32位机器的，老陈无话可说，反正64位又运行不了，你没的选；
- 在64位机器上坚决要使用64位版本，可以获得更大的内存映射空间，提升MongoDB的性能，Win Server 2008 有特别优化版，注意下载页面的“*2008+”；
- 至于操作系统，MongoDB是跨平台的，在下载页面找自己对应的版本即可；

下载安装

- 下载地址：http://www.mongodb.org/downloads；
- 直接运行：解压后，在命令行执行%setupPath%\bin\mongod.exe即可；
- 作为Windows服务：请参考http://www.mongodb.org/display/DOCS/Windows+Service；
- 其他操作系统请参考：http://www.mongodb.org/display/DOCS/Quickstart；

MONGODB SHELL

- 运行Shell
  - 打开命令行；
  - 执行%setupPath%\bin\mongo.exe即可进入MongoDB Shell；
  - 在MongoDB Shell中可以运行任何JavaScript脚本；
  - 在Shell中可以键入help以获取帮助，当然，都是E文的；
- 数据操作
  - 创建：
    ```
    db.test.insert({ “Name” : “ymind”, “age” : 8 });
    ```
  - 读取：
    ```
    db.test.find({ “Name” : “ymind” } );
    ```
  - 更新：
    ```
    db.test.update(“Name” : “ymind”, { “$set” : { “age” : “5” } } );
    ```
  - 删除：
    ```
    db.test.remove({ “Name” : “ymind” } );
    ```

这里只是简单示例，更详细的教学还在后面！

MONGODB的数据类型

MongoDB底层使用Bson来维护数据，Bson是一种二进制下的类似于Json的数据格式。

以下是MongoDB中几个值得啰嗦一下的数据类型：

- MongoDB的基本数据类型
- MongoDB的数组
- MongoDB的文档嵌套
- MongoDB的ObjectId

MONGODB的基本数据类型

null	这个有神马好说的？？
bool	true、false；
数字	仅支持64位浮点数，其他类型的数字会自动转化为此格式；
字符串	支持UTF-8字符集；
日期	存储标准纪元开始的毫秒数，不含时区；
regex	符合JavaScript语法的正则表达式；
code	任何JavaScript代码；
binary	可存储任意字节数的字节数组，但在Shell中无法使用；
数组	对象集合或列表在MongoDB可以被存储为数组；
嵌套文档	一个文档可以包含另外一个文档；
其他	详细内容请参考http://bsonspec.org/#/specification；

MONGODB的数组

MongoDB中的数组既可以作为有序对象（如列表）来操作，也可以作为无序对象（如集合）来操作。MongoDB可以以原子操作的形式来修改或删除数组中的元素。

下面是一个数组示例（不用怀疑，这就是Json语法）：

```
{
        // 这个不是数组
        "Status" : 0,
        // 这个是空数组
        "Messages" : [],
        // 包含3个元素的数组
        "ResponseBody" : [
                "h010t00101025",
                "h010t00101039",
                "h010t00101040"
        ]
}
```

MONGODB的文档嵌套

文档式存储的特色就是允许文档对象的嵌套，这能够使我们的实体数据在MongoDB的存在形式更加自然。

老陈忠告：不要滥用嵌套文档，对于需要复杂运算才能完成的嵌套应当使用“外键模式”来解决，否则之后的DAL（Data Access Layer，数据访问层）开发会相当痛苦！

以下是一个嵌套文档的格式示例：

```
// 根文档
{
        "Status" : 0,
        // 嵌套文档
        "ResponseBody" : {
        "CityName" : "北京",
        "Keyword" : "钓鱼台",
        // 再套一个数组，数组的每个元素又是文档，NB吧！
        "Landmarks" : [
                        { “Name” : “北京钓鱼台国际旅行社” },
                        { "Name" : "钓鱼台“ }
                ]
        }
}
```

MONGODB的OBJECTID

ObjectId是_id（貌似前面木有提过？木有关系，_id是MongoDB在每个文档中的默认唯一标识的名称）的默认类型。

ObjectId的结构如下：

0	1	2	3	4	5	6	7	8	9	10	11
time				machine			pid		inc		

如上图所述，ObjectId可以保证同一台机器上不同进程（即MongoDB实例）生成的值都是唯一的。详情请参考：http://www.mongodb.org/display/DOCS/Object+IDs

注意事项：

- 当插入的文档并没有指定_id的时候，MongoDB会自动创建，并使用ObjectId类型；
- _id可以被指定为其他的类型，比如整数；
- 关于自增标识，MongoDB没有内置自增标识，在目前的roadmap中也没有相关的规划，自定义自增标识的方法将会在后续内容中专门讲解；

结束语

采纳

- 老陈接触MongoDB快三年了，已经被彻底征服；
- 我自己的项目（www.yudingbao.com）已经完全抛弃了关系型数据库；

缺点

- 最大的缺点就是无法支持复杂运算，但这对提升MongoDB自身的负载能力有好处；
- GEO搜索不支持传统意义上的距离运算，而是使用*“经纬度”“差值”*；
- 不支持对MapReduce的结果进行后期处理、聚合查询不够直接给力（据说2.2+会改善）；

分享

- 强烈的推荐大家使用各种NoSQL方案来逐步的替代已有的关系型数据库（我不特指MongoDB）；
- 对于新技术要多创建原型程序进行实践论证，而不是人云亦云，实际上我并不认为MongoDB就是绝对的最好；

第二天：初识增删改

——NewSQL中文社区系列课程

NewSQL中文社区出品
社区：http://newsql.cn/
ＱＱ：705287(ymind)
Ｑ群：79377097(NoSQL三群)

陈彦铭出品

教学内容

- 添加文档
- 删除文档
- 更新文档
- 其他

添加文档

```
db.foo.insert({
    "bar" : "baz"
});
```

- 插入时MongoDB会检查文档是否包含_id，如果文档没有指定_id，MongoDB会为其创建；
- 对于多个文档，我们推荐批量操作，批量操作有如下优点：
  - 更少的连接次数；
  - 更少的信息头检测；
  - 对“待插入集合”的灵活控制；
- 默认情况下，插入文档时MongoDB仅检查传入数据是否包含_id以及数据大小是否超过16MB（v1.8-为4M），除此之外不再做任何验证；
  - 如此简单的判断可以得到更高的性能；
  - 缺点是可能被录入无效数据；
- MongoDB在插入数据时不执行任何代码，因此没有注入攻击的风险；

删除文档

```
// 删除全部文档（清空集合）
db.foo.remove();

// 删除指定记录
db.foo.remove({
    "bar" : "baz"
});
```

建议：如果要清空集合，而且集合内包含了过多记录时，不妨考虑使用db.drop_collection('foo')命令直接删除集合，然后使用db.foo.ensureIndex()命令重建索引（之前的索引已经被drop掉了）。

更新文档——文档替换(1)

```
// 原文档
{
        "_id" : ObjectId("4b2b9f67a1f631733d917a7a"),
        "name" : "joe",
        "friends" : 32,
        "enemies" : 2
}

// 期望结果
{
        "_id" : ObjectId("4b2b9f67a1f631733d917a7a"),
        "username" : "joe",
        "relationships" :
        {
                "friends" : 32,
                "enemies" : 2
        }
}
```

更新文档——文档替换(2)

```
// 操作：首先查找记录
var joe = db.users.findOne({"name" : "joe"});

// 修改记录属性，很普通的JavaScript语法
joe.relationships = {"friends" : joe.friends, "enemies" :
joe.enemies};
joe.username = joe.name;
delete joe.friends;
delete joe.enemies;
delete joe.name;

// 更新
db.users.update({ "name" : "joe" }, joe);
```

老陈：update操作会替换整个匹配到的文档，使用时请千万留意！
update有2个可选的bool参数，第1个表示是否开启upsert模式（文档存在时更新，不存在时自动创建）；第2个表示是否开启multi模式（默认情况下update操作只会更新第一个匹配到的文档，如果开启multi模式，则会更新所有匹配到的文档）。

更新文档——UPSERT模式

```
// 更新：指定第三个参数为true可以开启upsert模式
db.users.update({ "name" : "joe" }, joe, true);
```

在upsert模式下，如果找到匹配的记录则更新之，否则如果找不到匹配记录就会创建一条新的记录。

老陈：对于upsert模式要慎用！如果对不完整的文档开启upsert，那么当记录不存在时，自动创建的也是不完整的记录。有点儿绕口，慢慢体会吧！

更新文档——MULTI模式

```
// 更新：指定第四个参数为true可以开启multi模式
db.users.update({ "name" : "joe" }, joe, true, true);
```

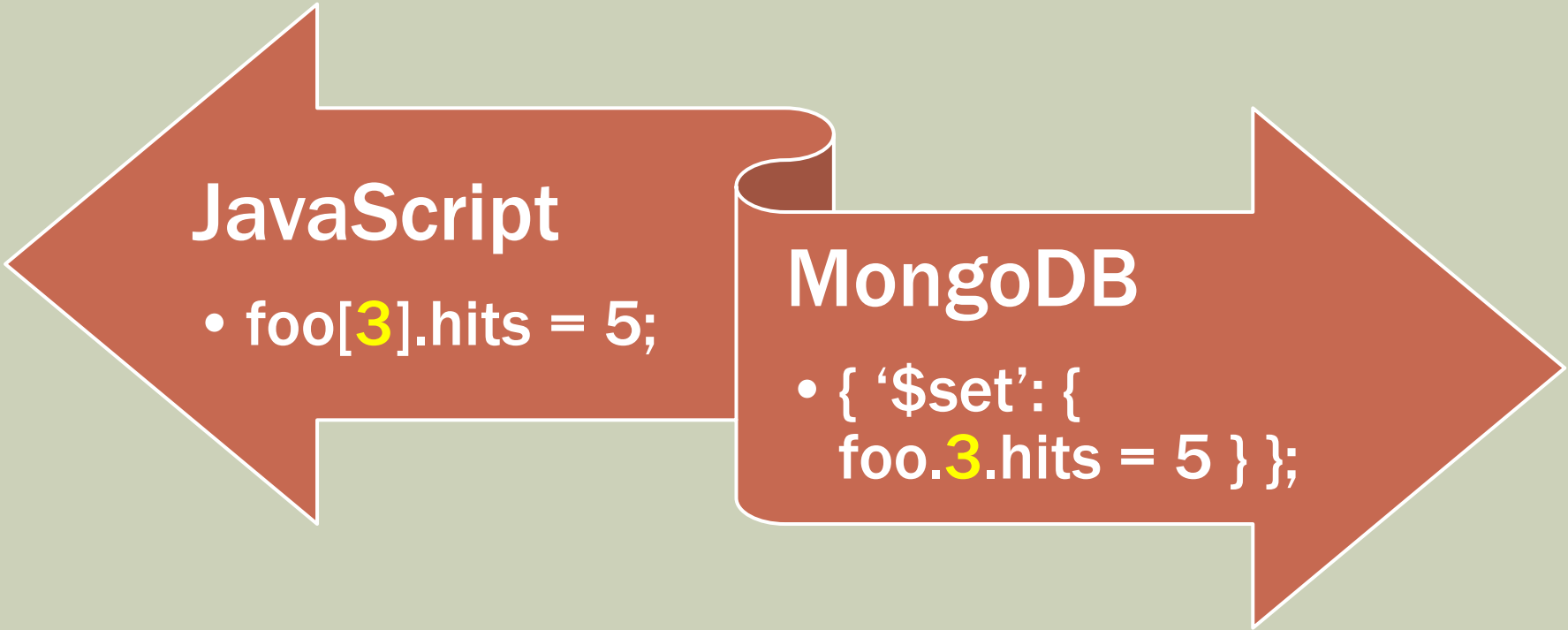
默认情况下update只会更新第一个匹配到的文档，开启multi模式，才会更新所有匹配到的文档。

老陈再次提醒大家：MongoDB的update操作是替换整个文档的，而不是对文档做出局部修改。如果要单独修改一个或多个键或值，请使用*修改器*。

更新文档——修改器

修改器	说明	备注
$inc	增加或减少数字的值，键不存在时会自动创建	数字
$set	设置指定键的值，键不存在时会自动创建	
$unset	$set的反操作，会删除键及键值	
$push	将元素追加到数组末尾，数组不存在会自动创建	数组
$pushAll	$push的批量操作版本	数组
$addToSet	同$pushAll，但会自动过滤重复元素	数组
$pop	{$pop : {key : 1}}——从数组末尾移除元素 {$pop : {key : -1}}——从数组开头移除原色	数组
$pull	从数组中移除所有匹配的元素	数组
$pullAll	$pull的批量操作版本	数组
$rename	修改指定键的键名	键
$bit	对*整形*键值执行位操作“与”、“或”等	数字

定位操作符（”$”）——神马是定位？



将数组foo中下标为3的元素的hits值设置为5
上图演示了JavaScript和MongoDB的不同实现方法

定位操作符（”$”）——未知？

*上一节中，注意标黄的数字3了吗？*对，就是它！3在两种语法中都表示数组下标。通过$set等修改器，我们可以对数组中的任意元素进行操作，然而问题在于，我们不知道某个元素的下标时该怎么办？？

不要慌，钱来啦！呃，不是，是”$”来啦！它用来定位查询文档已经匹配的元素，使用方法则很简单，就是替换上图MongoDB语法中的3即可。

```
// $inc操作
db.blog.update(
        {"post" : post_id},
        {"$inc" : {"comments.$.votes" : 1} }
);

// $set操作
db.blog.update(
        {"comments.author" : "John"},
        {"$set" : {"comments.$.author" : "Jim"} }
);

/* 注意：定位符只会更新第1个匹配到的元素！！！ */
```

FINDANDMODIFY——有神马不同？

从字面意思上理解findAndModify与update似乎很雷同，但其实不然，它们之间有着很大的差别。

FINDANDMODIFY——返回已更新的文档

关于findAndModify这里不做实例了，老陈翻译了一下它的几个参数的说明（这里以MongoDB官方网站为准，书中的内容已经过时）：

参数	说明	默认值
query	查询条件，用来确定匹配的项	{}
sort	如果匹配到多个文档，这里可以指定排序方式	{}
remove	表示是否删除匹配到的文档	N/A
update	更新操作。	N/A
new	如果设置为true，将返回更新后的文档。 注意，此参数在remove为true时无效。	false
fields	请参考http://www.mongodb.org/display/DOCS/Retrieving+a+Subset+of+Fields	All fields
upsert	如果设置为true，则在匹配项不存在时就自定创建 《MongoDB 权威指南》中的内容已经过时	false

FINDANDMODIFY——实现自增标识

在第一集中我们提到MongoDB没有内置的自增标识支持，怎么办呢？MongoDB官方提供了一个办法，使用findAndModify来实现自增标识，再加上findAndModify本身的操作是原子性的，不存在并发问题，因此很安全。下面是shell中的实现：

```
// 以下操作为users集合返回一个新的自增标识的值
db.foo.findAndModify(
{
        'query': {"_id": "users"},
        // 标识增量为 1
        'update': { $inc: { "seq": 1 } },
        // 返回修改后的值
        'new': true
});
```

参考地址：http://www.mongodb.org/display/DOCS/Object+IDs#ObjectIDs-SequenceNumbers

其他

瞬时完成

- 无需等待执行结果
- 操作完成后不会再向客户端反馈结果
- 优点：速度快
- 缺点：无法保证数据完整性

安全操作

- 安全操作会等待服务器的操作执行完毕并判断有无异常发生
- 优点：可以保证数据完整性
- 缺点：速度慢

请求和连接

- MongoDB内部维护一个连接队列，保证每个连接永远只有1个对应的客户端
- 在同一个连接内不会产生并发问题
- 缺点：多个连接之间存在并发问题
- 一些C#、Ruby、Python、Java等驱动库内置了连接池机制，应当注意由此带来的并发问题！

第三天：学会FIND

——NewSQL中文社区系列课程

陈彦铭出品

NewSQL中文社区出品
社区：http://newsql.cn/
ＱＱ：705287(ymind)
Ｑ群：79377097(NoSQL三群)

教学内容

- 认识find
- 子键筛选
- 复杂查询
- 正则表达式
- 数组查询
- 游标
- $where
- 排序分页
- 结束语

认识FIND

作用	SQL	MongoDB
所有记录	SELECT * FROM users	db.users.find()
age=33的记录	SELECT * FROM users WHERE age=33	db.users.find({age:33})
子键（字段）筛选	SELECT a, b FROM users WHERE age=33	db.users.find({age:33}, {a:1,b:1})
排序	SELECT * FROM users WHERE age=33 ORDER BY name	db.users.find({age:33}).sort({name:1})
比大小	SELECT * FROM users WHERE age>33	db.users.find({'age':{$gt:33}})})
正则（模糊匹配）	SELECT * FROM users WHERE name LIKE "Joe%"	db.users.find({name:/^Joe/})
忽略、限制	SELECT * FROM users LIMIT 10 SKIP 20	db.users.find().limit(10).skip(20)
Or操作	SELECT * FROM users WHERE a=1 or b=2	db.users.find({ $or : [{ a : 1 } , { b : 2 }] })
仅返回1条(TOP 1)	SELECT * FROM users LIMIT 1	db.users.findOne()
Distinct聚合	SELECT DISTINCT last_name FROM users	db.users.distinct('last_name')
Count聚合	SELECT COUNT(AGE) from users	db.users.find({age: {'$exists': true}}).count()
查询计划	EXPLAIN SELECT * FROM users WHERE z=3	db.users.find({z:3}).explain()
引用自：http://www.mongodb.org/display/DOCS/SQL+to+Mongo+Mapping+Chart		

这里仅简单的列举了几种常用的查询形式，后面会详细阐述。

老陈：**MongoDB的查询语句仅允许包含常量**，无法对文档或其他文档中的键值进行引用。这显然很不方便，但反过来想想，这样的约束可以迫使开发者做出更高性能的架构（在MongoDB世界里大家要学会发扬数据冗余的设计理念！越来越多的开发者对此表示不满，不了解10gen会不会妥协）。

子键筛选

子键筛选也就是我们在SQL数据库中见怪不怪的字段筛选。

然而，由于MongoDB采用的是基于Json的查询语句，所以其子键筛选显得非常怪异（限于篇幅，请留意示例代码中的注释）：

```
// 包含所有子键
db.users.find({age:33})

// 仅包含子键：_id、a、b
db.users.find({age:33}, {a:1, b:1});

// 仅包含子键：_id、a
db.users.find({age:33}, {a:1, b:0});

// 仅包含子键(这里给出了剔除_id的方法)：a
db.users.find({age:33}, {a:1, b:0, _id:0});
```

复杂查询

MongoDB可以支持一些比较操作（大于小于）、子句（or）、取反（not）等查询，同样的，请观察示例代码：

```
// 还记得上一节中我们提到的操作符吗（$all, $exists, $in等）？
// 实际上它们都是查询操作符！

// 返回年龄大于等于33岁，且小于40岁的用户：
db.users.find({ age: { '$gte' : 33, '$lt' : 40 } });

// 返回年龄大于33岁，且不等于40岁的用户：
db.users.find({ age: { '$gte' : 33, '$ne' : 40 } });

// 返回年龄等于33岁，或等于40岁的用户：
db.users.find({ '$or': [ {'age' : 33}, {'age' : 40} ] });
```

注意：默认情况下，MongoDB执行的是and操作，即所有指定的条件都是与的关系。只有在特别的情况下，才会使用或（or），而$or就是MongoDB的或子句。

查询条件操作符

	操作符	说明	示例
1	$lt, $lte $gt, $gte	<, <=, >, >=	db.things.find({ “a" : { $gt: value } });
2	$all	数组中的元素是否完全匹配	db.things.find({ a: { $all: [2, 3] } });
3	$exists	可选：true，false	db.things.find({ a : { $exists : true } });
4	$mod	取模：a % 10 == 1	db.things.find({ a : { $mod : [10 , 1] } });
5	$ne	取反：即not equals	db.things.find({ x : { $ne : 3 } });
6	$in	类似于SQL的IN操作	db.things.find({j:{$in: [2,4,6]}});
7	$nin	$in的反操作，即SQL的 NOT IN	db.things.find({j:{$nin: [2,4,6]}});
8	$nor	$or的反操作，即不匹配(a或b)	db.things.find({ name : "bob" , $nor : [{ a : 1 } , { b : 2 }] })
9	$or	Or子句，注意$or不能嵌套使用	db.things.find({ name : "bob" , $or : [{ a : 1 } , { b : 2 }] })
10	$size	匹配数组长度	db.things.find({ a : { $size: 1 } });
11	$type	匹配子键的数据类型，详情请看	db.things.find({ a : { $type : 2 } });

正则表达式

MongoDB的正则表达式实现了SQL中的通配符匹配的机制，而且比SQL更加强大。不过可惜的是，只有类似于/^.+/这样的前缀型正则表达式才能够使用索引，其他形式都会做全集合扫描。

MongoDB的正则表达式语法与JavaScript完全一致。

```
// 正则匹配方法之一
db.customers.find( { name : /acme.*corp/i } );

// 正则匹配方法之二
db.customers.find( { name : { $regex : 'acme.*corp', $options: 'i' } } );

// 与其他条件联合使用
db.customers.find( { name : { $regex : /acme.*corp/i, $nin : [`abc'] } } );
```

注意：MongoDB的正则表达式可以匹配其自身。如果您将正则表达式存入了MongoDB，那么查询的时候，它可以匹配它自身！

数组查询

MongoDB数组查询，除了之前提到的$all和$size等，还有：

- $size
- $slice
- 子项定位
- $elemMatch

数组查询——$SIZE

$size用来匹配数组长度（即最大下标）：

```
// 返回comments包含5个元素的文档
db.posts.find({}, {comments:{'$size': 5}});
```

请注意：$size操作符并不能与其他查询操作符（子句）联合使用，这意味着您不能使用如下代码：

```
// 以下的使用方法是错误的
db.posts.find({}, {comments:{'$size': { '$gt': 5 }}});
```

老陈也不了解MongoDB为何要作如此限制，或者说没有实现这样的功能。不支持这样的语法确实非常不给力。
但并非就没有办法了，MongoDB官方建议的做法是将数组长度作为独立属性写入，然后直接比对即可。貌似MongoDB在强迫开发者使用冗余数据以提升执行效率（事实上，MongoDB类似的做法还有很多）。

```
// 使用冗余字段来实现
db.posts.find({}, {'commentCount': { '$gt': 5 }});
```

数组查询——$SLICE

$slice操作符类似于子键筛选，只不过它筛选的是数组中的项。老陈还得遗憾的告诉您，$slice看起来不错，但其实还不算够用！

```
// 仅返回数组中的前5项
db.posts.find({}, {comments:{'$slice': 5}});

// 仅返回数组中的最后5项
db.posts.find({}, {comments:{'$slice': -5}});

// 跳过数组中的前20项，返回接下来的10项
db.posts.find({}, {comments:{'$slice': [20, 10]}});

// 跳过数组中的最后20项，返回接下来的10项
db.posts.find({}, {comments:{'$slice': [-20, 10]}});
```

有童鞋问了：“很给力啊，哪里不够用呢？”
老陈：“呵呵，如果你要返回的项在不确定的位置，怎么办？”
——这只是我的一个疑问，至于生产环境是否真的有这种需求，尚未可知哇！

数组查询——子项定位

细心的童鞋已经发现，数组的子项定位已经在上一个章节讲过了。是的，是讲过了，但我还是要再提一下，加深印象！

MongoDB允许在查询中指定数组的下标，以实现更加精确的匹配。

本小节有且仅有一个简单的代码示例：

```
// 返回comments中第1项的by子键为Abe的所有文档
db.blogposts.find( { "comments.0.by" : "Abe" } );
```

数组查询——$ELEMMATCH

童鞋们还记得上一集中我们提到的数组定位修改器中除了数组下标还有一个“$”符号吗？该符号在find中就不灵了，怎么办呢？

那就只有考虑使用$elemMatch操作符了，多说无益，看代码：

```
// 有这样一个包含了2个文档的集合，仔细观察
// 现在我们需要找到 shape = "square" 且 color = "purple" 的那项
{ "foo" : [ { "shape" : "square", "color" : "purple", "thick" :
false }, { "shape" : "circle", "color" : "red", "thick" : true } ]};
{ "foo" : [ { "shape" : "square", "color" : "red", "thick" : true },
{ "shape" : "circle", "color" : "purple", "thick" : false } ]};

// 这两个查询都会同时匹配两个文档
db.foo.find({"foo.shape": "square", "foo.color": "purple"});
db.foo.find({foo: {"shape": "square", "color": "purple"} });

// 这才是我们想要的
db.foo.find({foo: {"$elemMatch": {shape: "square", color:
"purple"}}});
```

游标

什么？MongoDB也有游标？做SQL的时候我们最讨厌游标了，不要提它，恨ING~~

是吧！听我说完先，MongoDB是使用游标来返回find的执行结果的。使用游标可以略过部分查询结果、限定返回项数、对结果执行排序等等。

下面的代码演示了如何在shell中获取一个游标的实例，并用它做一些迭代操作：

```
// 取得游标实例
var cursor = db.collection.find();

// 迭代
while (cursor.hasNext())
{
        var obj = cursor.next();

        // ...
}
```

老陈：实际上游标在各种驱动中已经封装的很好了。后面还会讲解MongoDB的一种特殊的被称作“尾部游标”的东东。

注意：MongoDB游标的实现机制导致find时返回的数据不一定就是即时数据。比如刚刚被更改过的记录可能会有所延缓。

$WHERE

$where操作符被大多数人看做一个鸡肋功能。因为它根本无法使用索引，且执行效率极低。

$where操作符是为了弥补其他查询操作符无法使用变量、无法引用子键及键值等问题而诞生的。它允许在查询中使用任意JavaScript脚本。

$where的工作机理：每个文档都会从BSON转换为JavaScript对象，然后运行$where指向的脚本（就像是委托），因此也无法使用索引，其效率低得可怜，不到万不得已，千万别用这玩意儿。

```
// 普通实现
db.myCollection.find( { a : { $gt: 3 } } );

// $where实现的方法之一
db.myCollection.find( { "$where" : "this.a > 3" } );

// $where实现的方法之二
db.myCollection.find("this.a > 3");

// $where实现的方法之三
db.myCollection.find(function()
{
    // 很明显，返回true时才会选定当前记录
    return this.a > 3;
});
```

排序分页

limit

- 限制返回结果的项目数量的上限
- 可以理解为每页最多N项

skip

- 在返回时略过的项目数量的上限
- 可以变通的实现翻页

sort

- 指定排序方式
- 可以支持多键

排序分页——排序比较的顺序

老陈：理解比较顺序（针对数据类型而言）对于编写排序代码非常有帮助。

排序分页——分页示例

```
// 这是一个非常简单的带有排序的分页示例
db.stock.find({
      'desc' : 'mp3',
      'status' : 0
})
// 每页最多10项
.limit(10)
// 略过100项
.skip(100)
// 排序方式
.sort({
      'price' : -1, // 价格降序
      'updateDate' : 1 // 更新日期升序
});
```

排序分页——不使用SKIP分页

在MongoDB中使用skip略过大量数据（多大量？尚未可知，俺没测试。）时会非常缓慢，那么不使用skip又该如何分页呢？官方给出的建议是使用一个可比大小的排序字段，并根据取值区间来模拟实现分页。

例如：如果您是按照时间升序排序的，那么从最小时间开始取10条就是第一页，再将这10条中的最后一条的时间作为下一页的起始，如此往复。

老陈：在MongoDB官方组织的2012年北京聚会上我们也向10gen请教了这个问题。不过10gen工程师给出的答案让我们不容乐观。他们建议采用“上一页”、“下一页”这样的翻页方式，而不要去尝试计算最大页码。

本小节不提供示例

结束语

基本查询

- find
- Findone
- 各种$操作符

数组查询

- $size
- $slice
- 子项定位
- $elemMatch

排序分页

- limit, skip, sort
- 不用skip分页
- 随机查询

第四天：使用索引

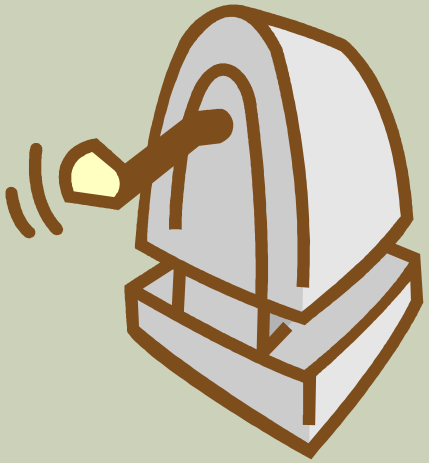
——NewSQL中文社区系列课程

陈彦铭出品

NewSQL中文社区出品
社区：http://newsql.cn/
ＱＱ：705287(ymind)
Ｑ群：79377097(NoSQL三群)

教学内容

- 创建索引
- 修改索引
- 删除索引
- 唯一索引
- 空间索引
- 查询计划
- 强行索引
- 实战技巧



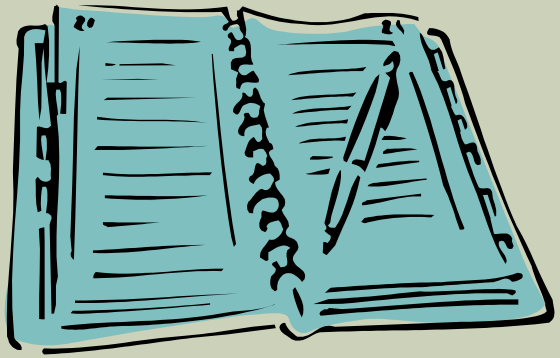
创建索引

```
// 创建索引
db.things.ensureIndex({
    'j': 1
});

// 创建子文档索引
db.things.ensureIndex({
    'user.Name': -1
});

// 创建复合索引
db.things.ensureIndex({
    'j': 1, // 升序
    'x': -1 // 降序
});

/*
    如果您的find操作只用到了一个键，那么索引方向是无关紧要的；
    当创建复合索引的时候，一定要谨慎斟酌每个键的排序方向。
*/
```



MongoDB的索引机制与传统的关系型数据库索引几乎是一样的，绝大多数优化SQL索引的技巧也都适用于MongoDB。

索引会增加数据插入、更新和删除的性能开销，应避免为每个键都创建索引。

修改索引

```
// 修改索引，只需要重新运行
// 索引命令即可
// 如果索引已经存在则会重
// 建，不存在的索引会被添加
db.things.ensureIndex({
    // 原来的索引会重建
    'user.Name': -1,
    // 新增一个升序索引
    'user.Name': 1,
    // 为Age新建降序索引
    'user.Age': -1
}, {
    // 打开后台执行
    'background': true
});

// 重建索引
db.things.reIndex();
```

注意：MongoDB不会检查要索引的键名是否真的存在。

当background为false或未指定时，创建索引的过程会以阻塞的形式运行，此时MongoDB无法处理任何请求。

老陈：如果已经为子键创建了索引，那么其包含的子文档就不需要索引了。比如左侧的user如果被索引了，那么再创建诸如“user.Name”之类的索引就是徒劳了，而且浪费空间！

删除索引

```
// 删除集合中的所有索引
db.things.dropIndexes();
// 删除指定键的索引
db.things.dropIndex({
    x: 1,
    y: -1
});
// 使用command删除指定键的索引
db.runCommand({
    dropIndexes:'foo',
    index : { y: 1 }
});
// 使用command删除所有索引
db.runCommand({
    dropIndexes:'foo',
    index : '*'
});
```

提示：如果删除整个集合（drop）也会删除集合中的索引。如果是删除集合中所有的文档（remove）则不会影响索引，当有新文档插入时，索引就会重建。

唯一索引

```
// 创建唯一索引，同时这也
// 是一个符合唯一索引
db.things.ensureIndex(
{
    'firstName': 1,
    'lastName': 1
}, {
    // 指定为唯一索引
    'unique': true,

    // 删除重复记录
    'dropDups': true
});
```

注意：

- _id本身就是个唯一索引，不同的是，它不能被删除；
- 删除重复记录时，仅会按顺序保留第一文档；
- 如果创建了唯一索引，但文档并不包含该键，默认以null存储，因此如果发现了第二项未指定该键或该键被指定为null的记录，就会被认为是重复的；
- 所谓复合唯一索引，允许单键重复，但组合起来之后是不允许重复的；

地理空间索引——创建(1)

```
// 文档格式
{
  '_id': 1,
  'name': '天安门',
  'location':[
    116.12345,
    39.54321
  ]
}

// 创建地理空间索引
db.map.ensureIndex({
  'location': '2d'
});
```

不了解神马是地理空间的童鞋，就可以不要往下看了，先找baidu大爷吧！

MongoDB为处理地理空间信息增加了一个专门的索引结构——2d，对就是二弟！

要使用二弟索引，表示地理坐标的键必须是一个长度为2的数组。然后为该键创建取值为“2d”的索引即可。请看左侧代码实例。

地理空间索引——创建(2)

```
// 创建地理空间索引
db.map.ensureIndex({
  'location': '2d'
}, {
  'min': -10000,
  'max': 10000
});
```

MongoDB的地理空间索引的取值没有单位，这意味着你可以为其设置任何单位。默认地理空间索引的假设值是-180 ~ 180。

如果需要自定义其他的取值范围，请参考左侧代码实例。

老陈：MongoDB中，每个集合只能包含1个2d索引，不过，在v1.9+之后，可以支持多个2d索引了。

地理空间索引——$NEAR

```
// $near操作符
db.places.find({
  loc : {
    // 中心点
    $near : [50, 50],
    // 查询半径
    $maxDistance : 5
  }
}).limit(20);
// 如果没有指定limit，其默认值为100
```

$near操作符与geoNear Command的区别在于：geoNear会同时返回每个地标与给定中心点的距离（注意，这里的距离没有单位）。

地理空间索引——$WITHIN(BOX)

```
// 定义一个矩形区域
var box = [
  // 左上角
  [40.73083, -73.99756],
  // 右下角
  [40.741404, -73.988135]
];

// 查找
db.places.find({
  "loc" : {
    "$within" : {
      "$box" : box
    }
  }
});
```

$within操作符支持以下几个选项：

- $box：矩形查找
- $center：圆形查找
- $polygon：多边形查找（v1.9+支持）

地理空间索引——$WITHIN(CENTER)

```
// 定义中心点
var center = [50, 50];
// 定义查找半径
var radius = 10;

// 查找
db.places.find({
  "loc" : {
    "$within" : {
      // 注意这里是数组传递
      "$center" : [
        center,
        radius
      ]
    }
  }
});
```

$within操作符支持以下几个选项：

- $box：矩形查找
- $center：圆形查找
- $polygon：多边形查找（v1.9+支持）

地理空间索引——$WITHIN(POLYGON)

```
// 以数组的形式定义一个多边形
var polygonA = [
    [10, 20],
    [10, 40],
    [30, 40],
    [30, 20]
];

// 以文档的形式定义一个多边形
var polygonB = {
    a: { x: 10, y: 20 },
    b: { x: 15, y: 25 },
    c: { x: 20, y: 20 }
};

// 细心观察，可以发现，查询时是一样的
db.places.find({
    "loc": {
        "$within": { "$polygon": polygonA }
    }
});
db.places.find({
    "loc": {
        "$within": { "$polygon": polygonB }
    }
});
```

$within操作符支持以下几个选项：

- $box：矩形查找
- $center：圆形查找
- $polygon：多边形查找（v1.9+支持）

老陈：很抱歉，本页代码示例较大，因此采用截图的方式，影响了布局的整体一致性。原不原谅我木有关系，你看懂了才是真的！

地理空间索引——球形查找

木有代码，哭去吧！

为什么会有球形查找，老陈都懒得说了，地球不是平面的，也不是方的，是个不太规则的球。

球形查找，请记住以下内容：

- **$nearSphere**
- **$centerSphere**

看起来眼熟吧，不解释了，参考前面的内容理解。

以命令的方式做球形查找也很简单，只需要增加一句代码即可：**spherical: true**

查询计划

```
// 使用查询计划
db.users.find({}).explain();

// 输出结果
{
  "cursor" : "BasicCursor",
  "indexBounds" : [ ],
  "nscanned" : 57594,
  "nscannedObjects" : 57594,
  "nYields" : 2,
  "n" : 3,
  "millis" : 108,
  "indexOnly" : false,
  "isMultiKey" : false,
  "nChunkSkips" : 0
}
```

几个参数说明：

- cursor：指示查询是否使用了索引；
  - BasicCursor：未使用索引；
  - BtreeCursor：使用了Btree索引，其后往往紧跟着用到的索引名称，同时indexBounds则会包含使用的索引详情；
- nscanned：查找过程中扫描的文档数；
- n：返回结果中包含的文档数；
- millis：表示查询耗时的毫秒数；
- allPlans：包含所有可能的查询方案；

强制使用索引

```
// 强制使用索引a和b
db.collection.find({
  'a': 4,
  'b': 5,
  'c': 6
}).hint({
  'a': 1,
  'b': 1
});

// 强制不使用任何索引
db.collection.find().hint({
  '$natural': 1
});
```

通过查询计划，我们可以测试评估出最佳的查询方案，然而在某些情况下，MongoDB仍然无法从诸多索引中选择最好的一个。此时可以手工指定。

实战技巧

总结

- 索引可以加速查询；
- 单个索引无需在意其索引方向；
- 多键索引需要慎重考虑每个索引的方向；
- 做海量数据更新时应当先卸载所有索引，待数据更新完成后再重建索引；
- 不要试图为每个键都创建索引，应考虑实际需要，并不是索引越多越好；
- 唯一索引可以用来消除重复记录；
- 地理空间索引是没有单位的，其内部实现是基本的勾股定理算法。

第五天：聚合操作

——NewSQL中文社区系列课程

NewSQL中文社区出品
社区：http://newsql.cn/
ＱＱ：705287(ymind)
Ｑ群：79377097(NoSQL三群)

陈彦铭出品

教学内容

- count
- distinct
- group
- MapReduce

聚合操作——COUNT

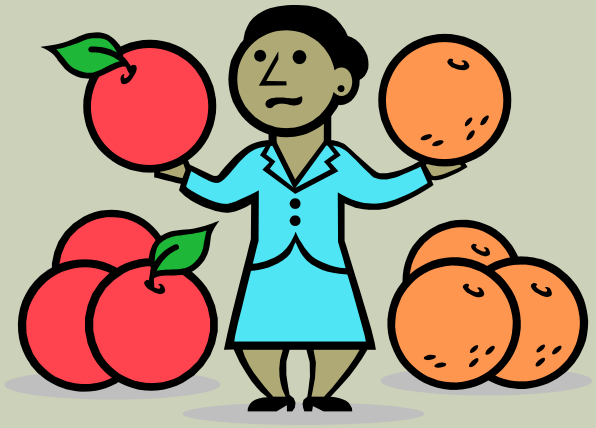
```
// count操作
db.collection.count({
  'status': 1
}
```

count是最简单的聚合操作了，有人不懂吗？回家面B死过5分钟.....

创建索引会大大提升count的性能。

聚合操作——DISTINCT

```
// distinct
db.address.distinct(
  "zip-code",
  {
    age : 30
  }
);
```



distinct是用来找出给定条件的所有不同的值的。

聚合操作——GROUP

```sql
-- sql伪代码
SELECT a, b, SUM(c), CSUM
FROM COLL
WHERE active = 1
GROUP BY a, b

// MongoDB
db.coll.group({
  'key': {
    'a': true,
    'b': true
  },
  'cond': { 'active':1 },
  'reduce': function(obj, prev)
  {
    prev.csum += obj.c;
  },
  'initial': { 'csum': 0 }
});
```

group是较为复杂的聚合操作，它与关系型数据库中的GROUP BY类似，但更加强大。

group先选定分组依据的键，然后将集合根据键值的不同分成若干组。之后再聚合每一组内的文档产生查询结果。

参数说明：
key：分组依据；
cond：查询条件；
reduce：聚合操作；
initial：指定聚合计数器的初始对象；

聚合操作——GROUP完成器

```
// 查询
db.test.group({
    'cond': {
        "invoked_at.d": {
            '$gte': "2009-11",
            '$lt': "2009-12"
        }
    },
    'key': { 'http_action': true },
    'initial': { 'count': 0, 'total_time': 0 },
    'reduce': function(doc, out) {
        out.count++;
        out.total_time += doc.response_time;
    },
    'finalize': function(out) {
        out.avg_time = out.total_time / out.count;
    }
});

// 结果
[
    {
        "http_action": "GET /display/DOCS/Aggregation",
        "count": 1,
        "total_time": 0.05,
        "avg_time": 0.05
    }
]
```

完成器可以用来精简结果集。也就是对结果集进行进一步的处理。

之所以会提供这个方法，是因为MongoDB的group操作有个特殊要求，即：返回结果必须能够完全包含在1个文档之内（之前是4M，现在是16M），所以就必须允许对结果集进行精简处理。

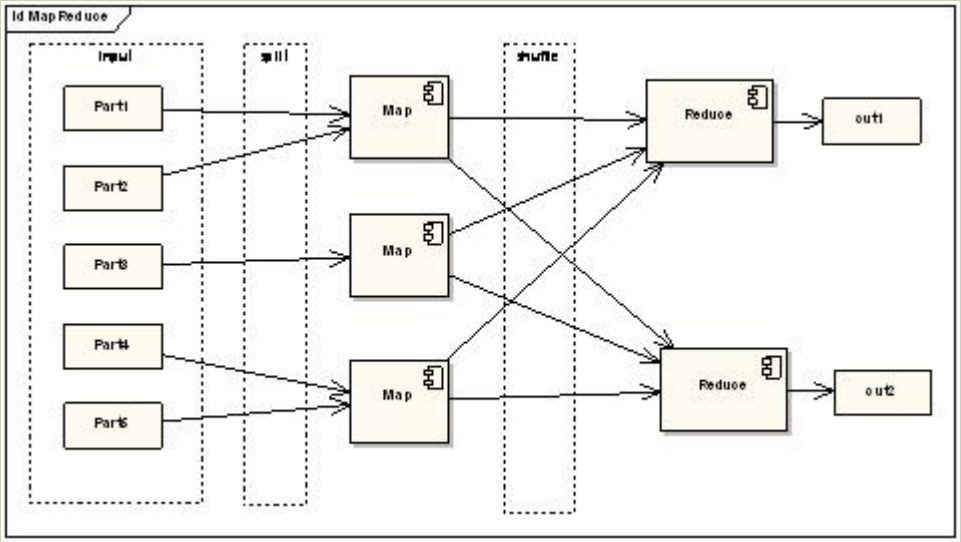
聚合操作——GROUP键函数

```
// 查询
db.test.group(
{
    'cond': { "x": 0 },
    // 将函数作为键使用
    '$keyf': function(doc) { return { "x": doc.x + '...' }; },
    'initial': { 'count': 0 },
    'reduce': function(doc, out) {
        out.count++;
    }
});
```

$keyf操作符可以将函数的返回值当作键来引用；

聚合操作——MAPREDUCE(简介)

参数	释义
map	映射操作
reduce	化简操作
query	数据筛选条件
sort	发往map的文档排序 注意，此排序是指进入MapReduce操作之前的排序，而不是最终结果集。
limit	发往map的文档数量
output	看下一节
keeptemp	是否保留临时集合
finalize	同group的完成器
scope	全局变量
verbose	是否生成详尽日志



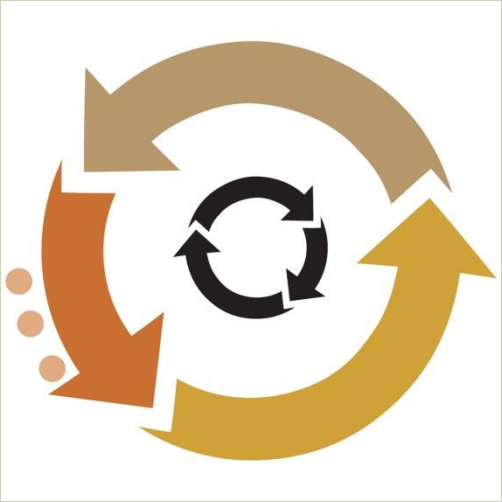
MapReduce是一种编程模型，用于大规模数据集（大于1TB）的并行运算。概念"Map（映射）"和"Reduce（化简）"，和他们的主要思想，都是从函数式编程语言里借来的，还有从矢量编程语言里借来的特性。他极大地方便了编程人员在不会分布式并行编程的情况下,将自己的程序运行在分布式系统上。

简单说来，一个映射函数就是对一些独立元素组成的概念上的列表（例如，一个测试成绩的列表）的每一个元素进行指定的操作。

摘自：http://baike.baidu.com/view/2902.htm

聚合操作——MAPREDUCE(OUTPUT)

取值	定义
集合名	同replace
replace: ‘集合名’	完全替换临时数据集
merge:’集合名’	合并临时数据集
reduce:’集合名’	再化简重复记录 当临时数据集存在了相似记录，则会继续执行reduce操作进一步化简
inline:1	内存模式 此时不会再产生临时集合



v1.74之前，如果指定了output的值，则隐含keeptemp:true

v1.74之后的定义请看左侧表格。

联机文档：
http://www.mongodb.org/display/DOCS/MapReduce

第六天：进阶指南

——NewSQL中文社区系列课程

陈彦铭出品

NewSQL中文社区出品
社区：http://newsql.cn/
ＱＱ：705287(ymind)
Ｑ群：79377097(NoSQL三群)

教学内容

- 数据库命令
- 定容集合
- 尾部游标
- GridFS
- 服务器端脚本
- 数据库引用

数据库命令——概览

buildInfo	collStats	distinct
drop	dropDatabase	dropIndexes
findAndModify	getLastError	isMaster
listCommands	listDatabases	ping
renameCollection	repairDatabase	serverStatus

MongoDB目前支持的命令数量已经超出了100多个，这里列出了一些常用命令。

用法：db.runCommand({命令, 参数});

数据库命令——BUILDINFO

```
{ “buildInfo” : 1 }
```

返回MongoDB的版本号和宿主服务器的操作系统信息。

数据库命令——COLLSTATS

{ “collStats” : “集合名称” }

返回指定集合的统计信息，包含：

- 数据大小
- 已分配的空间大小
- 索引大小

数据库命令——DROP

{ “drop” : “集合名称” }

删除当前数据库及其所有内容

数据库命令——DROPINDEXES

{ “dropIndexes”：“集合名称”,
“index”：“索引名称” }

删除指定集合中具有指定名称的索引

数据库命令——GETLASTERROR

{ “getLastError” : 1 }

查看最后一次操作的错误信息

数据库命令——ISMASTER

{ “isMaster” : 1 }

检查当前服务器是否为主服务器

数据库命令——LISTCOMMANDS

{ “listCommands” : 1 }

列出当前服务器支持的所有命令

数据库命令——LISTDATABASES

{ “listDatabases” : 1 }

列出服务器上所有的数据库

数据库命令——PING

{ “ping” : 1 }

检查服务器连接是否正常
该命令会忽略锁机制

数据库命令——RENAMECOLLECTION

{ “renameCollection” : a, “to” : b }

将集合a重命名为b
请注意，这里的a、b必须是完整的带有命名空间的集合名，例如”foo.bar”

数据库命令——REPAIRDATABASE

{ “repairDatabase” : 1}

修复并压缩当前数据库

数据库命令——SERVERSTATUS

{ “serverStatus” : 1}

返回当前服务器的状态信息

定容集合

```
db.createCollection("my_collection", {
capped: true, size: 100000});
```

定容集合顾名思义，就是容量大小不会改变的集合。

注意：

- 如果空间不足，插入新文档时会自动删除最早的记录；
- 在定容集合中，不允许手工删除文档；
- 导致文档位置发生变化的更新操作将会被拒绝；
- 默认情况下，定容集合没有索引，”_id”也是没有索引的，当然，索引是可以手工创建的；

定容集合——自然排序

```
db.my_collection.find()
.sort({"$natural" : -1})
```

自然顺序是指文档在磁盘上的物理排序。

$natural的取值为1时表示与默认顺序相同，取值为-1时则刚好相反

定容集合——尾部游标

```
$cursor = $collection->find()->tailable();
```

尾部游标的特点是连接持久化，即在明确指定的情况下，不会自动关闭。这意味着尾部游标不会在没有返回结果时自动销毁。

当关联的集合内有新文档被添加时，尾部游标就会被触发。

请参考：http://www.mongodb.org/display/DOCS/Tailable+Cursors

GRIDFS

GridFS

GridFS是MongoDB提供的用于存储大型二进制数据的机制。它本身就是一个分布式文件系统：

- GridFS会直接利用已建立的复制、分片机制；
- GridFS可以避免传统文件系统的某些弊端，例如同一目录下文件数量过多；
- GridFS不会产生磁盘碎片；

关于GridFS的具体操作，这里不详述，请参考：
http://www.mongodb.org/display/DOCS/GridFS

GRIDFS——原理

```
{ "_id" : ObjectId("..."),
 "n" : 0,
 "data" : BinData("..."),
 "files_id" : ObjectId("...") }
```

GridFS是简历在MongoDB普通文档基础之上的轻量级分布式文件存储规范。
GridFS的基本思想是，将大型文件切分成很多小块，每一块作为一个单独的文档存储。

在GridFS规范中定义了很多额外的键，最值得关注的是md5这个键，利用它可以做很多有趣的事情。

服务器端脚本

db.eval(“function() { return 5 + 6; }”)

利用db.eval可以在MongoDB的服务器端执行任意JavaScript脚本。用处有很多，比如事务模拟。

重点要提的就是，如果传入的JavaScript脚本过大会给调试代码很多不便，此时可以将调试信息写入到数据库日志中，方法是在脚本中调用如下语句：

```
db.eval("print('Hello, world');");
```

服务器端脚本——存储JAVASCRIPT

```
db.system.js.insert({"_id" : "x", "value" : func})
```

存储JavaScript这个译名感觉怪怪的，但暂时也找不到别的译法了。

存储JavaScript就类似于sql数据库当中的存储过程。可以将JavaScript脚本保存在服务器端，需要的时候可以使用前面提到的db.eval调用。

存储JavaScript方便实现代码重用。不过老陈建议对这个玩意儿要慎用！搞不好会让你的代码逻辑显得很乱很糟糕！

数据库引用——DBREF

{“$ref”：“集合名称”，“$id”：“唯一标识键”，“$db”：“数据库”}

DBRef是作为内嵌文档来存在的，语法如上。它的作用就相当于HTML标记中a标签的href特性。

DBRef可以跨数据库引用文档，就如上面的代码演示的那样。

需要注意的是，以上代码中的键的顺序是不能改变的，必须是这样的顺序，只有$db键是可选的。

并不是所有的驱动都实现了DBRef机制，这个需要大家自行挖掘。

第七天：服务器管理

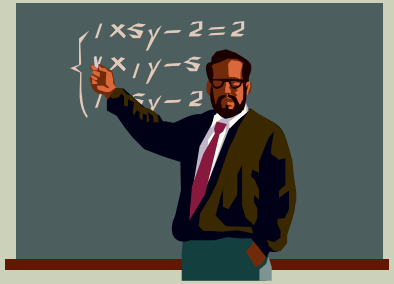
——NewSQL中文社区系列课程

陈彦铭出品

NewSQL中文社区出品
社区：http://newsql.cn/
ＱＱ：705287(ymind)
Ｑ群：79377097(NoSQL三群)

教学内容

- 启动和停止
- 配置文件
- 管理界面
- 安全与认证
- 备份与修复



启动和停止

启动，略过……

贸然的停止MongoDB引擎会导致数据丢失，想稳妥的终止MongoDB可以这么办：

- 如果是作为前端（shell方式）运行的，可以直接按下CTRL+C；
- 如果是Windows服务，可以在服务控制面板使用菜单停止；
- 通过驱动或shell关闭的方法是发送{“shutdown”：1}指令。

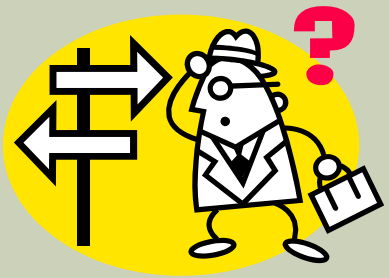
配置文件

MongoDB是支持使用配置文件的，只是以前我们不懂得欣赏……

mongod –config “D:\MongoSettings.conf”

```
#配置文件内容示例：
port = 27099
fork = true

logpath = D:\MongoDB\system.log
```

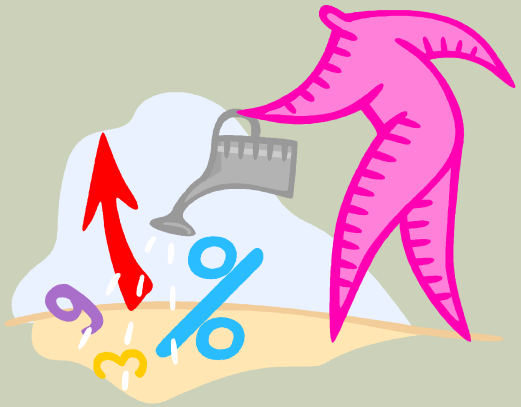


管理界面

MongoDB内置了一个监控和管理界面，如果只是简单的监控，可以在MongoDB实例的端口号（如10000）上+1000后（11000）在任何Web浏览器打开：

http://ipAddress:11000/

要使用管理功能，则需要为MongoDB引擎增加--rest参数



请不要使用驱动连接HTTP接口，那样做很无聊仨！

安全与认证

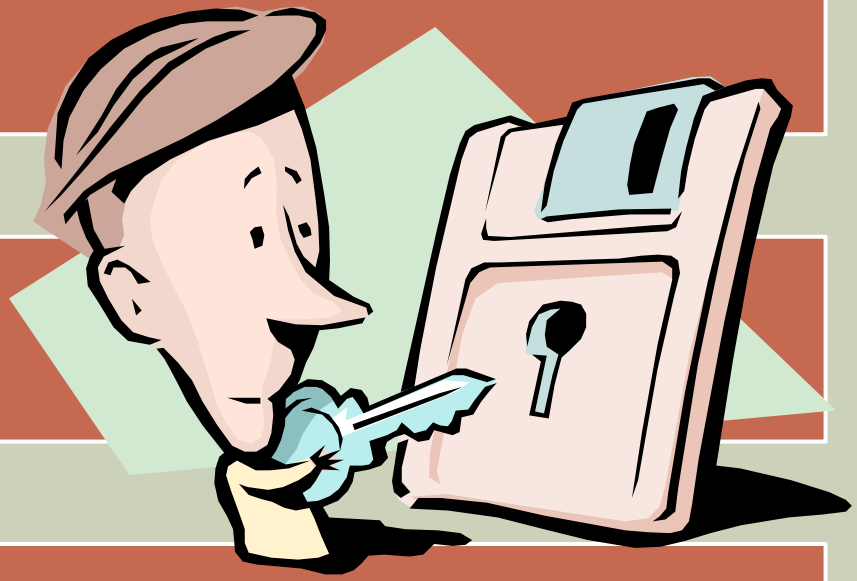
MongoDB有内置的安全认证功能，但用起来让人比较纠结：

1. 默认为无认证，启动用登录shell；
2. 添加账号；
3. 关闭shell、关闭MongoDB；
4. 为MongoDB增加—auth参数；
5. 重启MongoDB；
6. 登录shell，此时就需要认证了；

老陈：关于如何添加、修改以及删除账号信息，这里不再详述，请参考：
http://www.mongodb.org/display/DOCS/Security+and+Authentication

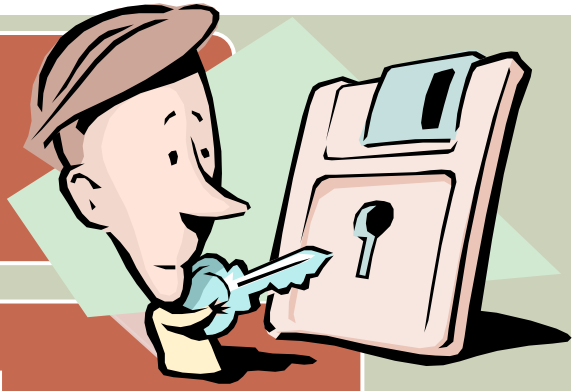
备份与恢复

- 冷备份
- 热备份
- 主从复制备份



备份与恢复——冷备份

关闭MongoDB引擎



拷贝数据库文件夹及文件

恢复时反向操作即可

优点：可以完全保证数据完整性；
缺点：需要数据库引擎离线

备份与恢复——热备份

保持MongoDB为运行状态

使用mongodump备份数据

使用mongorestore恢复数据

优点：数据库引擎无须离线
缺点：不能保证数据完整性，操作时会降低MongoDB性能

备份与恢复——主从复制备份

创建主从复制机制

配置完成后数据会自动同步

恢复途径很多

优点：可以保持MongoDB处于联机状态，不影响性能
缺点：在数据写入密集的情况下可能无法保证数据完整性

备份与恢复——修复

db.repairDatabase();

提示：

- 修复数据库还可以起到压缩数据的作用；
- 修复数据库的操作相当耗时，万不得已请不要使用；
- 建议经常做数据备份；

第八天：MONGODB复制

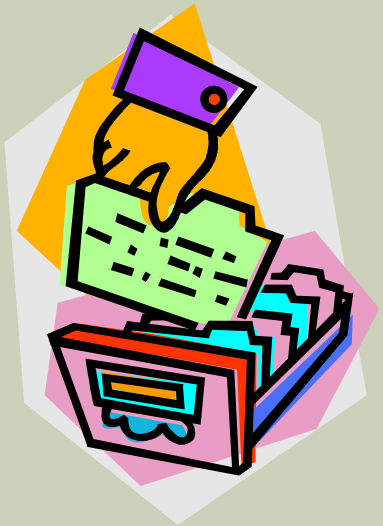
——NewSQL中文社区系列课程

NewSQL中文社区出品
社区：http://newsql.cn/
ＱＱ：705287(ymind)
Ｑ群：79377097(NoSQL三群)

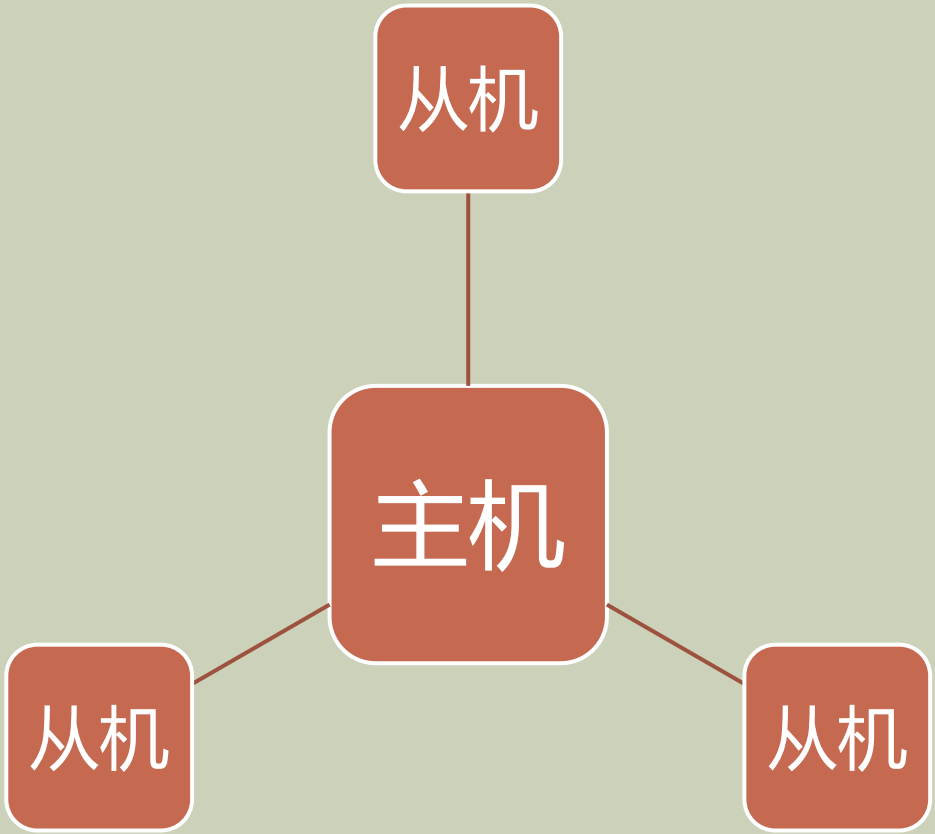
陈彦铭出品

教学内容

- 主从复制
- 副本集
- 故障转移
- 读写分离
- oplog
- 同步
- 诊断
- 认证



主从复制



MongoDB支持的主从复制模式：

- 一对多
- 多对多
- 多对一

貌似很强大！

主从复制——选项

选项	说明
–only	作用是限定仅复制指定的某个数据库
–slavedelay	为复制设置操作延迟，单位为秒
–fastsync	以主节点的数据快照为基础启动从节点。
–autoresync	当主从节点数据不一致时，是否自动重新同步
–oplogSize	设定主节点中的oplog的容量，单位是MB

副本集

副本集就是具有自动故障恢复功能的主从集群

故障转移集群

副本集——特点

- 与普通主从复制集群相比，具有自动检测机制
- 需要使用–replSet 选项指定副本同伴
- 任何时候，副本集当中最多只允许有1个活跃节点

副本集——活跃节点选举

活跃节点使用心跳跟踪集群中有多少节点对其可见

如果不够半数，活跃节点自动降级为备份节点

新的活跃节点总被假定为系统的最新数据

基于第三条，其他同伴节点的数据都可能会发生回滚

读写分离

将密集的读取操作分流到从节点上，降低主节点的负载

默认情况下，从节点是不允许处理客户端请求的，需要使用–slaveOkay打开

不适用于实时性要求非常高的应用

工作原理——OPLOG

oplog——操作日志

oplog保存在local数据库中，oplog就在其中的oplog.$main集合内保存。该集合的每个文档都记录了主节点上执行的一个操作，其键定义如下：

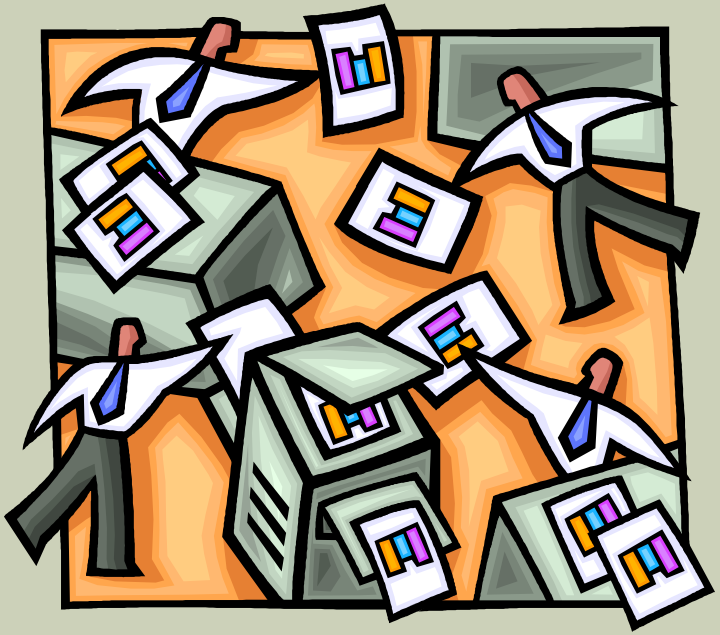
- ts：操作时间戳，占用4字节
- op：操作类型，占用1字节
- ns：操作对象的命名空间（或理解为集合全名）
- o：进一步指定所执行的操作，例如插入

工作原理——同步

从节点首次启动时，做完整同步

主节点数据发生变化时，做增量同步

从节点与主节点数据严重不一致时，做完整同步



复制管理——诊断

db.printReplicationInfo()

在主节点上使用

- 返回信息是oplog的大小以及各种操作的耗时、空间占用等数据

复制管理——诊断

db.printSlaveReplicationInfo()

在从节点上使用

- 返回信息是从节点的数据源列表、同步延迟时间等

复制管理——变更OPLOG容量

变更oplog的容量

在主节点上使用

- 设定—oplogSize参数
- 重启MongoDB

复制管理——复制认证

复制认证

主从节点皆须配置

- 存储在local.system.users
- 优先尝试repl用户
- 主从节点的用户配置必须保持一致

第九天：MONGODB分片

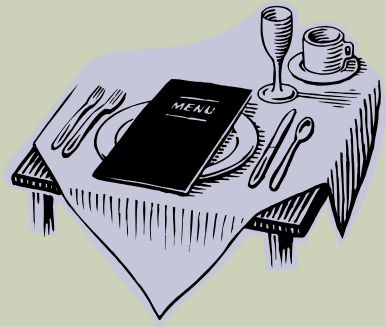
——NewSQL中文社区系列课程

陈彦铭出品

NewSQL中文社区出品
社区：http://newsql.cn/
ＱＱ：705287(ymind)
Ｑ群：79377097(NoSQL三群)

索引

- 分片与自动分片
- 片键与片键选择
- 创建并管理分片



分片与自动分片

分片

- 分片是指将数据拆分，分散到不同的实例上进行负载分流的做法。我们常说的“分表”、“分库”、“分区”等概念都属于分片的实际体现。

自动分片

- 传统分片做法是手工分表、分库。自动分片技术是根据指定的“片键”自动拆分数据并维护数据请求路由的过程。

分片，其实很容易！

片键与片键选择

片键

- 片键就是分片的依据。

好好回想一下

之前自己是根据什么规则来分表、分库的？

片键与片键选择

递增片键

- 连续
- 不均匀
- 写入集中
- 分流较差

随机片键

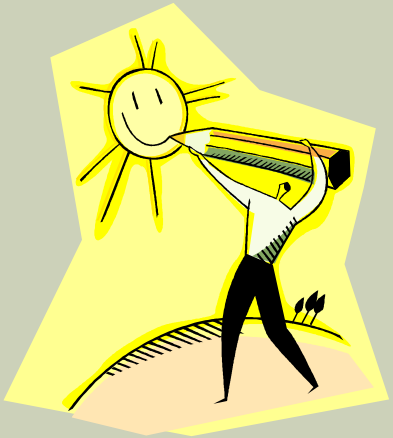
- 不连续
- 均匀
- 写入分散
- 分流较好

老陈：实际应用中应当各自权衡，而不是一味的追求哪个更好！

分片集群的组成部分

三个组成部分

片	mongos	配置服务器
• 保存子集数据的容器	• MongoDB的路由器进程	• 分片集群的配置信息



创建分片

启动配置服务器

- 可以创建一个或多个

添加片

- 每个片都应该是副本集

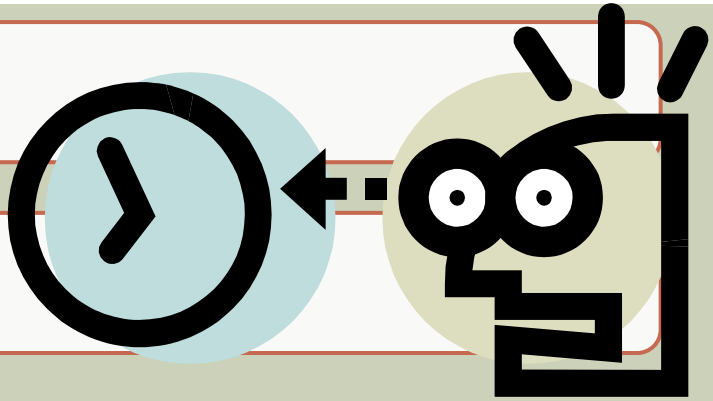
物理服务器

- 性能、安全和稳定性

管理分片

查询分片
- db.shards.find();

数据库
- db.databases.find();



块
- db.chunks.find();

分片状态
- db.printShardingStatus();

删除片
- db.runCommand({ “removeshard” : “ip:port” });

结语

本课的主要内容是让大家理解什么是分片与自动分片，以及自动分片机制的工作原理。

本课没有太多的代码实例，一切皆靠领悟！

如果感觉理解起来很郁闷，可以认真的回想一下在SQL下分表、分库的经验。如果您没有相关的经验，可以向其他高手请教！

第十天：分享实战经验

——NewSQL中文社区系列课程

NewSQL中文社区出品
社区：http://newsql.cn/
ＱＱ：705287(ymind)
Ｑ群：79377097(NoSQL三群)

陈彦铭出品

索引

本课没有目录，想到啥就写啥！

对NOSQL的理解

NoSQL并不是No-SQL，而是指Not Only SQL。

NoSQL的出现是为了弥补SQL数据库因为事务等机制带来的对海量数据、高并发请求的处理的性能上的欠缺。

NoSQL不是为了替代SQL而出现的，它是一种替补方案，而不是解决方案的首选。

绝大多数的NoSQL产品都是基于大内存和高性能随机读写的（比如具有更高性能的固态硬盘阵列），一般的小型企业在选择NoSQL时一定要慎重！不要为了NoSQL而NoSQL，可能会导致花了冤枉钱又耽搁了项目进程。

NoSQL不是万能的，但在大型项目中，你往往需要它！

为什么是MONGODB？

- 基于BSON，兼容JSON
- 有广泛的驱动支持
- 高性能、开源、面向文档
- 全文索引支持
- 自动复制分片
- 内置分布式文件系统
- 内置MapReduce支持
- ……

不要被老陈骗了，我只是想罗列一下MongoDB的优点，如果想了解其他NoSQL产品，不妨看看下面提供的网页。

请阅读：http://cloud.csdn.net/a/20110610/299526.html

文档结构设计

- 在SQL时代，我们可以任意设计自己的表结构，仅仅需要注意数据类型的选择，只是无法直接使用树形设计而已。
- 在MongoDB就没这么幸运了，MongoDB内置的查询机制非常有限，它无法让“你的原来的设计”那么轻松的就能迁移过来。
- 其实，MongoDB这样的“不给力”，都是“故意”的！通过核心算法的约束，强迫开发者大量的使用冗余数据来提升计算效率。
- 这个做法，好！也不好！好的是，的确可以解决很多问题，不好的是，因此我们需要改变之前一贯的思路，甚至是之前认为非常一般般的做法，还会给维护带来很多“看起来不必要”的麻烦。
- MongoDB宣称自己占用的处理器资源是很低的——其实是因为它的架构就直接绕开了很多运算！！！
- 在这里我没有太多想说的，总结起来就一句话——不要吝啬存储空间，要将冗余数据的优点发挥到极致！

索引及查询优化

- 传统的用于优化SQL的技巧，在MongoDB中同样适用；
- 不要为所有键都创建索引，也不要一个索引也不用；
- 如果需要，请优先考虑复合索引，但请注意键的顺序；
- 如果发现即使创建索引也无法很有效的提升，此时应该考虑将数据分片；
- 设计查询时，应当将能够过滤掉大量记录的条件放在前面，除非这样的改动影响了业务需求。
- 无论什么业务，请尽量避免在数据库端处理各种排序，如果可以，请尽量减少这样的设计，以提升数据库服务器的吞吐量。老陈的经验是：用户其实并不讨厌多点几次鼠标，讨厌的是点了N多次鼠标之后，找不到合适的内容。试想一下，为什么baidu不提供用户主动排序的功能？

复制分片及副本集

- 复制是指将数据完完整整的复制到其他MongoDB实例上；
- 分片是将一块数据很多小块并分发到不同的集合或MongoDB实例；
- 副本集带有故障转移的特性
- 而实际上，在生产环境中，我们应当将如上三个概念混合使用，并发挥到极致。
- 数据安全、运算效率以及负载分流、故障转移等等，每一个特性都需要做的很强壮！
- 还记得前面说过的吗，NoSQL不是谁都可以玩的，也不是谁都可以玩好的！一个强壮的MongoDB集群可能需要至少十几台服务器，光硬件就是不小的投入！
- 关于数据安全——不要迷信任何设备，应当经常备份重要的业务数据。
- 是的，使用复制可以对MongoDB的数据执行热备份。而不需要停掉主数据库引擎。

其他

- 尽量少用嵌套文档，它会让你的数据库膨胀的非常快，且不利于扩展；
- 当然，我不是说不要用，如果没有对子文档的深层查询、排序，或者根本就是记录一下而已，这种情况用子文档是最爽不过了！
- MongoDB 不适合存储高精度的数字，比如你需要精度非常高的财务数据存储，此时建议使用其他持久化方案，或者你干脆就存成字符串或者创建一个字符串格式的副本进行存储。
- MongoDB没有内联查询机制，全靠手工外键；
- MongoDB没有自增标识，未来的规划中也没有迹象表明要支持，实际上，这是MongoDB故意的！我们建议使用随机标识，而不是递增标识。
- 标识，读作biao zhi4，而不是biao shi2，记好了哈！

广而告之

NoSQL三群

- 79377097

NewSQL中文社区

- http://newsql.cn/

入驻NewSQL中文社区需要先到群里获取邀请码