

MongoDB 教程



MongoDB 是一个基于分布式文件存储的数据库。由 C++ 语言编写。旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。

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。

[现在开始学习 MongoDB！](#)

内容列表

[NoSQL 简介](#)

介绍NoSQL基础概念。

[MongoDB 简介](#)

介绍MongoDB基础概念。

[window平台安装MongoDB](#)

介绍如何在window平台上安装MongoDB。

[Linux平台安装MongoDB](#)

介绍如何在Linux平台上安装MongoDB。

[MongoDB - 概念解析](#)

[MongoDB 连接](#)

介绍 MongoDB 数据库，对象，集合应用。

[PHP安装MongoDB扩展](#)

介绍PHP安装MongoDB扩展的方法。

[MongoDB 插入文档](#)

介绍MongoDB 数据插入操作。

[MongoDB 更新文档](#)

介绍 MongoDB 更新数据操作。

[MongoDB 删除文档](#)

介绍 MongoDB 删除数据操作。

[MongoDB 查询](#)

介绍 MongoDB 数据查询操作。

[MongoDB条件操作符](#)

介绍MongoDB条件操作符的使用。

[MongoDB \\$type 操作符](#)

介绍 MongoDB 条件操作符\$type的使用。

参考地址

MongoDB 官网地址: <https://www.mongodb.com/>

MongoDB 官方英文文档: <https://docs.mongodb.com/manual/>

MongoDB 各平台下载地址: <https://www.mongodb.com/download-center#community>

NoSQL 简介

NoSQL (NoSQL = Not Only SQL), 意即"不仅仅是SQL"。

在现代的计算系统上每天网络上都会产生庞大的数据量。

这些数据有很大一部分是由关系数据库管理系统（RDBMS）来处理。1970年 E.F.Codd's提出的关系模型的论文 "A relational model of data for large shared data banks", 这使得数据建模和应用程序编程更加简单。

通过应用实践证明，关系模型是非常适合于客户服务器编程，远远超出预期的利益，今天它是结构化数据存储在网络和商务应用的主导技术。

NoSQL 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。

关系型数据库遵循ACID规则

事务在英文中是transaction，和现实世界中的交易很类似，它有如下四个特性：

1、A (Atomicity) 原子性

原子性很容易理解，也就是说事务里的所有操作要么全部做完，要么都不做，事务成功的条件是事务里的所有操作都成功，只要有一个操作失败，整个事务就失败，需要回滚。

比如银行转账，从A账户转100元至B账户，分为两个步骤：1）从A账户取100元；2）存入100元至B账户。这两步要么一起完成，要么一起不完成，如果只完成第一步，第二步失败，钱会莫名其妙少了100元。

2、C (Consistency) 一致性

一致性也比较容易理解，也就是说数据库要一直处于一致的状态，事务的运行不会改变数据库原本的一致性约束。

例如现有完整性约束 $a+b=10$ ，如果一个事务改变了a，那么必须得改变b，使得事务结束后依然满足 $a+b=10$ ，否则事务失败。

3、I (Isolation) 独立性

所谓的独立性是指并发的事务之间不会互相影响，如果一个事务要访问的数据正在被另外一个事务修改，只要另外一个事务未提交，它所访问的数据就不受未提交事务的影响。

比如现在有个交易是从A账户转100元至B账户，在这个交易还未完成的情况下，如果此时B查询自己的账户，是看不到新增加的100元的。

4、D (Durability) 持久性

持久性是指一旦事务提交后，它所做的修改将会永久的保存在数据库上，即使出现宕机也不会丢失。

分布式系统

分布式系统（distributed system）由多台计算机和通信的软件组件通过计算机网络连接（本地网络或广域网）组成。

分布式系统是建立在网络之上的软件系统。正是因为软件的特性，所以分布式系统具有高度的内聚性和透明性。

因此，网络和分布式系统之间的区别更多的在于高层软件（特别是操作系统），而不是硬件。

分布式系统可以应用在不同的平台上如：Pc、工作站、局域网和广域网上等。

分布式计算的优点

可靠性（容错）：

分布式计算系统中的一个重要的优点是可靠性。一台服务器的系统崩溃并不影响到其余的服务器。

可扩展性：

在分布式计算系统可以根据需要增加更多的机器。

资源共享：

共享数据是必不可少的应用，如银行，预订系统。

灵活性：

由于该系统是非常灵活的，它很容易安装，实施和调试新的服务。

更快的速度：

分布式计算系统可以有多个计算机的计算能力，使得它比其他系统有更快的处理速度。

开放系统：

由于它是开放的系统，本地或者远程都可以访问到该服务。

更高的性能：

相较于集中式计算机网络集群可以提供更高的性能（及更好的性价比）。

分布式计算的缺点

故障排除：

故障排除和诊断问题。

软件：

更少的软件支持是分布式计算系统的主要缺点。

网络：

网络基础设施的问题，包括：传输问题，高负载，信息丢失等。

安全性：

开放系统的特性让分布式计算系统存在着数据的安全性和共享的风险等问题。

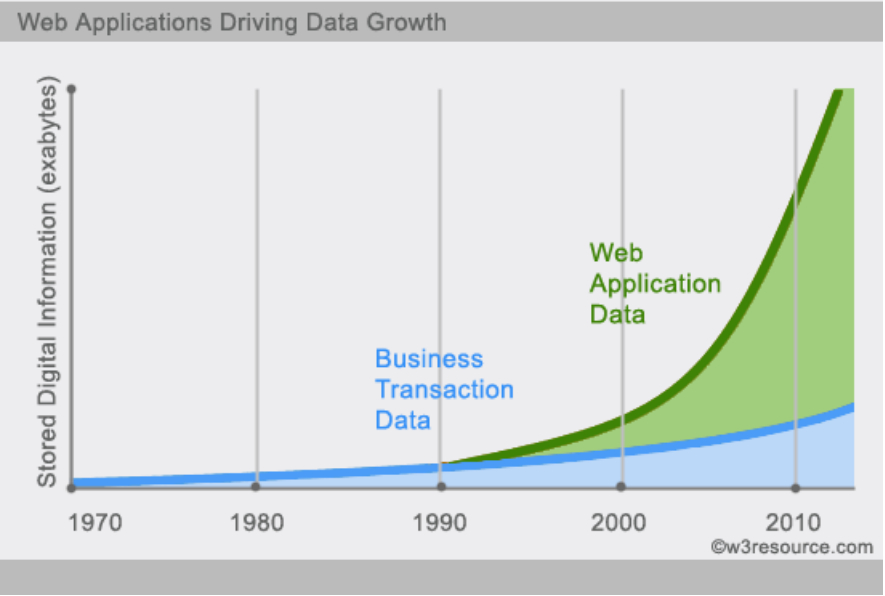
什么是NoSQL？

NoSQL，指的是非关系型的数据库。NoSQL有时也称作Not Only SQL的缩写，是对不同于传统的关系型数据库的数据库管理系统的统称。

NoSQL用于超大规模数据的存储。（例如谷歌或Facebook每天为他们的用户收集万亿比特的数据）。这些类型的数据存储不需要固定的模式，无需多余操作就可以横向扩展。

为什么使用NoSQL？

今天我们可以通过第三方平台（如：Google,Facebook等）可以很容易的访问和抓取数据。用户的个人信息，社交网络，地理位置，用户生成的数据和用户操作日志已经成倍的增加。我们如果要对这些用户数据进行挖掘，那SQL数据库已经不适合这些应用了，NoSQL数据库的发展也却能很好的处理这些大的数据。



实例

社会化关系网：

Each record: UserID1, UserID2
Separate records: UserID, first_name, last_name, age, gender, ...
Task: Find all friends of friends of friends of ... friends of a given user.

Wikipedia 页面：

Large collection of documents
Combination of structured and unstructured data
Task: Retrieve all pages regarding athletics of Summer Olympic before 1950.

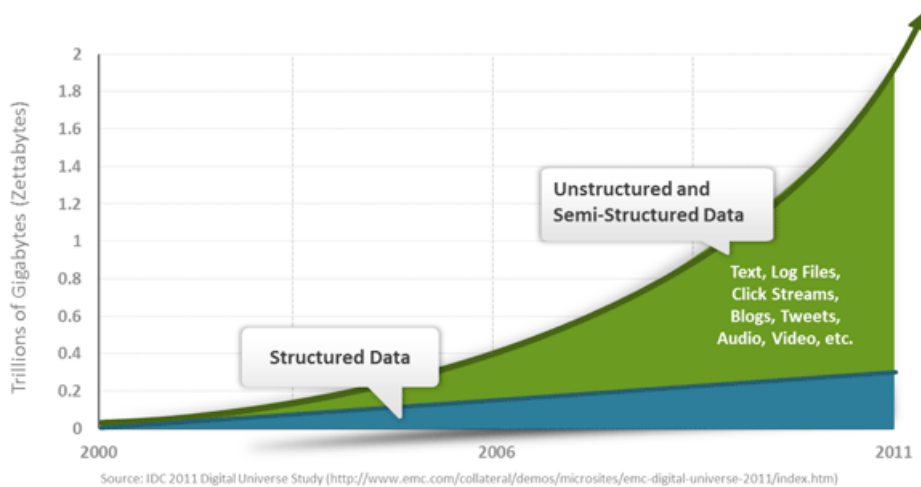
RDBMS vs NoSQL

RDBMS

- 高度组织化结构化数据
- 结构化查询语言（SQL）(SQL)
- 数据和关系都存储在单独的表中。
- 数据操纵语言，数据定义语言
- 严格的一致性
- 基础事务

NoSQL

- 代表着不仅仅是SQL
- 没有声明性查询语言
- 没有预定义的模式
- 键-值对存储，列存储，文档存储，图形数据库
- 最终一致性，而非ACID属性
- 非结构化和不可预知的数据
- CAP定理
- 高性能，高可用性和可伸缩性



NoSQL 简史

NoSQL一词最早出现于1998年，是Carlo Strozzi开发的一个轻量、开源、不提供SQL功能的关系数据库。

2009年，Last.fm的Johan Oskarsson发起了一次关于分布式开源数据库的讨论[2]，来自Rackspace的Eric Evans再次提出了NoSQL的概念，这时的NoSQL主要指非关系型、分布式、不提供ACID的数据库设计模式。

2009年在亚特兰大举行的"no:sql(east)"讨论会是一个里程碑，其口号是"select fun, profit from real_world where relational=false;"。因此，对NoSQL最普遍的解释是"非关联型的"，强调Key-Value Stores和文档数据库的优点，而不是单纯的反对RDBMS。

CAP定理（CAP theorem）

在计算机科学中，CAP定理（CAP theorem），又被称作 布鲁尔定理（Brewer's theorem），它指出对于一个分布式计算系统来说，不可能同时满足以下三点：

- 一致性(Consistency) (所有节点在同一时间具有相同的数据)
- 可用性(Availability) (保证每个请求不管成功或者失败都有响应)

分隔容忍(Partition tolerance) (系统中任意信息的丢失或失败不会影响系统的继续运作)

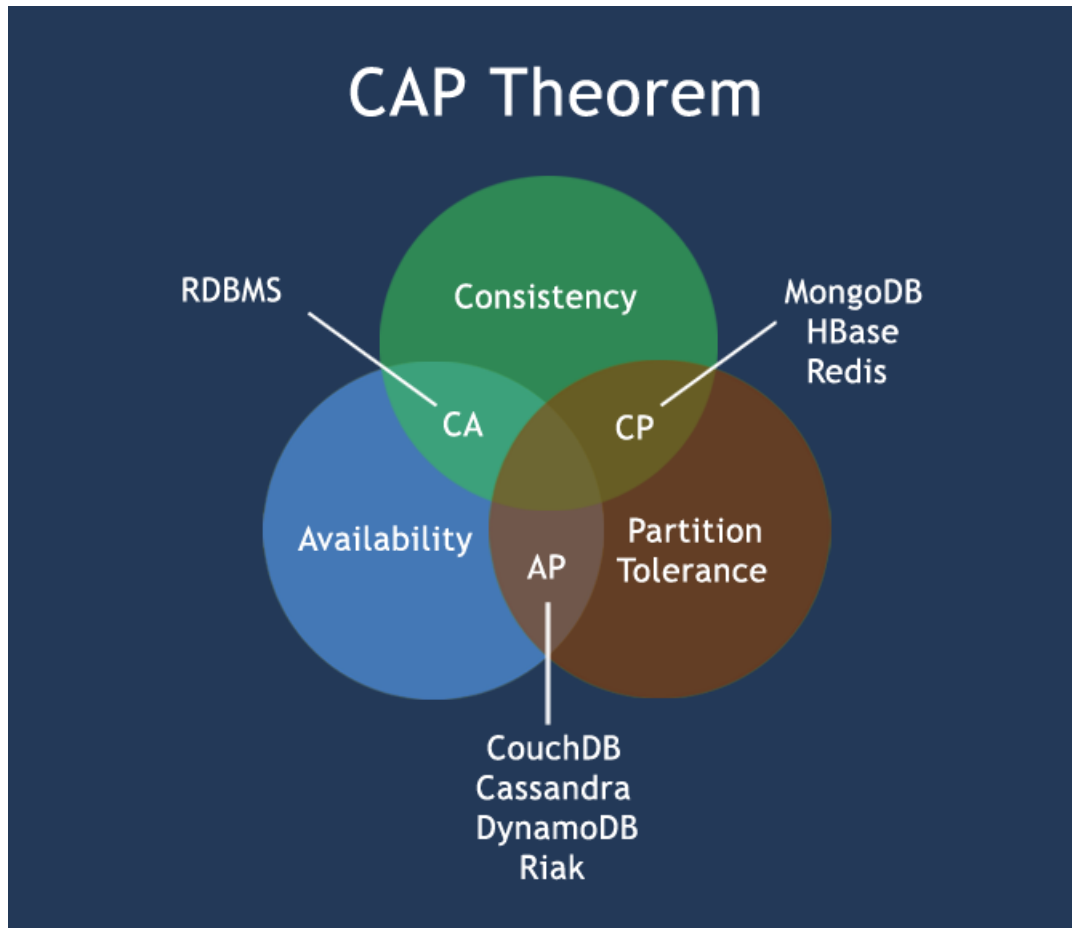
CAP理论的核心是：一个分布式系统不可能同时很好的满足一致性，可用性和分区容错性这三个需求，最多只能同时较好的满足两个。

因此，根据 CAP 原理将 NoSQL 数据库分成了满足 CA 原则、满足 CP 原则和满足 AP 原则三 大类：

CA - 单点集群，满足一致性，可用性的系统，通常在可扩展性上不太强大。

CP - 满足一致性，分区容忍性的系统，通常性能不是特别高。

AP - 满足可用性，分区容忍性的系统，通常可能对一致性要求低一些。



NoSQL的优点/缺点

优点：

- 高可扩展性
- 分布式计算
- 低成本
- 架构的灵活性，半结构化数据
- 没有复杂的关系

缺点：

- 没有标准化
- 有限的查询功能（到目前为止）
- 最终一致是不直观的程序

BASE

BASE: Basically Available, Soft-state, Eventually Consistent。 由 Eric Brewer 定义。

CAP理论的核心是：一个分布式系统不可能同时很好的满足一致性，可用性和分区容错性这三个需求，最多只能同时较好的满足两个。

BASE是NoSQL数据库通常对可用性及一致性的弱要求原则：

Basically Available –基本可用

Soft-state –软状态/柔性事务。 "Soft state" 可以理解为"无连接"的, 而 "Hard state" 是"面向连接"的

Eventual Consistency – 最终一致性， 也是是 ACID 的最终目的。

ACID vs BASE

ACID	BASE
原子性(Atomicity)	基本可用(Basically Available)
一致性(Consistency)	软状态/柔性事务(Soft state)
隔离性(Isolation)	最终一致性 (Eventual consistency)
持久性 (Durable)	

NoSQL 数据库分类

类型	部分代表	特点
列存储	Hbase Cassandra Hypertable	顾名思义，是按列存储数据的。最大的特点是方便存储结构化和半结构化数据，方便做数据压缩，对针对某一系列或者某几列的查询有非常大的IO优势。
文档存储	MongoDB CouchDB	文档存储一般用类似json的格式存储，存储的内容是文档型的。这样也就有机会对某些字段建立索引，实现关系数据库的某些功能。
key-value存储	Tokyo Cabinet / Tyrant Berkeley DB MemcacheDB Redis	可以通过key快速查询到其value。一般来说，存储不管value的格式，照单全收。（Redis包含了其他功能）
图存储	Neo4J FlockDB	图形关系的最佳存储。使用传统关系数据库来解决的话性能低下，而且设计使用不方便。
对象存储	db4o Versant	通过类似面向对象语言的语法操作数据库，通过对象的方式存取数据。
xml数据库	Berkeley DB XML BaseX	高效的存储XML数据，并支持XML的内部查询语法，比如XQuery, Xpath。

谁在使用

现在已经有公司使用了 NoSQL:

Google

Facebook

Mozilla

Adobe

Foursquare

LinkedIn

Digg

McGraw-Hill Education



什么是MongoDB ?

MongoDB 是由C++语言编写的，是一个基于分布式文件存储的开源数据库系统。

在高负载的情况下，添加更多的节点，可以保证服务器性能。

MongoDB 旨在为WEB应用提供可扩展的高性能数据存储解决方案。

MongoDB 将数据存储为一个文档，数据结构由键值(key=>value)对组成。MongoDB 文档类似于 JSON 对象。字段值可以包含其他文档，数组及文档数组。

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```

←

field: value

←

field: value

←

field: value

←

field: value

主要特点

MongoDB 是一个面向文档存储的数据库，操作起来比较简单和容易。

你可以在MongoDB记录中设置任何属性的索引 (如: FirstName="Sameer",Address="8 Gandhi Road")来实现更快的排序。

你可以通过本地或者网络创建数据镜像，这使得MongoDB有更强的扩展性。

如果负载的增加（需要更多的存储空间和更强的处理能力），它可以分布在计算机网络中的其他节点上这就是所谓的分片。

Mongo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。

MongoDb 使用update()命令可以实现替换完成的文档（数据）或者一些指定的数据字段。

Mongoddb中的Map/reduce主要是用来对数据进行批量处理和聚合操作。

Map和Reduce。Map函数调用emit(key,value)遍历集合中所有的记录，将key与value传给Reduce函数进行处理。

Map函数和Reduce函数是使用Javascript编写的，并可以通过db.runCommand或mapreduce命令来执行MapReduce操作。

GridFS是MongoDB中的一个内置功能，可以用于存放大量小文件。

MongoDB允许在服务端执行脚本，可以用Javascript编写某个函数，直接在服务端执行，也可以把函数的定义存储在服务端，下次直接调用即可。

MongoDB支持各种编程语言:RUBY, PYTHON, JAVA, C++, PHP, C#等多种语言。

历史

2007年10月，MongoDB由10gen团队所发展。2009年2月首度推出。

2012年05月23日，MongoDB 2.1 开发分支发布了! 该版本采用全新架构，包含诸多增强。

2012年06月06日，MongoDB 2.0.6 发布，分布式文档数据库。

2013年04月23日，MongoDB 2.4.3 发布，此版本包括了一些性能优化，功能增强以及bug修复。

2013年08月20日，MongoDB 2.4.6 发布。

2013年11月01日，MongoDB 2.4.8 发布。

.....

MongoDB 下载

你可以在mongodb官网下载该安装包，地址为：<https://www.mongodb.com/download-center#community>。MongoDB支持以下平台：

OS X 32-bit

OS X 64-bit

Linux 32-bit

Linux 64-bit

Windows 32-bit

Windows 64-bit

Solaris i86pc

Solaris 64

语言支持

MongoDB有官方的驱动如下：

C

C++

C# / .NET

Erlang

Haskell

Java

JavaScript

Lisp

node.JS

Perl

PHP

Python

Ruby

Scala

MongoDB 工具

有几种可用于MongoDB的管理工具。

监控

MongoDB提供了网络和系统监控工具Munin，它作为一个插件应用于MongoDB中。

Gangila是MongoDB高性能的系统监视的工具，它作为一个插件应用于MongoDB中。

基于图形界面的开源工具 Cacti, 用于查看CPU负载, 网络带宽利用率,它也提供了一个应用于监控 MongoDB 的插件。

GUI

- Fang of Mongo – 网页式,由Django和jQuery所构成。
- Futon4Mongo – 一个CouchDB Futon web的mongodb山寨版。
- Mongo3 – Ruby写成。
- MongoHub – 适用于OSX的应用程序。
- OpriCot – 一个基于浏览器的MongoDB控制台, 由PHP撰写而成。
- Database Master — Windows的mongodb管理工具
- RockMongo — 最好的PHP语言的MongoDB管理工具，轻量级, 支持多国语言。

MongoDB 应用案例

下面列举一些公司MongoDB的实际应用：

- Craigslist上使用MongoDB的存档数十亿条记录。
- FourSquare，基于位置的社交网站，在Amazon EC2的服务器上使用MongoDB分享数据。
- Shutterfly，以互联网为基础的社会和个人出版服务，使用MongoDB的各种持久性数据存储的要求。
- bit.ly, 一个基于Web的网址缩短服务，使用MongoDB的存储自己的数据。
- spike.com，一个MTV网络的联营公司， spike.com使用MongoDB的。
- Intuit公司，一个为小企业 and 个人的软件和服务提供商，为小型企业使用MongoDB的跟踪用户的数据。
- sourceforge.net，资源网站查找，创建和发布开源软件免费，使用MongoDB的后端存储。
- etsy.com，一个购买和出售手工制作物品网站，使用MongoDB。
- 纽约时报，领先的在线新闻门户网站之一，使用MongoDB。
- CERN，著名的粒子物理研究所，欧洲核子研究中心大型强子对撞机的数据使用MongoDB。

❏ NoSQL 简介

Windows 平台安装 MongoDB ❏

❏ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

❏ MongoDB 简介

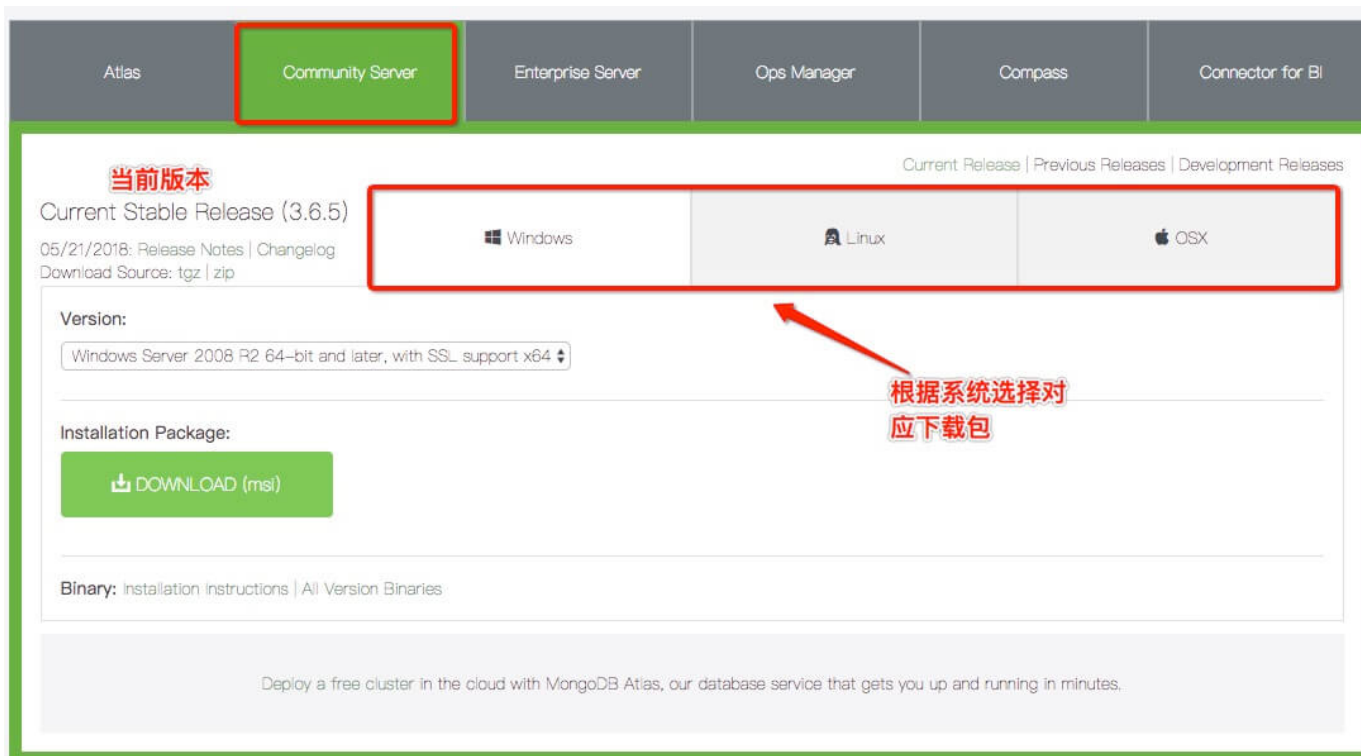
Linux 平台安装 MongoDB ❏

Windows 平台安装 MongoDB

MongoDB 下载

MongoDB 提供了可用于 32 位和 64 位系统的预编译二进制包，你可以从 MongoDB 官网下载安装，MongoDB 预编译二进制包下载地址：<https://www.mongodb.com/download-center#community>

注意：在 MongoDB 2.2 版本后已经不再支持 Windows XP 系统。最新版本也已经没有了 32 位系统的安装文件。



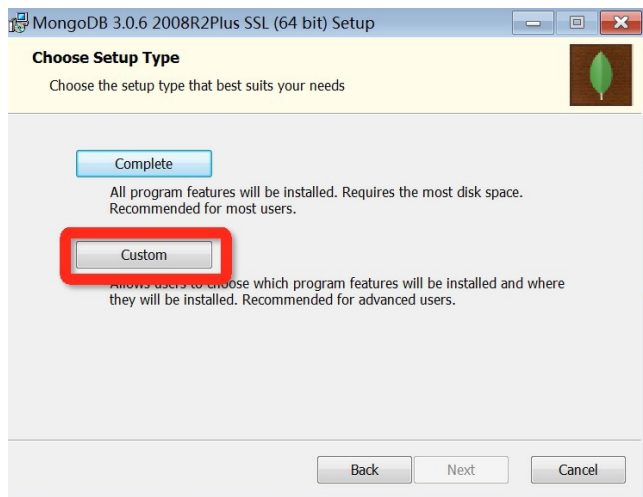
MongoDB for Windows 64-bit 适合 64 位的 Windows Server 2008 R2, Windows 7, 及最新版本的 Window 系统。

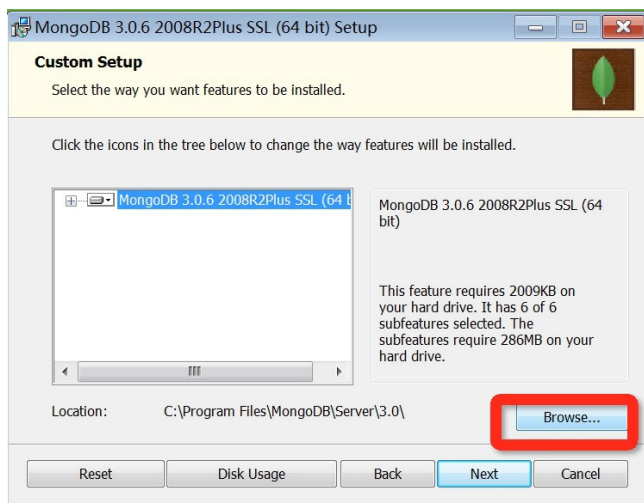
MongoDB for Windows 32-bit 适合 32 位的 Window 系统及最新的 Windows Vista。32 位系统上 MongoDB 的数据库最大为 2GB。

MongoDB for Windows 64-bit Legacy 适合 64 位的 Windows Vista, Windows Server 2003, 及 Windows Server 2008 。

根据你的系统下载 32 位或 64 位的 .msi 文件，下载后双击该文件，按操作提示安装即可。

安装过程中，你可以通过点击 "Custom(自定义)" 按钮来设置你的安装目录。





创建数据目录

MongoDB将数据目录存储在 **db** 目录下。但是这个数据目录不会主动创建，我们在安装完成后需要创建它。请注意，数据目录应该放在根目录下（如：**C:** 或者 **D:** 等）。

在本教程中，我们已经在 **C** 盘安装了 **mongodb**，现在让我们创建一个 **data** 的目录然后在 **data** 目录里创建 **db** 目录。

```
c:\>cd c:\
```

```
c:\>mkdir data
```

```
c:\>cd data
```

```
c:\data>mkdir db
```

```
c:\data>cd db
```

```
c:\data\db>
```

你也可以通过 **window** 的资源管理器中创建这些目录，而不一定通过命令行。

命令行下运行 MongoDB 服务器

为了从命令提示符下运行 **MongoDB** 服务器，你必须从 **MongoDB** 目录的 **bin** 目录中执行 **mongod.exe** 文件。

```
C:\mongodb\bin>mongod --dbpath c:\data\db
```

如果执行成功，会输出如下信息：

```
2015-09-25T15:54:09.212+0800 I CONTROL Hotfix KB2731284 or later update is not
installed, will zero-out data files
```

```
2015-09-25T15:54:09.229+0800 I JOURNAL [initandlisten] journal dir=c:\data\db\journal
journal

2015-09-25T15:54:09.237+0800 I JOURNAL [initandlisten] recover : no journal files present, no recovery needed

2015-09-25T15:54:09.290+0800 I JOURNAL [durability] Durability thread started

2015-09-25T15:54:09.294+0800 I CONTROL [initandlisten] MongoDB starting : pid=2488 port=27017 dbpath=c:\data\db 64-bit host=WIN-1VONBJOCE88

2015-09-25T15:54:09.296+0800 I CONTROL [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2

2015-09-25T15:54:09.298+0800 I CONTROL [initandlisten] db version v3.0.6

.....
```

连接 MongoDB

我们可以在命令窗口中运行 `mongo.exe` 命令即可连接上 MongoDB，执行如下命令：

```
C:\mongodb\bin\mongo.exe
```

配置 MongoDB 服务

管理员模式打开命令行窗口

创建目录，执行下面的语句来创建数据库和日志文件的目录

```
mkdir c:\data\db

mkdir c:\data\log
```

创建配置文件

创建一个配置文件。该文件必须设置 `systemLog.path` 参数，包括一些附加的配置选项更好。

例如，创建一个配置文件位于 `C:\mongodb\mongod.cfg`，其中指定 `systemLog.path` 和 `storage.dbPath`。具体配置内容如下：

```
systemLog:

  destination: file

  path: c:\data\log\mongod.log

storage:

  dbPath: c:\data\db
```

安装 MongoDB 服务

通过执行 `mongod.exe`，使用 `--install` 选项来安装服务，使用 `--config` 选项来指定之前创建的配置文件。

```
C:\mongodb\bin\mongod.exe --config "C:\mongodb\mongod.cfg" --install
```

要使用备用 `dbpath`，可以在配置文件（例如：C:\mongodb\mongod.cfg）或命令行中通过 `--dbpath` 选项指定。

如果需要，您可以安装 `mongod.exe` 或 `mongos.exe` 的多个实例的服务。只需要通过使用 `--serviceName` 和 `--serviceDisplayName` 指定不同的实例名。

。只有当存在足够的系统资源和系统的设计需要这么做。

启动MongoDB服务

```
net start MongoDB
```

关闭MongoDB服务

```
net stop MongoDB
```

移除 MongoDB 服务

```
C:\mongodb\bin\mongod.exe --remove
```

命令行下运行 *MongoDB 服务器* 和 *配置 MongoDB 服务* 任选一个方式启动就可以。

任选一个操作就好

MongoDB 后台管理 Shell

如果你需要进入MongoDB后台管理，你需要先打开mongodb装目录的下的bin目录，然后执行mongo.exe文件，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。

当你进入mongoDB后台后，它默认会链接到 `test` 文档（数据库）：

```
> mongo

MongoDB shell version: 3.0.6

connecting to: test

.....
```

由于它是一个JavaScript shell，您可以运行一些简单的算术运算：

```
> 2 + 2

4

>
```

`db` 命令用于查看当前操作的文档（数据库）：

```
> db

test

>
```

插入一些简单的记录并查找它：

```
> db.runoob.insert({x:10})

WriteResult({ "nInserted" : 1 })

> db.runoob.find()

{ "_id" : ObjectId("5604ff74a274a611b0c990aa"), "x" : 10 }

>
```

第一个命令将数字 10 插入到 runoob 集合的 x 字段中。

[MongoDB 简介](#)

[Linux 平台安装 MongoDB](#)

[点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号: 闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[Windows 平台安装 MongoDB](#)

[MongoDB 概念解析](#)

Linux平台安装MongoDB

MongoDB 提供了 linux 各发行版本 64 位的安装包，你可以在官网下载安装包。

下载地址: <https://www.mongodb.com/download-center#community>

8/24/2015: [Release Notes](#) | [Changelog](#)

Download Source: [tgz](#) | [zip](#)

Windows

Linux

Mac OS X

[Solaris](#)

Select your distribution from the list or the legacy Linux 64-bit version if your distribution is unavailable. Keep in mind that this legacy Linux 64 build may lack the performance optimizations present in targeted builds.

VERSION:

- ✓ Amazon Linux 64-bit
- Debian 7 Linux 64-bit
- RHEL 5 Linux 64-bit
- RHEL 6 Linux 64-bit**
- RHEL 7 Linux 64-bit
- SUSE 11 Linux 64-bit
- Ubuntu 12.04 Linux 64-bit
- Ubuntu 14.04 Linux 64-bit
- Ubuntu 14.10 Clang 64-bit
- Linux 64-bit legacy
- Linux 32-bit legacy

When compiled with SSL enabled and dynamically linked. This requires that SSL libraries be installed seperately. See [here](#) for more

Installing with yum

BINARY: [Installation Instructions](#) | [View Build Archive](#)

下载完安装包，并解压 **tgz**（以下演示的是 64 位 Linux 上的安装）。

```
curl -O https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-3.0.6.tgz # 下载
```

```
tar -zxvf mongodb-linux-x86_64-3.0.6.tgz # 解压

mv mongodb-linux-x86_64-3.0.6/ /usr/local/mongodb # 将解压包拷贝到指定目录
```

MongoDB 的可执行文件位于 **bin** 目录下，所以可以将其添加到 **PATH** 路径中：

```
export PATH=<mongodb-install-directory>/bin:$PATH
```

<mongodb-install-directory> 为你 MongoDB 的安装路径。如本文的 **/usr/local/mongodb**。

创建数据库目录

MongoDB 的数据存储在 **data** 目录的 **db** 目录下，但是这个目录在安装过程不会自动创建，所以你需要手动创建 **data** 目录，并在 **data** 目录中创建 **db** 目录。

以下实例中我们将 **data** 目录创建于根目录下 (**/**)。

注意：**/data/db** 是 MongoDB 默认的启动的数据库路径 (**--dbpath**)。

```
mkdir -p /data/db
```

命令行中运行 MongoDB 服务

你可以再命令行中执行 **mongo** 安装目录中的 **bin** 目录执行 **mongod** 命令来启动 **mongodb** 服务。

*注意：如果你的数据库目录不是 **/data/db**，可以通过 **--dbpath** 来指定。*

```
$ ./mongod

2015-09-25T16:39:50.549+0800 I JOURNAL [initandlisten] journal dir=/data/db/journal

2015-09-25T16:39:50.550+0800 I JOURNAL [initandlisten] recover : no journal files present, no recovery needed

2015-09-25T16:39:50.869+0800 I JOURNAL [initandlisten] preallocateIsFaster=true 3.16

2015-09-25T16:39:51.206+0800 I JOURNAL [initandlisten] preallocateIsFaster=true 3.52

2015-09-25T16:39:52.775+0800 I JOURNAL [initandlisten] preallocateIsFaster=true 7.7
```

MongoDB 后台管理 Shell

如果你需要进入 MongoDB 后台管理，你需要先打开 **mongodb** 装目录的下的 **bin** 目录，然后执行 **mongo** 命令文件。

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。

当你进入 **mongoDB** 后台后，它默认会链接到 **test** 文档（数据库）：

```
$ cd /usr/local/mongodb/bin

$ ./mongo
```

```
MongoDB shell version: 3.0.6

connecting to: test

Welcome to the MongoDB shell.

.....
```

由于它是一个JavaScript shell，您可以运行一些简单的算术运算：

```
> 2+2

4

> 3+6

9
```

现在让我们插入一些简单的数据，并对插入的数据进行检索：

```
> db.runoob.insert({x:10})

WriteResult({ "nInserted" : 1 })

> db.runoob.find()

{ "_id" : ObjectId("5604ff74a274a611b0c990aa"), "x" : 10 }

>
```

第一个命令将数字 10 插入到 runoob 集合的 x 字段中。

MongoDb web 用户界面

MongoDB 提供了简单的 HTTP 用户界面。如果你想启用该功能，需要在启动的时候指定参数 `--rest`。

注意：该功能只适用于 MongoDB 3.2 及之前的早期版本。

```
$ ./mongod --dbpath=/data/db --rest
```

MongoDB 的 Web 界面访问端口比服务的端口多1000。

如果你的MongoDB运行端口使用默认的27017，你可以在端口号为28017访问web用户界面，即地址为：<http://localhost:28017>。

mongod iZ23mtq8bs1Z

[List all commands](#) | [Replica set status](#)

Commands: [features](#) [replSetGetConfig](#) [serverStatus](#) [listIndexes](#) [top](#) [cursorInfo](#) [replSetGetStatus](#) [hostInfo](#) [listDatabases](#)

```
db version v3.0.6
git hash: 1ef45a23a4c5e3480ac919b28afcba3c615488f2
sys info: Linux build6.ny.cbi.10gen.cc 2.6.32-431.3.1.el6.x86_64 #1 SMP Fri Jan 3 21:39:27 UTC 2014 x86_
uptime: 7 seconds
```

overview (only reported if can acquire read lock quickly)

```
time to get readlock: 0ms
# Cursors: 0
replication:
master: 0
slave: 0
```

[Windows 平台安装 MongoDB](#)

[MongoDB 概念解析](#)

[点我分享笔记](#)

[反馈/建议](#)

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号: 闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[PHP7 MongoDB 安装与使用](#)

[MongoDB 创建集合](#)

Mac OSX 平台安装 MongoDB

MongoDB 提供了 OSX 平台上 64 位的安装包，你可以在官网下载安装包。

下载地址: <https://www.mongodb.com/download-center#community>

[Current Release](#) | [Previous Releases](#) | [Development Releases](#)

Current Stable Release (3.4.2)

02/01/2017: [Release Notes](#) | [Changelog](#)
Download Source: [tgz](#) | [zip](#)

Windows

Linux

OSX

Solaris

Version:

OS X 10.7+ 64-bit w/SSL x64

Package Manager:

[Instructions for installing with Homebrew](#)

Binary: [Installation Instructions](#) | [All Version Binaries](#)

[DOWNLOAD \(tgz\)](#)

https://fastdl.mongodb.org/osx/mongodb-osx-ssl-x86_64-3.4.2.tgz

从 MongoDB 3.0 版本开始只支持 OS X 10.7 (Lion) 版本及更新版本的系统。

接下来我们使用 `curl` 命令来下载安装：

```
# 进入 /usr/local

cd /usr/local


# 下载

sudo curl -O https://fastdl.mongodb.org/osx/mongodb-osx-x86_64-3.4.2.tgz


# 解压

sudo tar -zxvf mongodb-osx-x86_64-3.4.2.tgz


# 重命名为 mongodb 目录


sudo mv mongodb-osx-x86_64-3.4.2 mongodb
```

安装完成后，我们可以把 MongoDB 的二进制命令文件目录（安装目录/bin）添加到 PATH 路径中：

```
export PATH=/usr/local/mongodb/bin:$PATH
```

使用 **brew** 安装

此外你还可以使用 OSX 的 `brew` 来安装 `mongodb`：

```
sudo brew install mongodb
```

如果要安装支持 TLS/SSL 命令如下：

```
sudo brew install mongodb --with-openssl
```

安装最新开发版本：

```
sudo brew install mongodb --devel
```

运行 **MongoDB**

1、首先我们创建一个数据库存储目录 `/data/db`：

```
sudo mkdir -p /data/db
```

启动 `mongodb`，默认数据库目录即为 `/data/db`：

```
sudo mongod
```

如果没有创建全局路径 `PATH`，需要进入以下目录

```
cd /usr/local/mongodb/bin
```

```
sudo ./mongod
```

再打开一个终端进入执行以下命令：

```
$ cd /usr/local/mongodb/bin
```

```
$ ./mongo
```

```
MongoDB shell version v3.4.2
```

```
connecting to: mongodb://127.0.0.1:27017
```

```
MongoDB server version: 3.4.2
```

```
Welcome to the MongoDB shell.
```

```
.....
```

```
> 1 + 1
```

```
2
```

```
>
```

注意：如果你的数据库目录不是 `/data/db`，可以通过 `--dbpath` 来指定。

[☐ PHP7 MongoDB 安装与使用](#)

[MongoDB 创建集合 ☐](#)

[☐ 点我分享笔记](#)

[反馈/建议](#)

MongoDB 概念解析

不管我们学习什么数据库都应该学习其中的基础概念，在mongodb中基本的概念是文档、集合、数据库，下面我们挨个介绍。

下表将帮助您更容易理解Mongo中的一些概念：

SQL术语/概念	MongoDB术语/概念	解释/说明
database	database	数据库
table	collection	数据库表/集合
row	document	数据记录行/文档
column	field	数据字段/域
index	index	索引
table joins		表连接,MongoDB不支持
primary key	primary key	主键,MongoDB自动将_id字段设置为主键

通过下图实例，我们也可以更直观的了解Mongo中的一些概念：

id	user_name	email	age	city
1	Mark Hanks	mark@abc.com	25	Los Angeles
2	Richard Peter	richard@abc.com	31	Dallas



```

{
  "_id": ObjectId("5146bb52d8524270060001f3"),
  "age": 25,
  "city": "Los Angeles",
  "email": "mark@abc.com",
  "user_name": "Mark Hanks"
}
{
  "_id": ObjectId("5146bb52d8524270060001f2"),
  "age": 31,
  "city": "Dallas",
  "email": "richard@abc.com",
  "user_name": "Richard Peter"
}

```

数据库

一个mongodb中可以建立多个数据库。

MongoDB的默认数据库为"db"，该数据库存储在data目录中。

MongoDB的单个实例可以容纳多个独立的数据库，每一个都有自己的集合和权限，不同的数据库也放置在不同的文件中。

"show dbs" 命令可以显示所有数据的列表。

```

$ ./mongo

MongoDB shell version: 3.0.6

connecting to: test

> show dbs

local   0.078GB

test    0.078GB

>

```

执行 **"db"** 命令可以显示当前数据库对象或集合。

```
$ ./mongo

MongoDB shell version: 3.0.6

connecting to: test

> db

test

>
```

运行**"use"**命令，可以连接到一个指定的数据库。

```
> use local

switched to db local

> db

local

>
```

以上实例命令中，**"local"** 是你要链接的数据库。

在下一个章节我们将详细讲解MongoDB中命令的使用。

数据库也通过名字来标识。数据库名可以是满足以下条件的任意**UTF-8**字符串。

- 不能是空字符串（""）。

- 不得含有' '（空格）、.、\$、/、\和\0（空字符）。

- 应全部小写。

- 最多64字节。

有一些数据库名是保留的，可以直接访问这些有特殊作用的数据库。

- admin:** 从权限的角度来看，这是**"root"**数据库。要是将一个用户添加到这个数据库，这个用户自动继承所有数据库的权限。一些特定的服务器端命令也只能从这个数据库运行，比如列出所有的数据库或者关闭服务器。

- local:** 这个数据永远不会被复制，可以用来存储限于本地单台服务器的任意集合

- config:** 当Mongo用于分片设置时，**config**数据库在内部使用，用于保存分片的相关信息。

文档

文档是一组键值(key-value)对(即BSON)。MongoDB 的文档不需要设置相同的字段，并且相同的字段不需要相同的数据类型，这与关系型数据库有很大的区别，也是 MongoDB 非常突出的特点。

一个简单的文档例子如下：

```
{"site":"www.runoob.com", "name":"菜鸟教程"}
```

下表列出了 RDBMS 与 MongoDB 对应的术语：

RDBMS	MongoDB
-------	---------

数据库	数据库
表格	集合
行	文档
列	字段
表联合	嵌入文档
主键	主键 (MongoDB 提供了 key 为 _id)
数据库服务和客户端	
Mysqld/Oracle	mongod
mysql/sqlplus	mongo

需要注意的是：

1. 文档中的键/值对是有序的。
2. 文档中的值不仅可以是在双引号里面的字符串，还可以是其他几种数据类型（甚至可以是整个嵌入的文档）。
3. MongoDB区分类型和大小写。
4. MongoDB的文档不能有重复的键。
5. 文档的键是字符串。除了少数例外情况，键可以使用任意UTF-8字符。

文档键命名规范：

- 键不能含有\0 (空字符)。这个字符用来表示键的结尾。
- .和\$有特别的意义，只有在特定环境下才能使用。
- 以下划线"_"开头的键是保留的(不是严格要求的)。

集合

集合就是 MongoDB 文档组，类似于 RDBMS （关系数据库管理系统：Relational Database Management System)中的表格。

集合存在于数据库中，集合没有固定的结构，这意味着你在对集合可以插入不同格式和类型的数据，但通常情况下我们插入集合的数据都会有一定的关联性。

比如，我们可以将以下不同数据结构的文档插入到集合中：

```
{ "site": "www.baidu.com" }

{ "site": "www.google.com", "name": "Google" }

{ "site": "www.runoob.com", "name": "菜鸟教程", "num": 5 }
```

当第一个文档插入时，集合就会被创建。

合法的集合名

- 集合名不能是空字符串""。
- 集合名不能含有\0字符（空字符），这个字符表示集合名的结尾。
- 集合名不能以"system."开头，这是为系统集成保留的前缀。
- 用户创建的集合名字不能含有保留字符。有些驱动程序的确支持在集合名里面包含，这是因为某些系统生成的集合中包含该字符。除非你要访问这种系统创建的集合，否则千万不要在名字里出现\$。

如下实例：

```
db.col.findOne()
```

capped collections

Capped collections 就是固定大小的collection。

它有很高的性能以及队列过期的特性(过期按照插入的顺序). 有点和 "RRD" 概念类似。

Capped collections是高性能自动的维护对象的插入顺序。它非常适合类似记录日志的功能 和标准的collection不同，你必须显式的创建一个capped collection， 指定一个collection的大小，单位是字节。collection的数据存储空间值提前分配的。

要注意的是指定的存储大小包含了数据库的头信息。

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

- 在capped collection中，你能添加新的对象。
- 能进行更新，然而，对象不会增加存储空间。如果增加，更新就会失败 。
- 数据库不允许进行删除。使用drop()方法删除collection所有的行。
- 注意: 删除之后，你必须显式的重新创建这个collection。
- 在32bit机器中， capped collection最大存储为1e9(1X10⁹)个字节。

元数据

数据库的信息是存储在集合中。它们使用了系统的命名空间：

```
dbname.system.*
```

在MongoDB数据库中名字空间 <dbname>.system.* 是包含多种系统信息的特殊集合(Collection)， 如下：

集合命名空间	描述
dbname.system.namespaces	列出所有名字空间。
dbname.system.indexes	列出所有索引。
dbname.system.profile	包含数据库概要(profile)信息。
dbname.system.users	列出所有可访问数据库的用户。
dbname.local.sources	包含复制对端（slave）的服务器信息和状态。

对于修改系统集合中的对象有如下限制。

在{{system.indexes}}插入数据，可以创建索引。但除此之外该表信息是不可变的(特殊的drop index命令将自动更新相关信息)。

{{system.users}}是可修改的。 {{system.profile}}是可删除的。

MongoDB 数据类型

下表为MongoDB中常用的几种数据类型。

数据类型	描述
String	字符串。存储数据常用的数据类型。在 MongoDB 中， UTF-8 编码的字符串才是合法的。
Integer	整型数值。用于存储数值。根据你所采用的服务器，可分为 32 位或 64 位。
Boolean	布尔值。用于存储布尔值（真/假）。
Double	双精度浮点值。用于存储浮点值。

Min/Max keys	将一个值与 BSON（二进制的 JSON）元素的最低值和最高值相对比。
Array	用于将数组或列表或多个值存储为一个键。
Timestamp	时间戳。记录文档修改或添加的具体时间。
Object	用于内嵌文档。
Null	用于创建空值。
Symbol	符号。该数据类型基本上等同于字符串类型，但不同的是，它一般用于采用特殊符号类型的语言。
Date	日期时间。用 UNIX 时间格式来存储当前日期或时间。你可以指定自己的日期时间：创建 Date 对象，传入年月日信息。
Object ID	对象 ID。用于创建文档的 ID。
Binary Data	二进制数据。用于存储二进制数据。
Code	代码类型。用于在文档中存储 JavaScript 代码。
Regular expression	正则表达式类型。用于存储正则表达式。

下面说明下几种重要的数据类型。

ObjectId

ObjectId 类似唯一主键，可以很快的去生成和排序，包含 12 bytes，含义是：

前 4 个字节表示创建 **unix** 时间戳,格林尼治时间 **UTC** 时间，比北京时间晚了 8 个小时

接下来的 3 个字节是机器标识码

紧接的两个字节由进程 id 组成 PID

最后三个字节是随机数



MongoDB 中存储的文档必须有一个 `_id` 键。这个键的值可以是任何类型的，默认是个 `ObjectId` 对象

由于 `ObjectId` 中保存了创建的时间戳，所以你不需要为你的文档保存时间戳字段，你可以通过 `getTimestamp` 函数来获取文档的创建时间：

```
> var newObject = ObjectId()

> newObject.getTimestamp()

ISODate("2017-11-25T07:21:10Z")
```

ObjectId 转为字符串

```
> newObject.str

5a1919e63df83ce79df8b38f
```

字符串

BSON 字符串都是 **UTF-8** 编码。

时间戳

BSON 有一个特殊的时间戳类型用于 **MongoDB** 内部使用，与普通的 `日期` 类型不相关。时间戳值是一个 **64** 位的值。其中：

前32位是一个 `time_t` 值（与Unix新纪元相差的秒数）

后32位是在某秒中操作的一个递增的序数

在单个 **mongod** 实例中，时间戳值通常是唯一的。

在复制集中，**oplog** 有一个 **ts** 字段。这个字段中的值使用**BSON**时间戳表示了操作时间。

***BSON** 时间戳类型主要用于 **MongoDB** 内部使用。在大多数情况下的应用开发中，你可以使用 **BSON** 日期类型。*

日期

表示当前距离 **Unix**新纪元（1970年1月1日）的毫秒数。日期类型是有符号的, 负数表示 1970 年之前的日期。

```
> var mydate1 = new Date()      //格林尼治时间

> mydate1

ISODate("2018-03-04T14:58:51.233Z")

> typeof mydate1

object
```

```
> var mydate2 = ISODate() //格林尼治时间

> mydate2

ISODate("2018-03-04T15:00:45.479Z")

> typeof mydate2

object
```

这样创建的时间是日期类型，可以使用 **JS** 中的 **Date** 类型的方法。

返回一个时间类型的字符串：

```
> var mydate1str = mydate1.toString()

> mydate1str

Sun Mar 04 2018 14:58:51 GMT+0000 (UTC)

> typeof mydate1str

string
```

或者

```
> Date()

Sun Mar 04 2018 15:02:59 GMT+0000 (UTC)
```

[☐ Linux 平台安装 MongoDB](#)

[MongoDB 连接 ☐](#)

[☐ 点我分享笔记](#)

反馈/建议



MongoDB - 连接

在本教程我们将讨论 MongoDB 的不同连接方式。

启动 MongoDB 服务

在前面的教程中, 我们已经讨论了[如何启动 MongoDB 服务](#), 你只需要在 MongoDB 安装目录的 bin 目录下执行 **mongodb** 即可。

执行启动操作后, **mongodb** 在输出一些必要信息后不会输出任何信息, 之后就等待连接的建立, 当连接被建立后, 就会开始打印日志信息。

你可以使用 **MongoDB shell** 来连接 **MongoDB** 服务器。你也可以使用 **PHP** 来连接 **MongoDB**。本教程我们会使用 **MongoDB shell** 来连接 **Mongod** 服务, 之后的章节我们将会介绍如何通过 **php** 来连接 **MongoDB** 服务。

标准 URI 连接语法:

```
mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:portN]]][/[database][?options]]
```

mongodb:// 这是固定的格式, 必须要指定。

username:password@ 可选项, 如果设置, 在连接数据库服务器之后, 驱动都会尝试登陆这个数据库

host1 必须的指定至少一个 **host**, **host1** 是这个 **URI** 唯一要填写的。它指定了要连接服务器的地址。如果要连接复制集, 请指定多个主机地址。

portX 可选的指定端口, 如果不填, 默认为 **27017**

/database 如果指定 **username:password@**, 连接并验证登陆指定数据库。若不指定, 默认打开 **test** 数据库。

?options 是连接选项。如果不使用 **/database**, 则前面需要加上 **/**。所有连接选项都是键值对 **name=value**, 键值对之间通过 **&** 或 **(分号)** 隔开

标准的连接格式包含了多个选项(**options**), 如下所示:

选项	描述
replicaSet=name	验证 replica set 的名称。 Implies connect=replicaSet .
slaveOk=true false	true: 在 connect=direct 模式下, 驱动会连接第一台机器, 即使这台服务器不是主。在 connect=replicaSet 模式下, 驱动会发送所有的写请求到主并且把读取操作分布在其他从服务器。 false: 在 connect=direct 模式下, 驱动会自动找寻主服务器。在 connect=replicaSet 模式下, 驱动仅仅连接主服务器, 并且所有的读写命令都连接到主服务器。
safe=true false	true: 在执行更新操作之后, 驱动都会发送 getLastError 命令来确保更新成功。(还要参考 wtimeoutMS). false: 在每次更新之后, 驱动不会发送 getLastError 来确保更新成功。
w=n	驱动添加 { w : n } 到 getLastError 命令. 应用于 safe=true 。
wtimeoutMS=ms	驱动添加 { wtimeout : ms } 到 getlasterror 命令. 应用于 safe=true 。

<code>fsync=true false</code>	<code>true</code> : 驱动添加 <code>{ fsync : true }</code> 到 <code>getLastError</code> 命令.应用于 <code>safe=true</code> . <code>false</code> : 驱动不会添加到 <code>getLastError</code> 命令中。
<code>journal=true false</code>	如果设置为 <code>true</code> , 同步到 <code>journal</code> (在提交到数据库前写入到实体中). 应用于 <code>safe=true</code>
<code>connectTimeoutMS=ms</code>	可以打开连接的时间。
<code>socketTimeoutMS=ms</code>	发送和接受sockets的时间。

实例

使用默认端口来连接 MongoDB 的服务。

```
mongodb://localhost
```

通过 shell 连接 MongoDB 服务:

```
$ ./mongo

MongoDB shell version: 3.0.6

connecting to: test

...
```

这时候你返回查看运行 `./mongod` 命令的窗口，可以看到是从哪里连接到MongoDB的服务器，您可以看到如下信息：

```
.....省略信息.....

2015-09-25T17:22:27.336+0800 I CONTROL [initandlisten] allocator: tcmalloc

2015-09-25T17:22:27.336+0800 I CONTROL [initandlisten] options: { storage: { dbPath: "/data/db" } }

2015-09-25T17:22:27.350+0800 I NETWORK [initandlisten] waiting for connections on port 27017

2015-09-25T17:22:36.012+0800 I NETWORK [initandlisten] connection accepted from 127.0.0.1:37310 #1 (1 connection now open) # 该行表明一个来自本机的连接

.....省略信息.....
```

MongoDB 连接命令格式

使用用户名和密码连接到 MongoDB 服务器，你必须使用 `'username:password@hostname/dbname'` 格式，'username'为用户名，'password' 为密码。

使用用户名和密码连接登陆到默认数据库：

```
$ ./mongo

MongoDB shell version: 3.0.6

connecting to: test
```

使用用户 **admin** 使用密码 **123456** 连接到本地的 **MongoDB** 服务上。输出结果如下所示：

```
> mongodb://admin:123456@localhost/  
...
```

使用用户名和密码连接登录到指定数据库，格式如下：

```
mongodb://admin:123456@localhost/test
```

更多连接实例

连接本地数据库服务器，端口是默认的。

```
mongodb://localhost
```

使用用户名**fred**，密码**foobar**登录**localhost**的**admin**数据库。

```
mongodb://fred:foobar@localhost
```

使用用户名**fred**，密码**foobar**登录**localhost**的**baz**数据库。

```
mongodb://fred:foobar@localhost/baz
```

连接 **replica pair**，服务器1为**example1.com**服务器2为**example2**。

```
mongodb://example1.com:27017,example2.com:27017
```

连接 **replica set** 三台服务器 (端口 **27017**, **27018**, 和**27019**):

```
mongodb://localhost,localhost:27018,localhost:27019
```

连接 **replica set** 三台服务器，写入操作应用在主服务器 并且分布查询到从服务器。

```
mongodb://host1,host2,host3/?slaveOk=true
```

直接连接第一个服务器，无论是**replica set**一部分或者主服务器或者从服务器。

```
mongodb://host1,host2,host3/?connect=direct;slaveOk=true
```

当你的连接服务器有优先级，还需要列出所有服务器，你可以使用上述连接方式。

安全模式连接到**localhost**:

```
mongodb://localhost/?safe=true
```

以安全模式连接到**replica set**，并且等待至少两个复制服务器成功写入，超时时间设置为**2秒**。

```
mongodb://host1,host2,host3/?safe=true;w=2;wtimeoutMS=2000
```



MongoDB 创建数据库

语法

MongoDB 创建数据库的语法格式如下:

```
use DATABASE_NAME
```

如果数据库不存在, 则创建数据库, 否则切换到指定数据库。

实例

以下实例我们创建了数据库 `runoob`:

```
> use runoob

switched to db runoob

> db

runoob

>
```

如果你想查看所有数据库, 可以使用 **show dbs** 命令:

```
> show dbs

admin    0.000GB

local    0.000GB

>
```

可以看到, 我们刚创建的数据库 `runoob` 并不在数据库的列表中, 要显示它, 我们需要向 `runoob` 数据库插入一些数据。

```
> db.runoob.insert({"name":"菜鸟教程"})
```

```
WriteResult({ "nInserted" : 1 })
```

```
> show dbs
```

```
local    0.078GB
```

```
runoob   0.078GB
```

```
test     0.078GB
```

```
>
```

MongoDB 中默认的数据库为 **test**，如果你没有创建新的数据库，集合将存放在 **test** 数据库中。

注意：在 *MongoDB* 中，集合只有在内容插入后才会创建！就是说，创建集合(数据表)后再插入一个文档(记录)，集合才会真正创建。

[MongoDB 自动增长](#)

[MongoDB 删除数据库](#)

[点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[MongoDB 创建数据库](#)

[PHP7 MongoDB 安装与使用](#)

MongoDB 删除数据库

语法

MongoDB 删除数据库的语法格式如下：

```
db.dropDatabase()
```

删除当前数据库，默认为 **test**，你可以使用 **db** 命令查看当前数据库名。

实例

以下实例我们删除了数据库 **runoob**。

首先，查看所有数据库：

```
> show dbs
```

```
local    0.078GB
```

```
runoob   0.078GB
```

```
test    0.078GB
```

接下来我们切换到数据库 **runoob**:

```
> use runoob

switched to db runoob

>
```

执行删除命令:

```
> db.dropDatabase()

{ "dropped" : "runoob", "ok" : 1 }
```

最后, 我们再通过 **show dbs** 命令数据库是否删除成功:

```
> show dbs

local  0.078GB

test   0.078GB

>
```

删除集合

集合删除语法格式如下:

```
db.collection.drop()
```

以下实例删除了 **runoob** 数据库中的集合 **site**:

```
> use runoob

switched to db runoob

> show tables

site

> db.site.drop()

true

> show tables

>
```



MongoDB 创建集合

本章节我们为大家介绍如何使用 MongoDB 来创建集合。

MongoDB 中使用 **createCollection()** 方法来创建集合。

语法格式：

```
db.createCollection(name, options)
```

参数说明：

name: 要创建的集合名称

options: 可选参数, 指定有关内存大小及索引的选项

options 可以是如下参数：

字段	类型	描述
capped	布尔	(可选) 如果为 true , 则创建固定集合。固定集合是指有着固定大小的集合, 当达到最大值时, 它会自动覆盖最早的文档。 当该值为 true 时, 必须指定 size 参数。
autoIndexId	布尔	(可选) 如为 true , 自动在 _id 字段创建索引。默认为 false 。
size	数值	(可选) 为固定集合指定一个最大值 (以字节计)。 如果 capped 为 true , 也需要指定该字段。
max	数值	(可选) 指定固定集合中包含文档的最大数量。

在插入文档时, MongoDB 首先检查固定集合的 **size** 字段, 然后检查 **max** 字段。

实例

在 **test** 数据库中创建 **runoob** 集合：

```
> use test

switched to db test

> db.createCollection("runoob")

{ "ok" : 1 }
```



```
>
```

如果要查看已有集合，可以使用 `show collections` 命令：

```
> show collections
```

```
runoob
```

```
system.indexes
```

下面是带有几个关键参数的 `createCollection()` 的用法：

创建固定集合 `mycol`，整个集合空间大小 6142800 KB, 文档最大个数为 10000 个。

```
> db.createCollection("mycol", { capped : true, autoIndexId : true, size :
```

```
6142800, max : 10000 } )
```

```
{ "ok" : 1 }
```

```
>
```

在 MongoDB 中，你不需要创建集合。当你插入一些文档时，MongoDB 会自动创建集合。

```
> db.mycol2.insert({"name" : "菜鸟教程"})
```

```
> show collections
```

```
mycol2
```

```
...
```

[Mac OSX 平台安装 MongoDB](#)

[MongoDB 删除集合](#)

[点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1

□

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[MongoDB 创建集合](#)

MongoDB 删除集合

本章节我们为大家介绍如何使用 MongoDB 来删除集合。

MongoDB 中使用 `drop()` 方法来删除集合。

语法格式：

```
db.collection.drop()
```

参数说明：

无

返回值

如果成功删除选定集合，则 `drop()` 方法返回 `true`，否则返回 `false`。

实例

在数据库 `mydb` 中，我们可以先通过 `show collections` 命令查看已存在的集合：

```
>use mydb

switched to db mydb

>show collections

mycol

mycol2

system.indexes

runoob

>
```

接着删除集合 `mycol2`：

```
>db.mycol2.drop()

true

>
```

通过 `show collections` 再次查看数据库 `mydb` 中的集合：

```
>show collections

mycol

system.indexes

runoob

>
```

从结果中可以看出 `mycol2` 集合已被删除。

[☐ MongoDB 创建集合](#)

[☐ 点我分享笔记](#)

反馈/建议

MongoDB 插入文档

本章节中我们将向大家介绍如何将数据插入到MongoDB的集合中。

文档的数据结构和JSON基本一样。

所有存储在集合中的数据都是BSON格式。

BSON是一种类json的一种二进制形式的存储格式,简称Binary JSON。

插入文档

MongoDB 使用 insert() 或 save() 方法向集合中插入文档，语法如下：

```
db.COLLECTION_NAME.insert(document)
```

实例

以下文档可以存储在 MongoDB 的 runoob 数据库 的 col 集合中：

```
>db.col.insert({title: 'MongoDB 教程',  
  
    description: 'MongoDB 是一个 Nosql 数据库',  
  
    by: '菜鸟教程',  
  
    url: 'http://www.runoob.com',  
  
    tags: ['mongodb', 'database', 'NoSQL'],  
  
    likes: 100  
  
})
```

以上实例中 col 是我们的集合名，如果该集合不在该数据库中， MongoDB 会自动创建该集合并插入文档。

查看已插入文档：

```
> db.col.find()  
  
{ "_id" : ObjectId("56064886ade2f21f36b03134"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库"  
  , "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 100 }  
  
>
```

我们也可以将数据定义为一个变量，如下所示：

```
> document=({title: 'MongoDB 教程',
```

```
description: 'MongoDB 是一个 Nosql 数据库',

by: '菜鸟教程',

url: 'http://www.runoob.com',

tags: ['mongodb', 'database', 'NoSQL'],

likes: 100

});
```

执行后显示结果如下：

```
{

  "title" : "MongoDB 教程",

  "description" : "MongoDB 是一个 Nosql 数据库",

  "by" : "菜鸟教程",

  "url" : "http://www.runoob.com",

  "tags" : [

    "mongodb",

    "database",

    "NoSQL"

  ],

  "likes" : 100

}
```

执行插入操作：

```
> db.col.insert(document)

WriteResult({ "nInserted" : 1 })

>
```

插入文档你也可以使用 `db.col.save(document)` 命令。如果不指定 `_id` 字段 `save()` 方法类似于 `insert()` 方法。如果指定 `_id` 字段，则会更新该 `_id` 的数据。



3.2 版本后还有以下几种语法可用于插入文档:
db.collection.insertOne():向指定集合中插入一条文档数据
db.collection.insertMany():向指定集合中插入多条文档数据

插入单条数据

```
> var document = db.collection.insertOne({"a": 3})

> document

{
  "acknowledged" : true,
  "insertedId" : ObjectId("571a218011a82a1d94c02333")
}
```

插入多条数据

```
> var res = db.collection.insertMany([{"b": 3}, {'c': 4}])

> res

{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("571a22a911a82a1d94c02337"),
    ObjectId("571a22a911a82a1d94c02338")
  ]
}
```

二少1年前 (2017-07-12)

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号: 闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[MongoDB 插入文档](#)

[MongoDB 删除文档](#)

MongoDB 更新文档

MongoDB 使用 **update()** 和 **save()** 方法来更新集合中的文档。接下来让我们详细来看下两个函数的应用及其区别。

update() 方法

update() 方法用于更新已存在的文档。语法格式如下：

```
db.collection.update(  
  
    <query>,  
  
    <update>,  
  
    {  
  
        upsert: <boolean>,  
  
        multi: <boolean>,  
  
        writeConcern: <document>  
  
    }  
  
)
```

参数说明：

query：update的查询条件，类似sql update查询内where后面的。

update：update的对象和一些更新的操作符（如\$,\$inc...）等，也可以理解为sql update查询内set后面的

upsert：可选，这个参数的意思是，如果不存在update的记录，是否插入objNew,true为插入，默认是false，不插入。

multi：可选，mongodb 默认是false,只更新找到的第一条记录，如果这个参数为true,就把按条件查出来多条记录全部更新。

writeConcern :可选，抛出异常的级别。

实例

我们在集合 col 中插入如下数据：

```
>db.col.insert({  
  
    title: 'MongoDB 教程',  
  
    description: 'MongoDB 是一个 Nosql 数据库',  
  
    by: '菜鸟教程',  
  
    url: 'http://www.runoob.com',  
  
    tags: ['mongodb', 'database', 'NoSQL'],  
  
    likes: 100  
  
})
```

接着我们通过 update() 方法来更新标题(title):

```
>db.col.update({'title':'MongoDB 教程'},{$set:{'title':'MongoDB'}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })    # 输出信息

> db.col.find().pretty()

{

  "_id" : ObjectId("56064f89ade2f21f36b03136"),

  "title" : "MongoDB",

  "description" : "MongoDB 是一个 Nosql 数据库",

  "by" : "菜鸟教程",

  "url" : "http://www.runoob.com",

  "tags" : [

    "mongodb",

    "database",

    "NoSQL"

  ],

  "likes" : 100

}

>
```

可以看到标题(title)由原来的 "MongoDB 教程" 更新为了 "MongoDB"。

以上语句只会修改第一条发现的文档，如果你要修改多条相同的文档，则需要设置 `multi` 参数为 `true`。

```
>db.col.update({'title':'MongoDB 教程'},{$set:{'title':'MongoDB'}},{multi:true})
```

save() 方法

`save()` 方法通过传入的文档来替换已有文档。语法格式如下：

```
db.collection.save(

  <document>,

  {

    writeConcern: <document>

  }

)
```

参数说明：

document：文档数据。

writeConcern :可选，抛出异常的级别。

实例

以下实例中我们替换了 `_id` 为 `56064f89ade2f21f36b03136` 的文档数据：

```
>db.col.save({

  "_id" : ObjectId("56064f89ade2f21f36b03136"),

  "title" : "MongoDB",

  "description" : "MongoDB 是一个 Nosql 数据库",

  "by" : "Runoob",

  "url" : "http://www.runoob.com",

  "tags" : [

    "mongodb",

    "NoSQL"

  ],

  "likes" : 110

})
```

替换成功后，我们可以通过 `find()` 命令来查看替换后的数据

```
>db.col.find().pretty()

{

  "_id" : ObjectId("56064f89ade2f21f36b03136"),

  "title" : "MongoDB",

  "description" : "MongoDB 是一个 Nosql 数据库",

  "by" : "Runoob",

  "url" : "http://www.runoob.com",

  "tags" : [

    "mongodb",

    "NoSQL"

  ],

  "likes" : 110

}

>
```


更多实例

只更新第一条记录:

```
db.col.update( { "count" : { $gt : 1 } } , { $set : { "test2" : "OK" } } );
```

全部更新:

```
db.col.update( { "count" : { $gt : 3 } } , { $set : { "test2" : "OK" } },false,true );
```

只添加第一条:

```
db.col.update( { "count" : { $gt : 4 } } , { $set : { "test5" : "OK" } },true,false );
```

全部添加加进去:

```
db.col.update( { "count" : { $gt : 5 } } , { $set : { "test5" : "OK" } },true,true );
```

全部更新:

```
db.col.update( { "count" : { $gt : 15 } } , { $inc : { "count" : 1 } },false,true );
```

只更新第一条记录:

```
db.col.update( { "count" : { $gt : 10 } } , { $inc : { "count" : 1 } },false,false );
```

3篇笔记

写笔记

反馈/建议

MongoDB 删除文档

在前面的几个章节中我们已经学习了MongoDB中如何为集合添加数据和更新数据。在本章节中我们将继续学习MongoDB集合的删除。

MongoDB `remove()`函数是用来移除集合中的数据。

MongoDB数据更新可以使用`update()`函数。在执行`remove()`函数前先执行`find()`命令来判断执行的条件是否正确，这是一个比较好的习惯。

语法

`remove()` 方法的基本语法格式如下所示:

```
db.collection.remove(  
  
    <query>,  
  
    <justOne>  
  
)
```

如果你的 MongoDB 是 2.6 版本以后的，语法格式如下:

```
db.collection.remove(  
  
    <query>,  
  
    {  
  
        justOne: <boolean>,  
  
        writeConcern: <document>  
  
    }  
  
)
```

参数说明：

query：（可选）删除的文档的条件。

justOne：（可选）如果设为 **true** 或 **1**，则只删除一个文档。

writeConcern：（可选）抛出异常的级别。

实例

以下文档我们执行两次插入操作：

```
>db.col.insert({title: 'MongoDB 教程',  
  
    description: 'MongoDB 是一个 Nosql 数据库',  
  
    by: '菜鸟教程',  
  
    url: 'http://www.runoob.com',  
  
    tags: ['mongodb', 'database', 'NoSQL'],  
  
    likes: 100  
  
})
```

使用 **find()** 函数查询数据：

```
> db.col.find()  
  
{ "_id" : ObjectId("56066169ade2f21f36b03137"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库",  
  , "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 100 }  
  
{ "_id" : ObjectId("5606616dade2f21f36b03138"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库",  
  , "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 100 }
```

接下来我们移除 **title** 为 'MongoDB 教程' 的文档：

```
>db.col.remove({'title':'MongoDB 教程'})  
  
WriteResult({ "nRemoved" : 2 })          # 删除了两条数据  
  
>db.col.find()
```

.....

没有数据

如果你只想删除第一条找到的记录可以设置 `justOne` 为 `1`，如下所示：

```
>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
```

如果你想删除所有数据，可以使用以下方式（类似常规 `SQL` 的 `truncate` 命令）：

```
>db.col.remove({})

>db.col.find()

>
```

[MongoDB 更新文档](#)

[MongoDB 查询文档](#)



1 篇笔记
#1



[写笔记](#)

`remove()` 方法已经过时了，现在官方推荐使用 `deleteOne()` 和 `deleteMany()` 方法。
如删除集合下全部文档：

```
db.inventory.deleteMany({})
```

删除 `status` 等于 `A` 的全部文档：

```
db.inventory.deleteMany({ status : "A" })
```

删除 `status` 等于 `D` 的一个文档：

```
db.inventory.deleteOne( { status: "D" } )
```

三国电视台1年前
(2017-09-22)

[反馈/建议](#)

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[MongoDB 删除文档](#)

[MongoDB 条件操作符](#)

MongoDB 查询文档

MongoDB 查询文档使用 `find()` 方法。
`find()` 方法以非结构化的方式来显示所有文档。

语法

MongoDB 查询数据的语法格式如下：

```
db.collection.find(query, projection)
```

- query**：可选，使用查询操作符指定查询条件
- projection**：可选，使用投影操作符指定返回的键。查询时返回文档中所有键值， 只需省略该参数即可（默认省略）。

如果你需要以易读的方式来读取数据，可以使用 `pretty()` 方法，语法格式如下：

```
>db.col.find().pretty()
```

`pretty()` 方法以格式化的方式来显示所有文档。

实例

以下实例我们查询了集合 `col` 中的数据：

```
> db.col.find().pretty()

{
  "_id" : ObjectId("56063f17ade2f21f36b03133"),
  "title" : "MongoDB 教程",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb",
    "database",
    "NoSQL"
  ],
  "likes" : 100
}
```

除了 `find()` 方法之外，还有一个 `findOne()` 方法，它只返回一个文档。

MongoDB 与 RDBMS Where 语句比较

如果你熟悉常规的 SQL 数据，通过下表可以更好的理解 MongoDB 的条件语句查询：

操作	格式	范例	RDBMS中的类似语句
等于	{<key>:<value>}	db.col.find({"by":"菜鸟教程"}).pretty()	where by = '菜鸟教程'

操作	格式	范例	RDBMS中的类似语句
小于	{<key>:{\$lt:<value>}}	db.col.find({"likes":{\$lt:50}}).pretty()	where likes < 50
小于或等于	{<key>:{\$lte:<value>}}	db.col.find({"likes":{\$lte:50}}).pretty()	where likes <= 50
大于	{<key>:{\$gt:<value>}}	db.col.find({"likes":{\$gt:50}}).pretty()	where likes > 50
大于或等于	{<key>:{\$gte:<value>}}	db.col.find({"likes":{\$gte:50}}).pretty()	where likes >= 50
不等于	{<key>:{\$ne:<value>}}	db.col.find({"likes":{\$ne:50}}).pretty()	where likes != 50

MongoDB AND 条件

MongoDB 的 find() 方法可以传入多个键(key), 每个键(key)以逗号隔开, 即常规 SQL 的 AND 条件。

语法格式如下:

```
>db.col.find({key1:value1, key2:value2}).pretty()
```

实例

以下实例通过 **by** 和 **title** 键来查询 菜鸟教程 中 MongoDB 教程 的数据

```
> db.col.find({"by":"菜鸟教程", "title":"MongoDB 教程"}).pretty()

{
  "_id" : ObjectId("56063f17ade2f21f36b03133"),
  "title" : "MongoDB 教程",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb",
    "database",
    "NoSQL"
  ],
  "likes" : 100
}
```

以上实例中类似于 WHERE 语句: **WHERE by='菜鸟教程' AND title='MongoDB 教程'**

MongoDB OR 条件

MongoDB OR 条件语句使用了关键字 **\$or**,语法格式如下:

```
>db.col.find(
```

```
{

  $or: [

    {key1: value1}, {key2:value2}

  ]

}

).pretty()
```

实例

以下实例中，我们演示了查询键 **by** 值为 菜鸟教程 或键 **title** 值为 **MongoDB 教程** 的文档。

```
>db.col.find({$or:[{"by":"菜鸟教程"}, {"title": "MongoDB 教程"}]}).pretty()

{

  "_id" : ObjectId("56063f17ade2f21f36b03133"),

  "title" : "MongoDB 教程",

  "description" : "MongoDB 是一个 Nosql 数据库",

  "by" : "菜鸟教程",

  "url" : "http://www.runoob.com",

  "tags" : [

    "mongodb",

    "database",

    "NoSQL"

  ],

  "likes" : 100

}

>
```

AND 和 OR 联合使用

以下实例演示了 AND 和 OR 联合使用，类似常规 SQL 语句为: **'where likes>50 AND (by = '菜鸟教程' OR title = 'MongoDB 教程')**

```
>db.col.find({"likes": {$gt:50}, $or: [{"by": "菜鸟教程"}, {"title": "MongoDB 教程"}]}).pretty()

{

  "_id" : ObjectId("56063f17ade2f21f36b03133"),

  "title" : "MongoDB 教程",

  "description" : "MongoDB 是一个 Nosql 数据库",
```

```
"by" : "菜鸟教程",

"url" : "http://www.runoob.com",

"tags" : [

    "mongodb",

    "database",

    "NoSQL"

],

"likes" : 100

}
```

[MongoDB 删除文档](#)

MongoDB 条件操作符 [MongoDB 条件操作符](#)



2 篇笔记
#2

[写笔记](#)



补充一下 **projection** 参数的使用方法

```
db.collection.find(query, projection)
```

若不指定 **projection**，则默认返回所有键，指定 **projection** 格式如下，有两种模式

```
db.collection.find(query, {title: 1, by: 1}) // inclusion模式 指定返回的键，不返回其他键
```

```
db.collection.find(query, {title: 0, by: 0}) // exclusion模式 指定不返回的键, 返回其他键
```

_id 键默认返回，需要主动指定 **_id:0** 才会隐藏
两种模式不可混用（因为这样的话无法推断其他键是否应返回）

```
db.collection.find(query, {title: 1, by: 0}) // 错误
```

只能全1或全0，除了在**inclusion**模式时可以指定 **_id** 为0

```
db.collection.find(query, {_id:0, title: 1, by: 1}) // 正确
```

DisLido11个月前 (11-09)
#1



若不想指定查询条件参数 **query** 可以用 **{}** 代替，但是需要指定 **projection** 参数：

```
querydb.collection.find({}, {title: 1})
```

pengll3个月前 (07-10)

[反馈/建议](#)



MongoDB 条件操作符

描述

条件操作符用于比较两个表达式并从mongoDB集合中获取数据。

在本章节中,我们将讨论如何在MongoDB中使用条件操作符。

MongoDB中条件操作符有:

(>) 大于 - \$gt

(<) 小于 - \$lt

(>=) 大于等于 - \$gte

(<=) 小于等于 - \$lte

我们使用的数据库名称为"runoob" 我们的集合名称为"col", 以下为我们插入的数据。

为了方便测试,我们可以先使用以下命令清空集合 "col" 的数据:

```
db.col.remove({})
```

插入以下数据

```
>db.col.insert({  
  
  title: 'PHP 教程',  
  
  description: 'PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。',  
  
  by: '菜鸟教程',  
  
  url: 'http://www.runoob.com',  
  
  tags: ['php'],  
  
  likes: 200  
  
})
```

```
>db.col.insert({title: 'Java 教程',  
  
  description: 'Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。',
```



```
by: '菜鸟教程',

url: 'http://www.runoob.com',

tags: ['java'],

likes: 150

})
```

```
>db.col.insert({title: 'MongoDB 教程',

description: 'MongoDB 是一个 Nosql 数据库',

by: '菜鸟教程',

url: 'http://www.runoob.com',

tags: ['mongodb'],

likes: 100

})
```

使用find()命令查看数据:

```
> db.col.find()

{ "_id" : ObjectId("56066542ade2f21f36b0313a"), "title" : "PHP 教程", "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "php" ], "likes" : 200 }

{ "_id" : ObjectId("56066549ade2f21f36b0313b"), "title" : "Java 教程", "description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "java" ], "likes" : 150 }

{ "_id" : ObjectId("5606654fade2f21f36b0313c"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb" ], "likes" : 100 }
```

MongoDB (>) 大于操作符 - \$gt

如果你想获取 "col" 集合中 "likes" 大于 100 的数据，你可以使用以下命令：

```
db.col.find({likes : {$gt : 100}})
```

类似于SQL语句:

```
Select * from col where likes > 100;
```

输出结果:

```
> db.col.find({likes : {$gt : 100}})
```

```
{ "_id" : ObjectId("56066542ade2f21f36b0313a"), "title" : "PHP 教程", "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "php" ], "likes" : 200 }
```

```
{ "_id" : ObjectId("56066549ade2f21f36b0313b"), "title" : "Java 教程", "description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "java" ], "likes" : 150 }
```

```
>
```

MongoDB (>=) 大于等于操作符 - \$gte

如果你想获取"col"集合中 "likes" 大于等于 100 的数据，你可以使用以下命令：

```
db.col.find({likes : {$gte : 100}})
```

类似于SQL语句：

```
Select * from col where likes >=100;
```

输出结果：

```
> db.col.find({likes : {$gte : 100}})
```

```
{ "_id" : ObjectId("56066542ade2f21f36b0313a"), "title" : "PHP 教程", "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "php" ], "likes" : 200 }
```

```
{ "_id" : ObjectId("56066549ade2f21f36b0313b"), "title" : "Java 教程", "description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "java" ], "likes" : 150 }
```

```
{ "_id" : ObjectId("5606654fade2f21f36b0313c"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb" ], "likes" : 100 }
```

```
>
```

MongoDB (<) 小于操作符 - \$lt

如果你想获取"col"集合中 "likes" 小于 150 的数据，你可以使用以下命令：

```
db.col.find({likes : {$lt : 150}})
```

类似于SQL语句：

```
Select * from col where likes < 150;
```

输出结果：

```
> db.col.find({likes : {$lt : 150}})
```

```
{ "_id" : ObjectId("5606654fade2f21f36b0313c"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库",  
  , "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb" ], "likes" : 100 }
```

MongoDB (<=) 小于操作符 - \$lte

如果你想获取"col"集合中 "likes" 小于等于 150 的数据，你可以使用以下命令：

```
db.col.find({likes : {$lte : 150}})
```

类似于SQL语句：

```
Select * from col where likes <= 150;
```

输出结果：

```
> db.col.find({likes : {$lte : 150}})
```

```
{ "_id" : ObjectId("56066549ade2f21f36b0313b"), "title" : "Java 教程", "description" : "Java 是由Sun Microsystems公司于  
1995年5月推出的高级程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "java" ], "likes" : 150  
}
```

```
{ "_id" : ObjectId("5606654fade2f21f36b0313c"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库",  
  , "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb" ], "likes" : 100 }
```

MongoDB 使用 (<) 和 (>) 查询 - \$lt 和 \$gt

如果你想获取"col"集合中 "likes" 大于100，小于 200 的数据，你可以使用以下命令：

```
db.col.find({likes : {$lt : 200, $gt : 100}})
```

类似于SQL语句：

```
Select * from col where likes>100 AND likes<200;
```

输出结果：

```
> db.col.find({likes : {$lt : 200, $gt : 100}})
```

```
{ "_id" : ObjectId("56066549ade2f21f36b0313b"), "title" : "Java 教程", "description" : "Java 是由Sun Microsystems公司于  
1995年5月推出的高级程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "java" ], "likes" : 150  
}
```

```
>
```

☐

2 篇笔记

#2

☐

写笔记

一些简写说明：

\$gt	-----	greater than	>
\$gte	-----	gt equal	>=
\$lt	-----	less than	<
\$lte	-----	lt equal	<=
\$ne	-----	not equal	!=
\$eq	-----	equal	=

Mr.L1年前 (2017-07-12)

#1

☐

模糊查询

查询 title 包含"教"字的文档：

```
db.col.find({title:/教/})
```

查询 title 字段以"教"字开头的文档：

```
db.col.find({title:/^教/})
```

查询 title 字段以"教"字结尾的文档：

```
db.col.find({title:/教$/})
```

最初的梦想2个月前

(08-16)

反馈/建议

MongoDB \$type 操作符

描述

在本章节中，我们将继续讨论MongoDB中条件操作符 \$type。

\$type操作符是基于BSON类型来检索集合中匹配的数据类型，并返回结果。

MongoDB 中可以使用的类型如下表所示：

类型	数字	备注
Double	1	
String	2	
Object	3	
Array	4	
Binary data	5	
Undefined	6	已废弃。
Object id	7	
Boolean	8	
Date	9	
Null	10	
Regular Expression	11	
JavaScript	13	
Symbol	14	
JavaScript (with scope)	15	
32-bit integer	16	
Timestamp	17	
64-bit integer	18	
Min key	255	Query with -1.
Max key	127	

我们使用的数据库名称为"runoob" 我们的集合名称为"col"， 以下为我们插入的数据。

简单的集合"col"：

```
>db.col.insert({

    title: 'PHP 教程',

    description: 'PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。',
```

```
by: '菜鸟教程',

url: 'http://www.runoob.com',

tags: ['php'],

likes: 200

})
```

```
>db.col.insert({title: 'Java 教程',

description: 'Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。',

by: '菜鸟教程',

url: 'http://www.runoob.com',

tags: ['java'],

likes: 150

})
```

```
>db.col.insert({title: 'MongoDB 教程',

description: 'MongoDB 是一个 Nosql 数据库',

by: '菜鸟教程',

url: 'http://www.runoob.com',

tags: ['mongodb'],

likes: 100

})
```

使用find()命令查看数据:

```
> db.col.find()

{ "_id" : ObjectId("56066542ade2f21f36b0313a"), "title" : "PHP 教程", "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "php" ], "likes" : 200 }

{ "_id" : ObjectId("56066549ade2f21f36b0313b"), "title" : "Java 教程", "description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "java" ], "likes" : 150 }

{ "_id" : ObjectId("5606654fade2f21f36b0313c"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb" ], "likes" : 100 }
```

MongoDB 操作符 - \$type 实例

如果想获取 "col" 集合中 title 为 String 的数据，你可以使用以下命令：

```
db.col.find({"title" : {$type : 2}})

或

db.col.find({"title" : {$type : 'string'}})
```

输出结果为：

```
{ "_id" : ObjectId("56066542ade2f21f36b0313a"), "title" : "PHP 教程", "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "php" ], "likes" : 200 }

{ "_id" : ObjectId("56066549ade2f21f36b0313b"), "title" : "Java 教程", "description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "java" ], "likes" : 150 }

{ "_id" : ObjectId("5606654fade2f21f36b0313c"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb" ], "likes" : 100 }
```



MongoDB Limit与Skip方法

MongoDB Limit() 方法

如果你需要在MongoDB中读取指定数量的数据记录，可以使用MongoDB的Limit方法，limit()方法接受一个数字参数，该参数指定从MongoDB中读取的记录条数。

语法

limit()方法基本语法如下所示：

```
>db.COLLECTION_NAME.find().limit(NUMBER)
```

实例

集合 col 中的数据如下：

```
{ "_id" : ObjectId("56066542ade2f21f36b0313a"), "title" : "PHP 教程", "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "php" ], "likes" : 200 }

{ "_id" : ObjectId("56066549ade2f21f36b0313b"), "title" : "Java 教程", "description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "java" ], "likes" : 150 }

{ "_id" : ObjectId("5606654fade2f21f36b0313c"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb" ], "likes" : 100 }
```

以下实例为显示查询文档中的两条记录：

```
> db.col.find({},{"title":1,_id:0}).limit(2)

{ "title" : "PHP 教程" }

{ "title" : "Java 教程" }

>
```

注：如果你们没有指定limit()方法中的参数则显示集合中的所有数据。

MongoDB Skip() 方法

我们除了可以使用limit()方法来读取指定数量的数据外，还可以使用skip()方法来跳过指定数量的数据，skip方法同样接受一个数字参数作为跳过的记录条数。

语法

skip() 方法脚本语法格式如下：

```
>db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)
```

实例

以下实例只会显示第二条文档数据

```
>db.col.find({},{"title":1,_id:0}).limit(1).skip(1)

{ "title" : "Java 教程" }

>
```

注:skip()方法默认参数为 0 。



反馈/建议



MongoDB 排序

MongoDB sort() 方法

在 MongoDB 中使用 `sort()` 方法对数据进行排序，`sort()` 方法可以通过参数指定排序的字段，并使用 `1` 和 `-1` 来指定排序的方式，其中 `1` 为升序排列，而 `-1` 是用于降序排列。

语法

`sort()`方法基本语法如下所示：

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

实例

`col` 集合中的数据如下：

```
{ "_id" : ObjectId("56066542ade2f21f36b0313a"), "title" : "PHP 教程", "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "php" ], "likes" : 200 }

{ "_id" : ObjectId("56066549ade2f21f36b0313b"), "title" : "Java 教程", "description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "java" ], "likes" : 150 }

{ "_id" : ObjectId("5606654fade2f21f36b0313c"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb" ], "likes" : 100 }
```

以下实例演示了 `col` 集合中的数据按字段 `likes` 的降序排列：

```
>db.col.find({},{"title":1,_id:0}).sort({"likes":-1})

{ "title" : "PHP 教程" }

{ "title" : "Java 教程" }

{ "title" : "MongoDB 教程" }

>
```

MongoDB Limit与Skip方法

MongoDB 索引

1 篇笔记

#1

写笔记

skip(), limit(), sort()三个放在一起执行的时候，执行的顺序是先 **sort()**，然后是 **skip()**，最后是显示的 **limit()**。

niuyongjie10个月前 [11-26]

反馈/建议

首页 HTML CSS JS 本地书签

MongoDB 排序

MongoDB 聚合

MongoDB 索引

索引通常能够极大的提高查询的效率，如果没有索引，MongoDB在读取数据时必须扫描集合中的每个文件并选取那些符合查询条件的记录。这种扫描全集合的查询效率是非常低的，特别在处理大量的数据时，查询可以要花费几十秒甚至几分钟，这对网站的性能是非常致命的。索引是特殊的数据结构，索引存储在一个易于遍历读取的数据集合中，索引是对数据库表中一列或多列的值进行排序的一种结构

createIndex() 方法

MongoDB使用 createIndex() 方法来创建索引。

注意在 3.0.0 版本前创建索引方法为 db.collection.ensureIndex()，之后的版本使用了 db.collection.createIndex() 方法，ensureIndex() 还能用，但只是 createIndex() 的别名。

语法

createIndex()方法基本语法格式如下所示：

```
>db.collection.createIndex(keys, options)
```

语法中 Key 值为你要创建的索引字段，1 为指定按升序创建索引，如果你想按降序来创建索引指定为 -1 即可。

实例

```
>db.col.createIndex({"title":1})
>
```

createIndex() 方法中你也可以设置使用多个字段创建索引（关系型数据库中称作复合索引）。

```
>db.col.createIndex({"title":1,"description":-1})
```

>

createIndex() 接收可选参数，可选参数列表如下：

Parameter	Type	Description
background	Boolean	建索引过程会阻塞其它数据库操作，background可指定以后台方式创建索引，即增加 "background" 可选参数。 "background" 默认值为false。
unique	Boolean	建立的索引是否唯一。指定为true创建唯一索引。默认值为false.
name	string	索引的名称。如果未指定，MongoDB的通过连接索引的字段名和排序顺序生成一个索引名称。
dropDups	Boolean	3.0+版本已废弃 。在建立唯一索引时是否删除重复记录,指定 true 创建唯一索引。默认值为 false.
sparse	Boolean	对文档中不存在的字段数据不启用索引；这个参数需要特别注意，如果设置为true的话，在索引字段中不会查询出不包含对应字段的文档.。默认值为 false.
expireAfterSeconds	integer	指定一个以秒为单位的数值，完成 TTL设定，设定集合的生存时间。
v	index version	索引的版本号。默认的索引版本取决于mongod创建索引时运行的版本。
weights	document	索引权重值，数值在 1 到 99,999 之间，表示该索引相对于其他索引字段的得分权重。
default_language	string	对于文本索引，该参数决定了停用词及词干和词器的规则的列表。默认为英语
language_override	string	对于文本索引，该参数指定了包含在文档中的字段名，语言覆盖默认的language，默认值为 language.

实例

在后台创建索引：

```
db.values.createIndex({open: 1, close: 1}, {background: true})
```

通过在创建索引时加 background:true 的选项，让创建工作在后台执行



MongoDB 聚合

MongoDB中聚合(`aggregate`)主要用于处理数据(诸如统计平均值,求和等),并返回计算后的数据结果。有点类似sql语句中的 `count(*)`。

aggregate() 方法

MongoDB中聚合的方法使用`aggregate()`。

语法

`aggregate()` 方法的基本语法格式如下所示:

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

实例

集合中的数据如下:

```
{
  _id: ObjectId('7df78ad8902c')
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
  by_user: 'runoob.com',
  url: 'http://www.runoob.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100
},
{
  _id: ObjectId('7df78ad8902d')
  title: 'NoSQL Overview',
  description: 'No sql database is very fast',
  by_user: 'runoob.com',
  url: 'http://www.runoob.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 10
},
{
  _id: ObjectId('7df78ad8902e')
  title: 'Neo4j Overview',
```

```

description: 'Neo4j is no sql database',

by_user: 'Neo4j',

url: 'http://www.neo4j.com',

tags: ['neo4j', 'database', 'NoSQL'],

likes: 750

},

```

现在我们通过以上集合计算每个作者所写的文章数，使用`aggregate()`计算结果如下：

```

> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}}])

{
  "result" : [
    {
      "_id" : "runoob.com",
      "num_tutorial" : 2
    },
    {
      "_id" : "Neo4j",
      "num_tutorial" : 1
    }
  ],
  "ok" : 1
}
>

```

以上实例类似sql语句：

```
select by_user, count(*) from mycol group by by_user
```

在上面的例子中，我们通过字段 `by_user` 字段对数据进行分组，并计算 `by_user` 字段相同值的总和。

下表展示了一些聚合的表达式：

表达式	描述	实例
<code>\$sum</code>	计算总和。	<code>db.mycol.aggregate([{\$group : {_id : "\$by_user", num_tutorial : {\$sum : "\$likes"}}}}])</code>
<code>\$avg</code>	计算平均值	<code>db.mycol.aggregate([{\$group : {_id : "\$by_user", num_tutorial : {\$avg : "\$likes"}}}}])</code>

\$min	获取集合中所有文档对应值得最小值。	db.mycol.aggregate([{\$group : {_id : "\$by_user", num_tutorial : {\$min : "\$likes"}}}])
\$max	获取集合中所有文档对应值得最大值。	db.mycol.aggregate([{\$group : {_id : "\$by_user", num_tutorial : {\$max : "\$likes"}}}])
\$push	在结果文档中插入值到一个数组中。	db.mycol.aggregate([{\$group : {_id : "\$by_user", url : {\$push: "\$url"}}}])
\$addToSet	在结果文档中插入值到一个数组中，但不创建副本。	db.mycol.aggregate([{\$group : {_id : "\$by_user", url : {\$addToSet : "\$url"}}}])
\$first	根据资源文档的排序获取第一个文档数据。	db.mycol.aggregate([{\$group : {_id : "\$by_user", first_url : {\$first : "\$url"}}}])
\$last	根据资源文档的排序获取最后一个文档数据	db.mycol.aggregate([{\$group : {_id : "\$by_user", last_url : {\$last : "\$url"}}}])

管道的概念

管道在Unix和Linux中一般用于将当前命令的输出结果作为下一个命令的参数。

MongoDB的聚合管道将MongoDB文档在一个管道处理完毕后将结果传递给下一个管道处理。管道操作是可以重复的。

表达式：处理输入文档并输出。表达式是无状态的，只能用于计算当前聚合管道的文档，不能处理其它的文档。

这里我们介绍一下聚合框架中常用的几个操作：

\$project: 修改输入文档的结构。可以用来重命名、增加或删除域，也可以用于创建计算结果以及嵌套文档。

\$match: 用于过滤数据，只输出符合条件的文档。**\$match**使用MongoDB的标准查询操作。

\$limit: 用来限制MongoDB聚合管道返回的文档数。

\$skip: 在聚合管道中跳过指定数量的文档，并返回余下的文档。

\$unwind: 将文档中的某一个数组类型字段拆分成多条，每条包含数组中的一个值。

\$group: 将集合中的文档分组，可用于统计结果。

\$sort: 将输入文档排序后输出。

\$geoNear: 输出接近某一地理位置的有序文档。

管道操作符实例

1、\$project实例

```
db.article.aggregate(  
  
  { $project : {  
  
    title : 1 ,  
  
    author : 1 ,  
  
  } }  
  
);
```

这样的话结果中就只剩下_id,title和author三个字段了，默认情况下_id字段是被包含的，如果要想不包含_id的话可以这样：

```
db.article.aggregate(  
  
  { $project : {  
  
    title : 1 ,  
  
    author : 1 ,  
  
    _id : 0 ,  
  
  } }  
  
);
```

```
{ $project : {  
  
    _id : 0 ,  
  
    title : 1 ,  
  
    author : 1  
  
}});
```

2.\$match实例

```
db.articles.aggregate( [  
  
    { $match : { score : { $gt : 70, $lte : 90 } } },  
  
    { $group: { _id: null, count: { $sum: 1 } } }  
  
] );
```

\$match用于获取分数大于70小于或等于90记录，然后将符合条件的记录送到下一阶段\$group管道操作符进行处理。

3.\$skip实例

```
db.article.aggregate(  
  
    { $skip : 5 }));
```

经过\$skip管道操作符处理后，前五个文档被"过滤"掉。

[MongoDB 索引](#)

[MongoDB 复制\(副本集\)](#)



3 篇笔记

[写笔记](#)

[反馈/建议](#)



[MongoDB 聚合](#)

[MongoDB 分片](#)

MongoDB 复制（副本集）

MongoDB复制是将数据同步在多个服务器的过程。

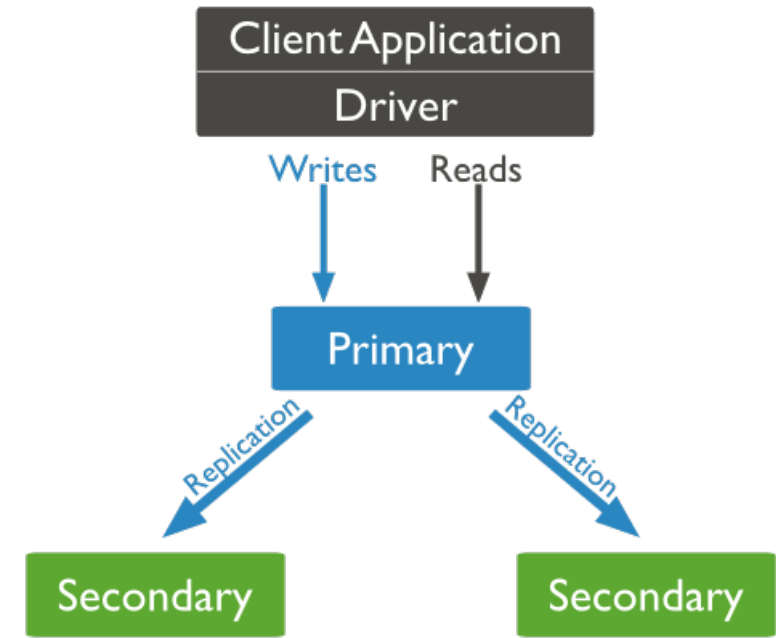
复制提供了数据的冗余备份，并在多个服务器上存储数据副本，提高了数据的可用性， 并可以保证数据的安全性。
复制还允许您从硬件故障和服务中断中恢复数据。

什么是复制？

保障数据的安全性
数据高可用性 (24*7)
灾难恢复
无需停机维护（如备份，重建索引，压缩）
分布式读取数据

MongoDB复制原理

mongodb的复制至少需要两个节点。其中一个是主节点，负责处理客户端请求，其余的都是从节点，负责复制主节点上的数据。
mongodb各个节点常见的搭配方式为：一主一从、一主多从。
主节点记录在其上的所有操作oplog，从节点定期轮询主节点获取这些操作，然后对自己的数据副本执行这些操作，从而保证从节点的数据与主节点一致。
MongoDB复制结构图如下所示：



以上结构图中，客户端从主节点读取数据，在客户端写入数据到主节点时， 主节点与从节点进行数据交互保障数据的一致性。

副本集特征：

N 个节点的集群
任何节点可作为主节点
所有写入操作都在主节点上
自动故障转移
自动恢复

MongoDB副本集设置

在本教程中我们使用同一个MongoDB来做MongoDB主从的实验， 操作步骤如下：

1、关闭正在运行的MongoDB服务器。

现在我们通过指定 `--replSet` 选项来启动mongoDB。`--replSet` 基本语法格式如下：

```
mongod --port "PORT" --dbpath "YOUR_DB_DATA_PATH" --replSet "REPLICA_SET_INSTANCE_NAME"
```


实例

```
mongod --port 27017 --dbpath "D:\set up\mongodb\data" --replSet rs0
```

以上实例会启动一个名为rs0的MongoDB实例，其端口号为27017。

启动后打开命令提示框并连接上mongoDB服务。

在Mongo客户端使用命令rs.initiate()来启动一个新的副本集。

我们可以使用rs.conf()来查看副本集的配置

查看副本集状态使用 rs.status() 命令

副本集添加成员

添加副本集的成员，我们需要使用多台服务器来启动mongo服务。进入Mongo客户端，并使用rs.add()方法来添加副本集的成员。

语法

rs.add() 命令基本语法格式如下：

```
>rs.add(HOST_NAME:PORT)
```

实例

假设你已经启动了一个名为mongod1.net，端口号为27017的Mongo服务。 在客户端命令窗口使用rs.add() 命令将其添加到副本集中，命令如下所示：

```
>rs.add("mongod1.net:27017")

>
```

MongoDB中你只能通过主节点将Mongo服务添加到副本集中， 判断当前运行的Mongo服务是否为主节点可以使用命令db.isMaster() 。

MongoDB的副本集与我们常见的主从有所不同，主从在主机宕机后所有服务将停止，而副本集在主机宕机后，副本会接管主节点成为主节点，不会出现宕机的情况。

☐ MongoDB 聚合

MongoDB 分片 ☐

☐ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

☐ MongoDB 复制(副本集)

MongoDB 备份(mongodump)与恢复(mongorestore) ☐

MongoDB 分片

分片

在MongoDB里面存在另一种集群，就是分片技术,可以满足MongoDB数据量大量增长的需求。

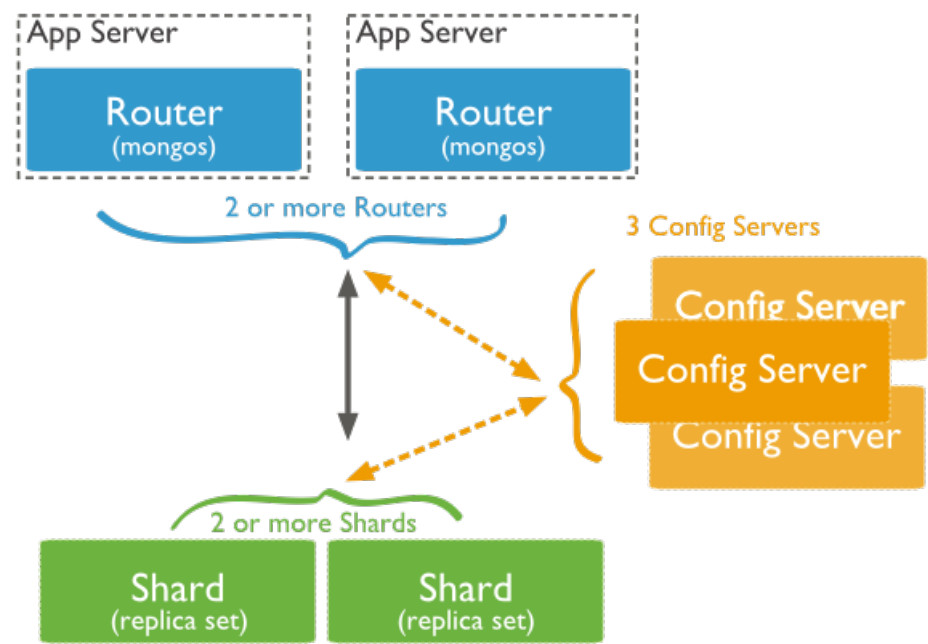
当MongoDB存储海量的数据时，一台机器可能不足以存储数据，也可能不足以提供可接受的读写吞吐量。这时，我们就可以通过在多台机器上分割数据，使得数据库系统能存储和处理更多的数据。

为什么使用分片

- 复制所有的写入操作到主节点
- 延迟的敏感数据会在主节点查询
- 单个副本集限制在12个节点
- 当请求量巨大时会出现内存不足。
- 本地磁盘不足
- 垂直扩展价格昂贵

MongoDB分片

下图展示了在MongoDB中使用分片集群结构分布：



上图中主要有如下所述三个主要组件：

- Shard:**
用于存储实际的数据块，实际生产环境中一个shard server角色可由几台机器组一个replica set承担，防止主机单点故障
- Config Server:**
mongod实例，存储了整个 ClusterMetadata，其中包括 chunk信息。
- Query Routers:**
前端路由，客户端由此接入，且让整个集群看上去像单一数据库，前端应用可以透明使用。

分片实例

分片结构端口分布如下：

Shard Server 1: 27020
Shard Server 2: 27021

Shard Server 3: 27022

Shard Server 4: 27023

Config Server : 27100

Route Process: 40000

步骤一：启动Shard Server

```
[root@100 /]# mkdir -p /www/mongoDB/shard/s0

[root@100 /]# mkdir -p /www/mongoDB/shard/s1

[root@100 /]# mkdir -p /www/mongoDB/shard/s2

[root@100 /]# mkdir -p /www/mongoDB/shard/s3

[root@100 /]# mkdir -p /www/mongoDB/shard/log

[root@100 /]# /usr/local/mongoDB/bin/mongod --port 27020 --dbpath=/www/mongoDB/shard/s0 --logpath=/www/mongoDB/shard/log/s0.log --logappend --fork

....

[root@100 /]# /usr/local/mongoDB/bin/mongod --port 27023 --dbpath=/www/mongoDB/shard/s3 --logpath=/www/mongoDB/shard/log/s3.log --logappend --fork
```

步骤二：启动Config Server

```
[root@100 /]# mkdir -p /www/mongoDB/shard/config

[root@100 /]# /usr/local/mongoDB/bin/mongod --port 27100 --dbpath=/www/mongoDB/shard/config --logpath=/www/mongoDB/shard/log/config.log --logappend --fork
```

注意：这里我们完全可以像启动普通mongodb服务一样启动，不需要添加—shardsvr和configsvr参数。因为这两个参数的作用就是改变启动端口的，所以我们自行指定了端口就可以。

步骤三：启动Route Process

```
/usr/local/mongoDB/bin/mongos --port 40000 --configdb localhost:27100 --fork --logpath=/www/mongoDB/shard/log/route.log --chunkSize 500
```

mongos启动参数中，chunkSize这一项是用来指定chunk的大小的，单位是MB，默认大小为200MB。

步骤四：配置Sharding

接下来，我们使用MongoDB Shell登录到mongos，添加Shard节点

```
[root@100 shard]# /usr/local/mongoDB/bin/mongo admin --port 40000

MongoDB shell version: 2.0.7

connecting to: 127.0.0.1:40000/admin
```

```
mongos> db.runCommand({ addshard:"localhost:27020" })

{ "shardAdded" : "shard0000", "ok" : 1 }

.....

mongos> db.runCommand({ addshard:"localhost:27029" })

{ "shardAdded" : "shard0009", "ok" : 1 }

mongos> db.runCommand({ enablesharding:"test" }) #设置分片存储的数据库

{ "ok" : 1 }

mongos> db.runCommand({ shardcollection: "test.log", key: { id:1,time:1}})

{ "collectionsharded" : "test.log", "ok" : 1 }
```

步骤五： 程序代码内无需太大更改，直接按照连接普通的mongo数据库那样，将数据库连接接入接口40000

☐ MongoDB 复制(副本集)

MongoDB 备份(mongodump)与恢复(mongorestore) ☐



1 篇笔记
#1

☐ 写笔记



1. 创建 Sharding 复制集 rs0

```
# mkdir /data/log

# mkdir /data/db1

# nohup mongod --port 27020 --dbpath=/data/db1 --logpath=/data/log/rs0-1.log --logappend --fork --shardsvr --replSet=rs0 &

# mkdir /data/db2

# nohup mongod --port 27021 --dbpath=/data/db2 --logpath=/data/log/rs0-2.log --logappend --fork --shardsvr --replSet=rs0 &
```

1.1 复制集rs0配置

mongo localhost:27020 > rs.initiate({_id: 'rs0', members: [{_id: 0, host: 'localhost:27020'}, {_id: 1, host: 'localhost:27021'}]}) > rs.isMaster() #查看主从关系

2. 创建 Sharding 复制集 rs1

```
# mkdir /data/db3

# nohup mongod --port 27030 --dbpath=/data/db3 --logpath=/data/log/rs1-1.log --logappend --fork --shardsvr --replSet=rs1 &

# mkdir /data/db4

# nohup mongod --port 27031 --dbpath=/data/db4 --logpath=/data/log/rs1-2.log --logappend --fork --shardsvr --replSet=rs1 &
```

2.1 复制集rs1配置

```
# mongo localhost:27030

> rs.initiate({_id: 'rs1', members: [{_id: 0, host: 'localhost:27030'}, {_id: 1, host: 'localhost:27031'}]})

> rs.isMaster() #查看主从关系
```

3. 创建Config复制集 conf

```
# mkdir /data/conf1

# nohup mongod --port 27100 --dbpath=/data/conf1 --logpath=/data/log/conf-1.log --logappend --fork --configsvr --replSet=conf &

# mkdir /data/conf2

# nohup mongod --port 27101 --dbpath=/data/conf2 --logpath=/data/log/conf-2.log --logappend --fork --configsvr --replSet=conf &
```

3.1 复制集conf配置

```
# mongo localhost:27100

> rs.initiate({_id: 'conf', members: [{_id: 0, host: 'localhost:27100'}, {_id: 1, host: 'localhost:27101'}]})

> rs.isMaster() #查看主从关系
```

4. 创建Route

```
# nohup mongos --port 40000 --configdb conf/localhost:27100,localhost:27101 --fork --logpath=/data/log/route.log --logappend &
```

4.1 设置分片

```
# mongo localhost:40000

> use admin

> db.runCommand({ addshard: 'rs0/localhost:27020,localhost:27021'})

> db.runCommand({ addshard: 'rs1/localhost:27030,localhost:27031'})

> db.runCommand({ enablesharding: 'test'})

> db.runCommand({ shardcollection: 'test.user', key: {name: 1}})
```

chad10个月前 (11-24)

反馈/建议



MongoDB 备份(mongodump)与恢复(mongorestore)

MongoDB数据备份

在Mongoddb中使用mongodump命令来备份MongoDB数据。该命令可以导出所有数据到指定目录中。

mongodump命令可以通过参数指定导出的数据量级转存的服务器。

语法

mongodump命令脚本语法如下：

```
>mongodump -h dbhost -d dbname -o dbdirectory
```

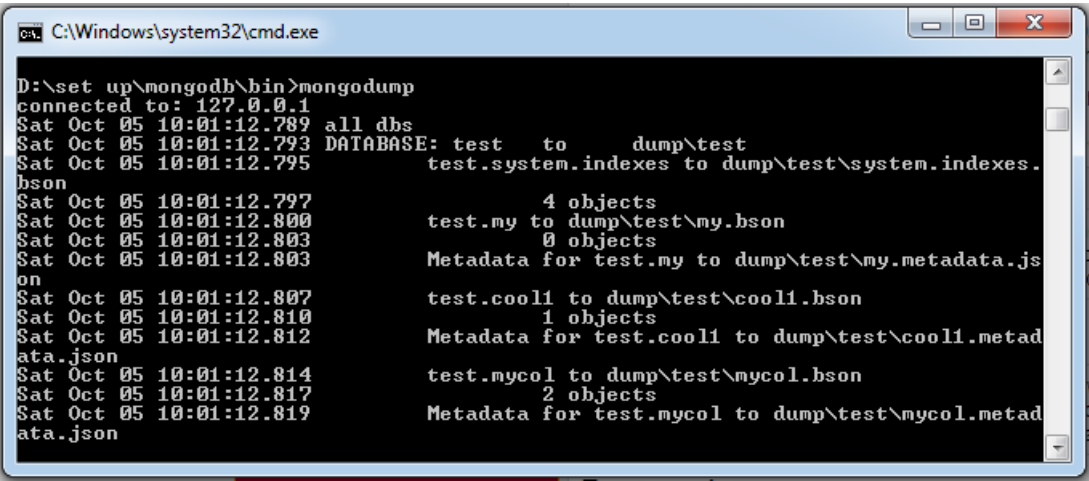
- h: MongoDB所在服务器地址，例如：127.0.0.1，当然也可以指定端口号：127.0.0.1:27017
- d: 需要备份的数据库实例，例如：test
- o: 备份的数据存放位置，例如：c:\data\dump，当然该目录需要提前建立，在备份完成后，系统自动在dump目录下建立一个test目录，这个目录里面存放该数据库实例的备份数据。

实例

在本地使用 27017 启动你的mongod服务。打开命令提示符窗口，进入MongoDB安装目录的bin目录输入命令mongodump:

```
>mongodump
```

执行以上命令后，客户端会连接到ip为 127.0.0.1 端口号为 27017 的MongoDB服务上，并备份所有数据到 bin/dump/ 目录中。命令输出结果如下：



mongodump 命令可选参数列表如下所示：

语法	描述	实例
mongodump --host HOST_NAME --port PORT_NUMBER	该命令将备份所有MongoDB数据	mongodump --host runoob.com --port 27017
mongodump --dbpath DB_PATH --out BACKUP_DIRECTORY		mongodump --dbpath /data/db/ --out /data/backup/
mongodump --collection COLLECTION --db DB_NAME	该命令将备份指定数据库的集合。	mongodump --collection mycol --db test

MongoDB数据恢复

mongodb使用 mongorestore 命令来恢复备份的数据。

语法

mongorestore命令脚本语法如下：

```
>mongorestore -h <hostname>:<port> -d dbname <path>
```

-host <:port>, -h <:port>:

MongoDB所在服务器地址，默认为：localhost:27017

-db, -d:

需要恢复的数据库实例，例如：test，当然这个名称也可以和备份时候的不一样，比如test2

-drop:

恢复的时候，先删除当前数据，然后恢复备份的数据。就是说，恢复后，备份后添加修改的数据都会被删除，慎用哦！

<path>:

mongorestore 最后的一个参数，设置备份数据所在位置，例如：c:\data\dump\test。

你不能同时指定 <path> 和 -dir 选项，-dir也可以设置备份目录。

-dir:

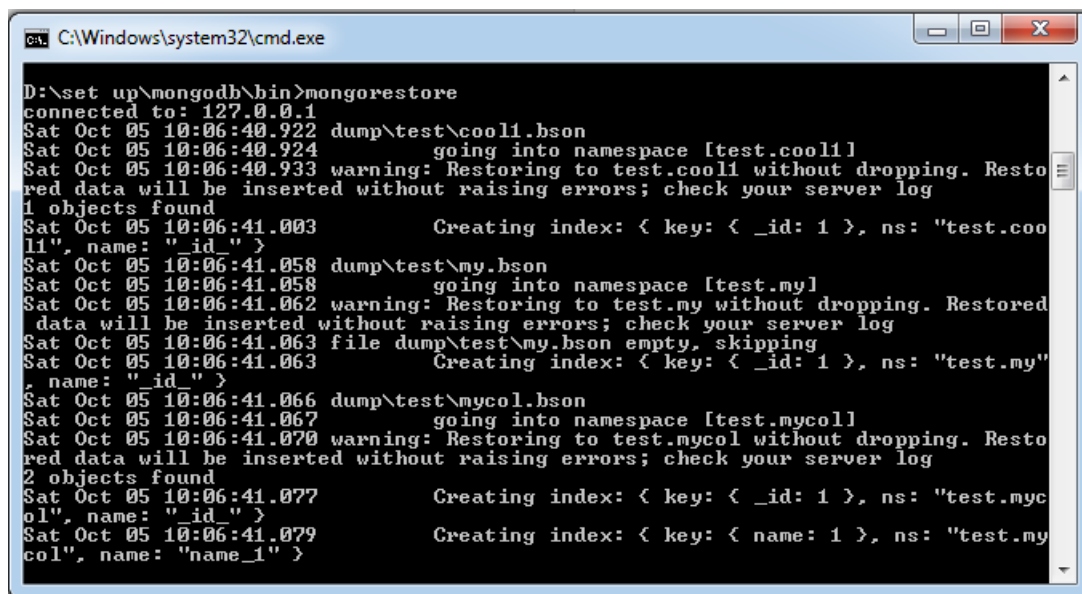
指定备份的目录

你不能同时指定 <path> 和 -dir 选项。

接下来我们执行以下命令：

```
>mongorestore
```

执行以上命令输出结果如下：



```
C:\Windows\system32\cmd.exe
D:\set up\mongodb\bin>mongorestore
connected to: 127.0.0.1
Sat Oct 05 10:06:40.922 dump\test\cool1.bson
Sat Oct 05 10:06:40.924 going into namespace [test.cool1]
Sat Oct 05 10:06:40.933 warning: Restoring to test.cool1 without dropping. Restored data will be inserted without raising errors; check your server log
1 objects found
Sat Oct 05 10:06:41.003 Creating index: { key: { _id: 1 }, ns: "test.cool1", name: "_id" }
Sat Oct 05 10:06:41.058 dump\test\my.bson
Sat Oct 05 10:06:41.058 going into namespace [test.my]
Sat Oct 05 10:06:41.062 warning: Restoring to test.my without dropping. Restored data will be inserted without raising errors; check your server log
Sat Oct 05 10:06:41.063 file dump\test\my.bson empty, skipping
Sat Oct 05 10:06:41.063 Creating index: { key: { _id: 1 }, ns: "test.my", name: "_id" }
Sat Oct 05 10:06:41.066 dump\test\mycol.bson
Sat Oct 05 10:06:41.067 going into namespace [test.mycol]
Sat Oct 05 10:06:41.070 warning: Restoring to test.mycol without dropping. Restored data will be inserted without raising errors; check your server log
2 objects found
Sat Oct 05 10:06:41.077 Creating index: { key: { _id: 1 }, ns: "test.mycol", name: "_id" }
Sat Oct 05 10:06:41.079 Creating index: { key: { name: 1 }, ns: "test.mycol", name: "name_1" }
```

☐ MongoDB 分片

MongoDB 监控 ☐

☐ 点我分享笔记

反馈/建议

MongoDB 监控

在你已经安装部署并允许MongoDB服务后，你必须要了解MongoDB的运行情况，并查看MongoDB的性能。这样在大流量得情况下可以很好的应对并保证MongoDB正常运作。

MongoDB中提供了mongostat 和 mongotop 两个命令来监控MongoDB的运行情况。

mongostat 命令

mongostat是mongodb自带的状态检测工具，在命令行下使用。它会间隔固定时间获取mongodb的当前运行状态，并输出。如果你发现数据库突然变慢或者有其他问题的话，你第一手的操作就考虑采用mongostat来查看mongo的状态。

启动你的Mongo服务，进入到你安装的MongoDB目录下的bin目录， 然后输入mongostat命令，如下所示：

```
D:\set up\mongodb\bin>mongostat
```

以上命令输出结果如下：

```

C:\Windows\system32\cmd.exe - mongostat
local:0.0% 0 0:0 0:0 115b 4k 2 19:20:24
insert query update delete getmore command flushes mapped vsize res faults
locked db idx miss % qriqw ar!aw netIn netOut conn time
local:0.0% 0 0:0 0:0 115b 4k 2 19:20:25
*0 *0 *0 *0 0 0:0 2:0 0 14.1g 28.3g 40m 0
local:0.0% 0 0:0 0:0 115b 4k 2 19:20:26
*0 *0 *0 *0 0 0:0 2:0 0 14.1g 28.3g 40m 0
local:0.0% 0 0:0 0:0 115b 4k 2 19:20:27
*0 *0 *0 *0 0 0:0 2:0 0 14.1g 28.3g 40m 0
local:0.0% 0 0:0 0:0 115b 4k 2 19:20:28
*0 *0 *0 *0 0 0:0 2:0 0 14.1g 28.3g 40m 0
local:0.0% 0 0:0 0:0 115b 4k 2 19:20:29
*0 *0 *0 *0 0 0:0 2:0 0 14.1g 28.3g 40m 0
local:0.0% 0 0:0 0:0 115b 4k 2 19:20:30
*0 *0 *0 *0 0 0:0 2:0 0 14.1g 28.3g 40m 0
local:0.0% 0 0:0 0:0 115b 4k 2 19:20:31
*0 *0 *0 *0 0 0:0 2:0 0 14.1g 28.3g 40m 0
local:0.0% 0 0:0 0:0 115b 4k 2 19:20:32
*0 *0 *0 *0 0 0:0 2:0 0 14.1g 28.3g 40m 0
local:0.0% 0 0:0 0:0 115b 4k 2 19:20:33
*0 *0 *0 *0 0 0:0 2:0 0 14.1g 28.3g 40m 0
local:0.0% 0 0:0 0:0 115b 4k 2 19:20:34
insert query update delete getmore command flushes mapped vsize res faults
locked db idx miss % qriqw ar!aw netIn netOut conn time
local:0.0% 0 0:0 0:0 115b 4k 2 19:20:35
*0 *0 *0 *0 0 0:0 2:0 0 14.1g 28.3g 40m 0
local:0.0% 0 0:0 0:0 115b 4k 2 19:20:36
*0 *0 *0 *0 0 0:0 2:0 0 14.1g 28.3g 40m 0
local:0.0% 0 0:0 0:0 115b 4k 2 19:20:37
*0 *0 *0 *0 0 0:0 2:0 0 14.1g 28.3g 40m 0
local:0.0% 0 0:0 0:0 115b 4k 2 19:20:38
*0 *0 *0 *0 0 0:0 2:0 0 14.1g 28.3g 40m 0
local:0.0% 0 0:0 0:0 115b 4k 2 19:20:39
*0 *0 *0 *0 0 0:0 2:0 0 14.1g 28.3g 40m 0
local:0.0% 0 0:0 0:0 115b 4k 2 19:20:40
*0 *0 *0 *0 0 0:0 2:0 0 14.1g 28.3g 40m 0
local:0.0% 0 0:0 0:0 115b 4k 2 19:20:41

```

mongotop 命令

mongotop也是mongodb下的一个内置工具，mongotop提供了一个方法，用来跟踪一个MongoDB的实例，查看哪些大量的时间花费在读取和写入数据。 mongotop提供每个集合的水平的统计数据。默认情况下，mongotop返回值的每一秒。

启动你的Mongo服务，进入到你安装的MongoDB目录下的bin目录， 然后输入mongotop命令，如下所示：

```
D:\set up\mongodb\bin>mongotop
```

以上命令执行输出结果如下：


```
C:\Windows\system32\cmd.exe - mongotop

      local.system.users      0ms      0ms      0ms
      local.system.replset    0ms      0ms      0ms
      local.startup_log       0ms      0ms      0ms

2013-10-06T13:53:28      ns      total      read      write
      test.system.users      0ms      0ms      0ms
      local.system.users      0ms      0ms      0ms
      local.system.replset    0ms      0ms      0ms
      local.startup_log       0ms      0ms      0ms

2013-10-06T13:53:29      ns      total      read      write
      test.system.users      0ms      0ms      0ms
      local.system.users      0ms      0ms      0ms
      local.system.replset    0ms      0ms      0ms
      local.startup_log       0ms      0ms      0ms

2013-10-06T13:53:30      ns      total      read      write
      test.system.users      0ms      0ms      0ms
      local.system.users      0ms      0ms      0ms
      local.system.replset    0ms      0ms      0ms
      local.startup_log       0ms      0ms      0ms
```

带参数实例

```
E:\mongodb-win32-x86_64-2.2.1\bin>mongotop 10
```

```
E:\mongodb-win32-x86_64-2.2.1\bin>mongotop
connected to: 127.0.0.1

      ns      total      read      write
2013-03-29T04:12:17
      local.system.replset      0ms      0ms      0ms
      local.system.namespaces  0ms      0ms      0ms
      local.system.indexes      0ms      0ms      0ms
      admin.system.indexes      0ms      0ms      0ms
      admin.                      0ms      0ms      0ms
CpsCommodityInfo.system.namespaces  0ms      0ms      0ms
CpsCommodityInfo.system.js      0ms      0ms      0ms

      ns      total      read      write
2013-03-29T04:12:18
      local.system.replset      0ms      0ms      0ms
      local.system.namespaces  0ms      0ms      0ms
      local.system.indexes      0ms      0ms      0ms
      admin.system.indexes      0ms      0ms      0ms
      admin.                      0ms      0ms      0ms
CpsCommodityInfo.system.namespaces  0ms      0ms      0ms
CpsCommodityInfo.system.js      0ms      0ms      0ms
```

后面的10是<sleeptime>参数，可以不使用，等待的时间长度，以秒为单位，mongotop等待调用之间。通过的默认mongotop返回数据的每一秒。

```
E:\mongodb-win32-x86_64-2.2.1\bin>mongotop --locks
```

报告每个数据库的锁的使用中，使用mongotop - 锁，这将产生以下输出：

```
E:\mongodb-win32-x86_64-2.2.1\bin>mongotop --locks
connected to: 127.0.0.1

      db      total      read      write
2013-03-29T04:21:59
      local      0ms      0ms      0ms
      admin      0ms      0ms      0ms
      CpsCommodityInfo      0ms      0ms      0ms
      .      0ms      0ms      0ms

      db      total      read      write
2013-03-29T04:22:00
      local      0ms      0ms      0ms
      admin      0ms      0ms      0ms
      CpsCommodityInfo      0ms      0ms      0ms
      .      0ms      0ms      0ms

      db      total      read      write
2013-03-29T04:22:01
```

输出结果字段说明：

ns:

包含数据库命名空间，后者结合了数据库名称和集合。

db:

包含数据库的名称。名为 . 的数据库针对全局锁定，而非特定数据库。

total:

mongod花费的时间工作在这个命名空间提供总额。

read:

提供了大量的时间，这mongod花费在执行读操作，在此命名空间。

write:

提供这个命名空间进行写操作，这mongod花了大量的时间。



MongoDB Java

环境配置

在 **Java** 程序中如果要使用 **MongoDB**，你需要确保已经安装了 **Java** 环境及 **MongoDB JDBC** 驱动。

本章节实例时候 **Mongo 3.x** 以上版本。

你可以参考本站的[Java教程](#)来安装**Java**程序。现在让我们来检测你是否安装了 **MongoDB JDBC** 驱动。

首先你必须下载mongo jar包，下载地址：<http://mongodb.github.io/mongo-java-driver/>，请确保下载最新版本。

DOWNLOAD **mongo-java-driver** 3.2.2 Maven

```
<dependencies>
  <dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongo-java-driver</artifactId>
    <version>3.2.2</version>
  </dependency>
</dependencies>
```

An uber jar containing the bson library, the core library and the mongodb-driver.
This artifact is a valid OSGi bundle.

你需要将 `mongo-java-driver-3.2.2.jar`（找到合适的版本）包含在你的 `classpath` 中。。

国内 `mongodb-driver jar` 下载地址：<http://central.maven.org/maven2/org/mongodb/mongo-java-driver/>

连接数据库

连接数据库，你需要指定数据库名称，如果指定的数据库不存在，`mongo`会自动创建数据库。

连接数据库的Java代码如下：

```
import com.mongodb.MongoClient;

import com.mongodb.client.MongoDatabase;

public class MongoDBJDBC{

    public static void main( String args[] ){

        try{

            // 连接到 mongodb 服务

            MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

            // 连接到数据库

            MongoDatabase mongoDatabase = mongoClient.getDatabase("mycol");

            System.out.println("Connect to database successfully");

        }catch(Exception e){

            System.err.println( e.getClass().getName() + ": " + e.getMessage() );

        }

    }

}
```

现在，让我们来编译运行程序并连接到数据库 mycol。

你可以根据你的实际环境改变 MongoDB JDBC 驱动的路径。

本实例将 MongoDB JDBC 启动包 mongo-java-driver-3.2.2.jar 放在本地目录下：

```
$ javac -cp .:mongo-java-driver-3.2.2.jar MongoDBJDBC.java

$ java -cp .:mongo-java-driver-3.2.2.jar MongoDBJDBC

Connect to database successfully

Authentication: true
```

本实例中 Mongo 数据库无需用户名密码验证。如果你的 Mongo 需要验证用户名及密码，可以使用以下代码：

```
import java.util.ArrayList;

import java.util.List;

import com.mongodb.MongoClient;

import com.mongodb.MongoCredential;

import com.mongodb.ServerAddress;

import com.mongodb.client.MongoDatabase;

public class MongoDBJDBC {

    public static void main(String[] args){

        try {

            //连接到MongoDB服务 如果是远程连接可以替换“localhost”为服务器所在IP地址

            //ServerAddress()两个参数分别为 服务器地址 和 端口

            ServerAddress serverAddress = new ServerAddress("localhost",27017);

            List<ServerAddress> addrs = new ArrayList<ServerAddress>();

            addrs.add(serverAddress);

            //MongoCredential.createScramSha1Credential()三个参数分别为 用户名 数据库名称 密码

            MongoCredential credential = MongoCredential.createScramSha1Credential("username", "databaseName", "password".toCharArray());

            List<MongoCredential> credentials = new ArrayList<MongoCredential>();

            credentials.add(credential);

            //通过连接认证获取MongoDB连接
```

```

        MongoClient mongoClient = new MongoClient(addr, credentials);

        //连接到数据库

        MongoDBDatabase mongoDatabase = mongoClient.getDatabase("databaseName");

        System.out.println("Connect to database successfully");

    } catch (Exception e) {

        System.err.println( e.getClass().getName() + ": " + e.getMessage() );

    }

}

}

```

创建集合

我们可以使用 `com.mongodb.client.MongoDatabase` 类中的`createCollection()`来创建集合

代码片段如下：

```

import com.mongodb.MongoClient;

import com.mongodb.client.MongoDatabase;

public class MongoDBJDBC{

    public static void main( String args[] ){

        try{

            // 连接到 mongodb 服务

            MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

            // 连接到数据库

            MongoDBDatabase mongoDatabase = mongoClient.getDatabase("mycol");

            System.out.println("Connect to database successfully");

            mongoDatabase.createCollection("test");

            System.out.println("集合创建成功");

        }catch(Exception e){

            System.err.println( e.getClass().getName() + ": " + e.getMessage() );

        }

    }

}

```

```
}  
  
}  
  
}
```

编译运行以上程序，输出结果如下：

```
Connect to database successfully
```

```
集合创建成功
```

获取集合

我们可以使用`com.mongodb.client.MongoDatabase`类的 `getCollection()` 方法来获取一个集合

代码片段如下：

```
import org.bson.Document;  
  
import com.mongodb.MongoClient;  
  
import com.mongodb.client.MongoCollection;  
  
import com.mongodb.client.MongoDatabase;  
  
  
public class MongoDBJDBC{  
  
    public static void main( String args[] ){  
  
        try{  
  
            // 连接到 mongodb 服务  
  
            MongoClient mongoClient = new MongoClient( "localhost" , 27017 );  
  
  
            // 连接到数据库  
  
            MongoDatabase mongoDatabase = mongoClient.getDatabase("mycol");  
  
            System.out.println("Connect to database successfully");  
  
  
  
            MongoCollection<Document> collection = mongoDatabase.getCollection("test");  
  
            System.out.println("集合 test 选择成功");  
  
        }catch(Exception e){  
  
            System.err.println( e.getClass().getName() + ": " + e.getMessage() );  
  
        }  
  
    }  
  
}
```

```
}
```

编译运行以上程序，输出结果如下：

```
Connect to database successfully
```

```
集合 test 选择成功
```

插入文档

我们可以使用`com.mongodb.client.MongoCollection`类的 `insertMany()` 方法来插入一个文档

代码片段如下：

```
import java.util.ArrayList;

import java.util.List;

import org.bson.Document;

import com.mongodb.MongoClient;

import com.mongodb.client.MongoCollection;

import com.mongodb.client.MongoDatabase;

public class MongoDBJDBC{

    public static void main( String args[] ){

        try{

            // 连接到 mongodb 服务

            MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

            // 连接到数据库

            MongoDatabase mongoDatabase = mongoClient.getDatabase("mycol");

            System.out.println("Connect to database successfully");

            MongoCollection<Document> collection = mongoDatabase.getCollection("test");

            System.out.println("集合 test 选择成功");

            //插入文档

            /**

            * 1. 创建文档 org.bson.Document 参数为key-value的格式
```

```

* 2. 创建文档集合List<Document>

* 3. 将文档集合插入数据库集合中 mongoCollection.insertMany(List<Document>) 插入单个文档可以用 mongoCollection.insertOne(Document)

* */

Document document = new Document("title", "MongoDB").
append("description", "database").
append("likes", 100).
append("by", "Fly");

List<Document> documents = new ArrayList<Document>();

documents.add(document);

collection.insertMany(documents);

System.out.println("文档插入成功");

}catch(Exception e){

    System.err.println( e.getClass().getName() + ": " + e.getMessage() );

}

}

}

```

编译运行以上程序，输出结果如下：

```
Connect to database successfully
```

```
集合 test 选择成功
```

```
文档插入成功
```

检索所有文档

我们可以使用 `com.mongodb.client.MongoCollection` 类中的 `find()` 方法来获取集合中的所有文档。

此方法返回一个游标，所以你需要遍历这个游标。

代码片段如下：

```

import org.bson.Document;

import com.mongodb.MongoClient;

import com.mongodb.client.FindIterable;

import com.mongodb.client.MongoCollection;

import com.mongodb.client.MongoCursor;

```



```

import com.mongodb.client.MongoDatabase;

public class MongoDBJDBC{

    public static void main( String args[] ){

        try{

            // 连接到 mongodb 服务

            MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

            // 连接到数据库

            MongoDatabase mongoDatabase = mongoClient.getDatabase("mycol");

            System.out.println("Connect to database successfully");

            MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

            MongoDatabase mongoDatabase = mongoClient.getDatabase("mycol");

            MongoCollection<Document> collection = mongoDatabase.getCollection("test");

            System.out.println("集合 test 选择成功");

            //检索所有文档

            /**

            * 1. 获取迭代器FindIterable<Document>

            * 2. 获取游标MongoCursor<Document>

            * 3. 通过游标遍历检索出的文档集合

            * */

            FindIterable<Document> findIterable = collection.find();

            MongoCursor<Document> mongoCursor = findIterable.iterator();

            while(mongoCursor.hasNext()){

                System.out.println(mongoCursor.next());

            }

        }catch(Exception e){

            System.err.println( e.getClass().getName() + ": " + e.getMessage() );

        }

    }

}

```

编译运行以上程序，输出结果如下：

```
Connect to database successfully
```

```
集合 test 选择成功
```

```
Document({_id=56e65fb1fd57a86304fe2692, title=MongoDB, description=database, likes=100, by=Fly}}
```

更新文档

你可以使用 `com.mongodb.client.MongoCollection` 类中的 `updateMany()` 方法来更新集合中的文档。

代码片段如下：

```
import org.bson.Document;

import com.mongodb.MongoClient;

import com.mongodb.client.FindIterable;

import com.mongodb.client.MongoCollection;

import com.mongodb.client.MongoCursor;

import com.mongodb.client.MongoDatabase;

import com.mongodb.client.model.Filters;

public class MongoDBJDBC{

    public static void main( String args[] ){

        try{

            // 连接到 mongodb 服务

            MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

            // 连接到数据库

            MongoDatabase mongoDatabase = mongoClient.getDatabase("mycol");

            System.out.println("Connect to database successfully");

            MongoCollection<Document> collection = mongoDatabase.getCollection("test");

            System.out.println("集合 test 选择成功");

            //更新文档    将文档中likes=100的文档修改为likes=200

            collection.updateMany(Filters.eq("likes", 100), new Document("$set",new Document("likes",200)));

            //检索查看结果
```

```

        FindIterable<Document> findIterable = collection.find();

        MongoClient mongoClient = findIterable.iterator();

        while(mongoCursor.hasNext()){

            System.out.println(mongoCursor.next());

        }

    }catch(Exception e){

        System.err.println( e.getClass().getName() + ": " + e.getMessage() );

    }

}

}

```

编译运行以上程序，输出结果如下：

```

Connect to database successfully

集合 test 选择成功

Document{{_id=56e65fb1fd57a86304fe2692, title=MongoDB, description=database, likes=200, by=Fly}}

```

删除第一个文档

要删除集合中的第一个文档，首先你需要使用`com.mongodb.DBCollection`类中的 `findOne()`方法来获取第一个文档，然后使用`remove`方法删除。

代码片段如下：

```

import org.bson.Document;

import com.mongodb.MongoClient;

import com.mongodb.client.FindIterable;

import com.mongodb.client.MongoCollection;

import com.mongodb.client.MongoCursor;

import com.mongodb.client.MongoDatabase;

import com.mongodb.client.model.Filters;

public class MongoDBJDBC{

    public static void main( String args[] ){

        try{

            // 连接到 mongodb 服务

```

```

MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

// 连接到数据库

MongoDatabase mongoDatabase = mongoClient.getDatabase("mycol");

System.out.println("Connect to database successfully");

MongoCollection<Document> collection = mongoDatabase.getCollection("test");

System.out.println("集合 test 选择成功");

//删除符合条件的第一个文档

collection.deleteOne(Filters.eq("likes", 200));

//删除所有符合条件的文档

collection.deleteMany (Filters.eq("likes", 200));

//检索查看结果

FindIterable<Document> findIterable = collection.find();

MongoCursor<Document> mongoCursor = findIterable.iterator();

while(mongoCursor.hasNext()){

    System.out.println(mongoCursor.next());

}

}catch(Exception e){

    System.err.println( e.getClass().getName() + ": " + e.getMessage() );

}

}

}

```

编译运行以上程序，输出结果如下：

```

Connect to database successfully

集合 test 选择成功

```

更多操作可以参考：<http://mongodb.github.io/mongo-java-driver/3.0/driver/getting-started/quick-tour/>

参考文档：<http://blog.csdn.net/ererfei/article/details/50857103>

MongoDB PHP 扩展

本教程将向大家介绍如何在Linux、window、Mac平台上安装MongoDB扩展。

Linux 上安装 MongoDB PHP 扩展

在终端上安装

你可以在 Linux 中执行以下命令来安装 MongoDB 的 PHP 扩展驱动

```
$ sudo pecl install mongo
```

使用php的pecl安装命令必须保证网络连接可用以及root权限。

安装手册

如果你想通过源码来编译扩展驱动。你必须手动编译源码包，这样做的好是最新修正的bug包含在源码包中。

你可以在 PHP 官网下载 MongoDB PHP 驱动包,下载地址: <http://pecl.php.net/package/mongodb>。

Available Releases				
Version	State	Release Date	Downloads	
1.5.2	stable	2018-08-01	mongodb-1.5.2.tgz (1032.4kB)	DLL
1.5.1	stable	2018-07-09	mongodb-1.5.1.tgz (1028.5kB)	DLL
1.5.0	stable	2018-06-26	mongodb-1.5.0.tgz (1030.1kB)	DLL
1.4.4	stable	2018-06-06	mongodb-1.4.4.tgz (1019.0kB)	DLL
1.4.3	stable	2018-04-19	mongodb-1.4.3.tgz (1018.4kB)	DLL

完整安装命令如下:

```
$ wget http://pecl.php.net/get/mongodb-1.5.2.tgz

$ cd /mongodb-1.5.2

$ phpize

$ ./configure

$ make && make install
```

如果你的 **php** 是自己编译的，则安装方法如下(假设是编译在 `/usr/local/php` 目录中):

```
$ wget http://pecl.php.net/get/mongodb-1.5.2.tgz

$ cd /mongodb-1.5.2

$ /usr/local/php/bin/phpize

$ ./configure --with-php-config=/usr/local/php/bin/php-config

$ make && make install
```

安装成功后，会有类似以下安装目录信息输出：

```
...

Installing shared extensions:      /usr/lib/php/extensions/debug-non-zts-20151012/
```

执行以上命令后，你需要修改**php.ini**文件，在 **php.ini** 文件中添加**mongo**配置，配置如下：

```
extension_dir=/usr/lib/php/extensions/debug-non-zts-20151012/

extension=mongodb.so
```

注意：你需要指明 **extension_dir** 配置项的路径。

可以通过以下命令查看目录地址：

```
$ php -i | grep extension_dir

extension_dir => /usr/lib/php/extensions/debug-non-zts-20151012 =>

                /usr/lib/php/extensions/debug-non-zts-20151012
```

Window 上安装 MongoDB PHP扩展

PECL 上已经提供了用于 Window 平台的预编译 **php mongodb** 驱动二进制包(下载地址：<https://pecl.php.net/package/mongodb>)，你可以下载与你**p**hp对应的版本，但是你需要注意以下几点问题：

VC6 是运行于 **Apache** 服务器

'Thread safe'（线程安全）是运行在**Apache**上以模块的**PHP**上，如果你以**CGI**的模式运行**PHP**，请选择非线程安全模式（'non-thread safe'）。

VC9是运行于 **IIS** 服务器上。

下载完你需要的二进制包后，解压缩压缩包，将 **php_mongodb.dll**文件添加到你的**PHP**扩展目录中（**ext**）。**ext**目录通常在**PHP**安装目录下的**ext**目录。

打开**php**配置文件 **php.ini** 添加以下配置：

```
extension=php_mongodb.dll
```

重启服务器。

通过浏览器访问**phpinfo**，如果安装成功，就会看到类型以下的信息：

mongo

MongoDB Support	enabled
Version	1.2.0-

Directive	Local Value	Master Value
mongo.allow_empty_keys	0	0
mongo.allow_persistent	1	1
mongo.auto_reconnect	1	1
mongo.chunk_size	262144	262144
mongo.cmd	\$	\$
mongo.default_host	localhost	localhost
mongo.default_port	27017	27017
mongo.long_as_object	0	0
mongo.native_long	0	0
mongo.no_id	0	0
mongo.utf8	1	1

MAC 中安装 MongoDB PHP扩展驱动

你可以使用 `autoconf` 安装 MongoDB PHP 扩展驱动。

你可以使用 `Xcode` 安装 MongoDB PHP 扩展驱动。

如果你使用 `XAMPP`，你可以使用以下命令安装 MongoDB PHP 扩展驱动：

```
sudo /Applications/XAMPP/xamppfiles/bin/pecl install mongo
```

如果以上命令在 `XMPP` 或者 `MAMP` 中不起作用，你需要在 `Github` 上下载兼容的预编译包。

然后添加 `extension=mongodb.so` 配置到你的 `php.ini` 文件中。

[MongoDB 连接](#)

MongoDB 插入文档 [MongoDB 插入文档](#)

[点我分享笔记](#)

反馈/建议



MongoDB Java	MongoDB 关系 MongoDB 关系
------------------------------	---------------------------------------

MongoDB PHP

在php中使用mongodb你必须使用 `mongodb` 的 `php`驱动。

MongoDB PHP在各平台上的安装及驱动包下载请查看:[PHP安装MongoDB扩展驱动](#)

如果你使用的是 `PHP7`，请参阅：[PHP7 MongoDB 安装与使用](#)。

确保连接及选择一个数据库

为了确保正确连接，你需要指定数据库名，如果数据库在mongoDB中不存在，mongoDB会自动创建
代码片段如下：

```
<?php

$m = new MongoClient(); // 连接默认主机和端口为：mongodb://localhost:27017

$db = $m->test; // 获取名称为 "test" 的数据库

?>
```

创建集合

创建集合的代码片段如下：

```
<?php

$m = new MongoClient(); // 连接

$db = $m->test; // 获取名称为 "test" 的数据库

$collection = $db->createCollection("runoob");

echo "集合创建成功";

?>
```

执行以上程序，输出结果如下：

```
集合创建成功
```

插入文档

在mongoDB中使用 insert() 方法插入文档：

插入文档代码片段如下：

```
<?php

$m = new MongoClient(); // 连接到mongodb

$db = $m->test; // 选择一个数据库

$collection = $db->runoob; // 选择集合

$document = array(

    "title" => "MongoDB",

    "description" => "database",

    "likes" => 100,

    "url" => "http://www.runoob.com/mongodb/",
```



```
"by", "菜鸟教程"

);

$collection->insert($document);

echo "数据插入成功";

?>
```

执行以上程序，输出结果如下：

```
数据插入成功
```

然后我们在 mongo 客户端使用 `db.runoob.find().pretty()`; 命令查看数据：

```
{
  "_id" : ObjectId("57512b3a57c9150f178b4567"),
  "title" : "MongoDB",
  "description" : "database",
  "likes" : NumberLong(100),
  "url" : "http://www.runoob.com/mongodb/",
  "0" : "by",
  "1" : "菜鸟教程 "
}
```

查找文档

使用`find()`方法来读取集合中的文档。

读取使用文档的代码片段如下：

```
<?php

$m = new MongoClient();    // 连接到mongodb

$db = $m->test;             // 选择一个数据库

$collection = $db->runoob; // 选择集合

$cursor = $collection->find();

// 迭代显示文档标题

foreach ($cursor as $document) {

    echo $document["title"] . "\n";

}

?>
```

执行以上程序，输出结果如下：

```
MongoDB
```

更新文档

使用 `update()` 方法来更新文档。

以下实例将更新文档中的标题为' MongoDB 教程'， 代码片段如下：

```
<pre>

<?php

$m = new MongoClient();    // 连接到mongodb

$db = $m->test;             // 选择一个数据库

$collection = $db->runoob; // 选择集合

// 更新文档

$collection->update(array("title"=>"MongoDB"), array('$set'=>array("title"=>"MongoDB 教程")));

// 显示更新后的文档

$cursor = $collection->find();

// 循环显示文档标题

foreach ($cursor as $document) {

    echo $document["title"] . "\n";

}

?>
```

执行以上程序，输出结果如下：

```
MongoDB 教程
```

然后我们在 mongo 客户端使用 `db.runoob.find().pretty()`; 命令查看数据：

```
{
  "_id" : ObjectId("57512b3a57c9150f178b4567"),
  "title" : "MongoDB",
  "description" : "database",
  "likes" : NumberLong(100),
  "url" : "http://www.runoob.com/mongodb/",
  "0" : "by",
  "1" : "菜鸟教程 "
}
```

删除文档

使用 `remove()` 方法来删除文档。

以下实例中我们将移除 'title' 为 'MongoDB 教程' 的一条数据记录。， 代码片段如下：

```
<?php
```

```
$m = new MongoClient();    // 连接到mongodb

$db = $m->test;             // 选择一个数据库

$collection = $db->runoob; // 选择集合


// 移除文档

$collection->remove(array("title"=>"MongoDB 教程"), array("justOne" => true));


// 显示可用文档数据

$cursor = $collection->find();

foreach ($cursor as $document) {

    echo $document["title"] . "\n";

}

?>
```

除了以上实例外，在php中你还可以使用findOne(), save(), limit(), skip(), sort()等方法来操作Mongodb数据库。
更多的操作方法可以参考 Mongodb 核心类：<http://php.net/manual/zh/mongo.core.php>。

[MongoDB Java](#)

MongoDB 关系 [MongoDB 关系](#)

[点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[MongoDB 删除数据库](#)

Mac OSX 平台安装 MongoDB [MongoDB](#)

PHP7 MongoDB 安装与使用

本文教程只适合在 PHP7 的环境，如果你是 PHP5 环境，你可以参阅 [PHP MongoDB 安装与使用](#)。

PHP7 Mongodb 扩展安装

我们使用 pecl 命令来安装：

```
$ /usr/local/php7/bin/pecl install mongodb
```

执行成功后，会输出以下结果：

```
.....

Build process completed successfully

Installing '/usr/local/php7/lib/php/extensions/no-debug-non-zts-20151012/mongodb.so'

install ok: channel://pecl.php.net/mongodb-1.1.7

configuration option "php_ini" is not set to php.ini location

You should add "extension=mongodb.so" to php.ini
```

接下来我们打开 `php.ini` 文件，添加 `extension=mongodb.so` 配置。

可以直接执行以下命令来添加。

```
$ echo "extension=mongodb.so" >> `/usr/local/php7/bin/php --ini | grep "Loaded Configuration" | sed -e "s|.*:|s*||"``
```

注意：以上执行的命令中 `php7` 的安装目录为 `/usr/local/php7/`，如果你安装在其他目录，需要相应修改 `pecl` 与 `php` 命令的路径。

Mongodb 使用

PHP7 连接 MongoDB 语法如下：

```
$manager = new MongoDB\Driver\Manager("mongodb://localhost:27017");
```

插入数据

将 `name` 为"菜鸟教程"的数据插入到 `test` 数据库的 `runoob` 集合中。

```
<?php

$bulk = new MongoDB\Driver\BulkWrite;

$document = ['_id' => new MongoDB\BSON\ObjectId, 'name' => '菜鸟教程'];

$_id= $bulk->insert($document);

var_dump($_id);

$manager = new MongoDB\Driver\Manager("mongodb://localhost:27017");

$writeConcern = new MongoDB\Driver\WriteConcern(MongoDB\Driver\WriteConcern::MAJORITY, 1000);

$result = $manager->executeBulkWrite('test.runoob', $bulk, $writeConcern);
```

```
?>
```

读取数据

这里我们将三个网址数据插入到 **test** 数据库的 **sites** 集合，并读取迭代出来：

```
<?php

$manager = new MongoDB\Driver\Manager("mongodb://localhost:27017");

// 插入数据

$bulk = new MongoDB\Driver\BulkWrite;

$bulk->insert(['x' => 1, 'name'=>'菜鸟教程', 'url' => 'http://www.runoob.com']);

$bulk->insert(['x' => 2, 'name'=>'Google', 'url' => 'http://www.google.com']);

$bulk->insert(['x' => 3, 'name'=>'taobao', 'url' => 'http://www.taobao.com']);

$manager->executeBulkWrite('test.sites', $bulk);

$filter = ['x' => ['$gt' => 1]];

$options = [

    'projection' => ['_id' => 0],

    'sort' => ['x' => -1],

];

// 查询数据

$query = new MongoDB\Driver\Query($filter, $options);

$cursor = $manager->executeQuery('test.sites', $query);

foreach ($cursor as $document) {

    print_r($document);

}

?>
```

输出结果为：

```
stdClass Object

(
```

```

    [x] => 3

    [name] => taobao

    [url] => http://www.taobao.com

)

stdClass Object

(

    [x] => 2

    [name] => Google

    [url] => http://www.google.com

)

```

更新数据

接下来我们将更新 **test** 数据库 **sites** 集合中 **x** 为 **2** 的数据：

```

<?php

$bulk = new MongoDB\Driver\BulkWrite;

$bulk->update(

    ['x' => 2],

    ['$set' => ['name' => '菜鸟工具', 'url' => 'tool.runoob.com']],

    ['multi' => false, 'upsert' => false]

);

$manager = new MongoDB\Driver\Manager("mongodb://localhost:27017");

$writeConcern = new MongoDB\Driver\WriteConcern(MongoDB\Driver\WriteConcern::MAJORITY, 1000);

$result = $manager->executeBulkWrite('test.sites', $bulk, $writeConcern);

?>

```

接下来我们使用 **"db.sites.find()"** 命令查看数据的变化，**x** 为 **2** 的数据已经变成了菜鸟工具：

```

{ "_id" : ObjectId("57511e43b3ab2214705adfe1"), "x" : 1, "name" : "菜鸟教程", "url" : "http://www.runoob.com" }
{ "_id" : ObjectId("57511e43b3ab2214705adfe2"), "x" : 2, "name" : "菜鸟工具", "url" : "tool.runoob.com" }
{ "_id" : ObjectId("57511e43b3ab2214705adfe3"), "x" : 3, "name" : "taobao", "url" : "http://www.taobao.com" }

```

删除数据

以下实例删除了 **x** 为 **1** 和 **x** 为 **2** 的数据，注意 **limit** 参数的区别：

```

<?php

$bulk = new MongoDB\Driver\BulkWrite;

```

```
$bulk->delete(['x' => 1], ['limit' => 1]); // limit 为 1 时，删除第一条匹配数据

$bulk->delete(['x' => 2], ['limit' => 0]); // limit 为 0 时，删除所有匹配数据


$manager = new MongoDB\Driver\Manager("mongodb://localhost:27017");

$writeConcern = new MongoDB\Driver\WriteConcern(MongoDB\Driver\WriteConcern::MAJORITY, 1000);

$result = $manager->executeBulkWrite('test.sites', $bulk, $writeConcern);

?>
```

更多使用方法请参考：<http://php.net/manual/en/book.mongodb.php>。

[MongoDB 删除数据库](#)

[Mac OSX 平台安装 MongoDB](#)

[点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[Node.js 连接 MySQL](#)

Node.js 连接 MongoDB

MongoDB是一种文档导向数据库管理系统，由C++撰写而成。

本章节我们将为大家介绍如何使用 Node.js 来连接 MongoDB，并对数据库进行操作。

如果你还没有 MongoDB 的基本知识，可以参考我们的教程：[MongoDB 教程](#)。

安装驱动

本教程使用了[淘宝定制的 cnpm 命令](#)进行安装：

```
$ cnpm install mongodb
```

接下来我们来实现增删改查功能。

创建数据库

要在 MongoDB 中创建一个数据库，首先我们需要创建一个 MongoClient 对象，然后配置好指定的 URL 和 端口号。

如果数据库不存在，MongoDB 将创建数据库并建立连接。

创建连接

```
var MongoClient = require('mongodb').MongoClient;
```

```
var url = "mongodb://localhost:27017/runoob";
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  console.log("数据库已创建!");
  db.close();
});
```

创建集合

我们可以使用 `createCollection()` 方法来创建集合：

创建集合

```
var MongoClient = require('mongodb').MongoClient;
var url = 'mongodb://localhost:27017/runoob';
MongoClient.connect(url, function (err, db) {
  if (err) throw err;
  console.log('数据库已创建');
  var dbase = db.db("runoob");
  dbase.createCollection('site', function (err, res) {
    if (err) throw err;
    console.log("创建集合!");
    db.close();
  });
});
```

数据库操作(CURD)

与 MySQL 不同的是 MongoDB 会自动创建数据库和集合，所以使用前我们不需要手动去创建。

插入数据

以下实例我们连接数据库 `runoob` 的 `site` 表，并插入一条数据条数据，使用 `insertOne()`：

插入一条数据

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("runoob");
  var myobj = { name: "菜鸟教程", url: "www.runoob" };
  dbo.collection("site").insertOne(myobj, function(err, res) {
    if (err) throw err;
    console.log("文档插入成功");
    db.close();
  });
});
```

执行以下命令输出就结果为：

```
$ node test.js
```

文档插入成功

从输出结果来看，数据已插入成功。

我们也可以打开 MongoDB 的客户端查看数据，如：

```
> show dbs
```

```
runoob  0.000GB          # 自动创建了 runoob 数据库
```

```
> show tables
```



```
site                                # 自动创建了 site 集合（数据表）

> db.site.find()

{ "_id" : ObjectId("5a794e36763eb821b24db854"), "name" : "菜鸟教程", "url" : "www.runoob" }

>
```

如果要插入多条数据可以使用 `insertMany()`：

插入多条数据

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("runoob");
  var myobj = [
    { name: '菜鸟工具', url: 'https://c.runoob.com', type: 'cn' },
    { name: 'Google', url: 'https://www.google.com', type: 'en' },
    { name: 'Facebook', url: 'https://www.google.com', type: 'en' }
  ];
  dbo.collection("site").insertMany(myobj, function(err, res) {
    if (err) throw err;
    console.log("插入的文档数量为: " + res.insertedCount);
    db.close();
  });
});
```

`res.insertedCount` 为插入的条数。

查询数据

可以使用 `find()` 来查找数据，`find()` 可以返回匹配条件的所有数据。 如果未指定条件，`find()` 返回集合中的所有数据。

find()

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("runoob");
  dbo.collection("site").find({}).toArray(function(err, result) { // 返回集合中所有数据
    if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

以下实例检索 `name` 为 "菜鸟教程" 的实例：

查询指定条件的数据

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("runoob");
  var whereStr = {"name":'菜鸟教程'}; // 查询条件
  dbo.collection("site").find(whereStr).toArray(function(err, result) {
    if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

执行以下命令输出就结果为：

```
[ { _id: 5a794e36763eb821b24db854,

  name: '菜鸟教程',

  url: 'www.runoob' } ]
```

更新数据

我们也可以对数据库的数据进行修改，以下实例将 **name** 为 "菜鸟教程" 的 **url** 改为 <https://www.runoob.com>：

更新一条数据

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("runoob");
  var whereStr = {"name":'菜鸟教程'}; // 查询条件
  var updateStr = {$set: { "url" : "https://www.runoob.com" }};
  dbo.collection("site").updateOne(whereStr, updateStr, function(err, res) {
    if (err) throw err;
    console.log("文档更新成功");
    db.close();
  });
});
```

执行成功后，进入 **mongo** 管理工具查看数据已修改：

```
> db.site.find().pretty()

{

  "_id" : ObjectId("5a794e36763eb821b24db854"),

  "name" : "菜鸟教程",

  "url" : "https://www.runoob.com"    // 已修改为 https

}
```

如果要更新所有符合条的文档数据可以使用 **updateMany()**：

更新多条数据

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("runoob");
  var whereStr = {"type":'en'}; // 查询条件
  var updateStr = {$set: { "url" : "https://www.runoob.com" }};
  dbo.collection("site").updateMany(whereStr, updateStr, function(err, res) {
    if (err) throw err;
    console.log(res.result.nModified + " 条文档被更新");
    db.close();
  });
});
```

result.nModified 为更新的条数。

删除数据

以下实例将 **name** 为 "菜鸟教程" 的数据删除：

删除一条数据

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("runoob");
  var whereStr = {"name": '菜鸟教程'}; // 查询条件
  dbo.collection("site").deleteOne(whereStr, function(err, obj) {
    if (err) throw err;
    console.log("文档删除成功");
    db.close();
  });
});
```

执行成功后，进入 **mongo** 管理工具查看数据已删除：

```
> db.site.find()

>
```

如果要删除多条语句可以使用 **deleteMany()** 方法

以下实例将 **type** 为 **en** 的所有数据删除：

删除多条数据

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("runoob");
  var whereStr = { type: "en" }; // 查询条件
  dbo.collection("site").deleteMany(whereStr, function(err, obj) {
    if (err) throw err;
    console.log(obj.result.n + " 条文档被删除");
    db.close();
  });
});
```

obj.result.n 删除的条数。

排序

排序 使用 **sort()** 方法，该方法接受一个参数，规定是升序(**1**)还是降序(**-1**)。

例如：

```
{ type: 1 } // 按 type 字段升序

{ type: -1 } // 按 type 字段降序
```

按 **type** 升序排列：

排序

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("runoob");
  var mysort = { type: 1 };
  dbo.collection("site").find().sort(mysort).toArray(function(err, result) {
    if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

```
});  
});
```

查询分页

如果要设置指定的返回条数可以使用 `limit()` 方法，该方法只接受一个参数，指定了返回的条数。

limit(): 读取两条数据

```
var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("runoob");  
  dbo.collection("site").find().limit(2).toArray(function(err, result) {  
    if (err) throw err;  
    console.log(result);  
    db.close();  
  });  
});
```

如果要指定跳过的条数，可以使用 `skip()` 方法。

skip(): 跳过前面两条数据，读取两条数据

```
var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("runoob");  
  dbo.collection("site").find().skip(2).limit(2).toArray(function(err, result) {  
    if (err) throw err;  
    console.log(result);  
    db.close();  
  });  
});
```

连接操作

mongoDB 不是一个关系型数据库，但我们可以使用 `$lookup` 来实现左连接。

例如我们有两个集合数据分别为：

集合1: orders

```
[  
  
  { _id: 1, product_id: 154, status: 1 }  
  
]
```

集合2: products

```
[  
  
  { _id: 154, name: '笔记本电脑' },  
  
  { _id: 155, name: '耳机' },  
  
  { _id: 156, name: '台式电脑' }  
  
]
```

\$lookup 实现左连接

```
var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://127.0.0.1:27017/";  
MongoClient.connect(url, function(err, db) {
```

```
if (err) throw err;
var dbo = db.db("runoob");
dbo.collection('orders').aggregate([
  { $lookup:
    {
      from: 'products', // 右集合
      localField: 'product_id', // 左集合 join 字段
      foreignField: '_id', // 右集合 join 字段
      as: 'orderdetails' // 新生成字段（类型array）
    }
  }
]).toArray(function(err, res) {
  if (err) throw err;
  console.log(JSON.stringify(res));
  db.close();
});
});
```

删除集合

我们可以使用 `drop()` 方法来删除集合：

drop()

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("runoob");
  // 删除 test 集合
  dbo.collection("test").drop(function(err, delOK) { // 执行成功 delOK 返回 true, 否则返回 false
    if (err) throw err;
    if (delOK) console.log("集合已删除");
    db.close();
  });
});
```

☐ Node.js 连接 MySQL

☐ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

☐ MongoDB PHP

MongoDB 数据库引用 ☐

MongoDB 关系

MongoDB 的关系表示多个文档之间在逻辑上的相互联系。

文档间可以通过嵌入和引用来建立联系。

MongoDB 中的关系可以是：

1:1 (1对1)

1: N (1对多)

N: 1 (多对1)

N: N (多对多)

接下来我们来考虑下用户与用户地址的关系。

一个用户可以有多个地址，所以是一对多的关系。

以下是 **user** 文档的简单结构：

```
{
  "_id": ObjectId("52ffc33cd85242f436000001"),
  "name": "Tom Hanks",
  "contact": "987654321",
  "dob": "01-01-1991"
}
```

以下是 **address** 文档的简单结构：

```
{
  "_id": ObjectId("52ffc4a5d85242602e000000"),
  "building": "22 A, Indiana Apt",
  "pincode": 123456,
  "city": "Los Angeles",
  "state": "California"
}
```

嵌入式关系

使用嵌入式方法，我们可以把用户地址嵌入到用户的文档中：

```
{
  "_id": ObjectId("52ffc33cd85242f436000001"),
  "contact": "987654321",
  "dob": "01-01-1991",
  "name": "Tom Benzamin",
  "address": [
    {
      "building": "22 A, Indiana Apt",
      "pincode": 123456,
```

```
      "city": "Los Angeles",

      "state": "California"

    },

    {

      "building": "170 A, Acropolis Apt",

      "pincode": 456789,

      "city": "Chicago",

      "state": "Illinois"

    }

  ]

}
```

以上数据保存在单一的文档中，可以比较容易的获取和维护数据。 你可以这样查询用户的地址：

```
>db.users.findOne({"name":"Tom Benzamin"}, {"address":1})
```

注意：以上查询中 **db** 和 **users** 表示数据库和集合。

这种数据结构的缺点是，如果用户和用户地址在不断增加，数据量不断变大，会影响读写性能。

引用式关系

引用式关系是设计数据库时经常用到的方法，这种方法把用户数据文档和用户地址数据文档分开，通过引用文档的 **id** 字段来建立关系。

```
{

  "_id":ObjectId("52ffc33cd85242f436000001"),

  "contact": "987654321",

  "dob": "01-01-1991",

  "name": "Tom Benzamin",

  "address_ids": [

    ObjectId("52ffc4a5d85242602e000000"),

    ObjectId("52ffc4a5d85242602e000001")

  ]

}
```

以上实例中，用户文档的 **address_ids** 字段包含用户地址的对象id（**ObjectId**）数组。

我们可以读取这些用户地址的对象id（**ObjectId**）来获取用户的详细地址信息。

这种方法需要两次查询，第一次查询用户地址的对象id（**ObjectId**），第二次通过查询的id获取用户的详细地址信息。

```
>var result = db.users.findOne({"name":"Tom Benzamin"}, {"address_ids":1})
```

```
>var addresses = db.address.find({"_id":{"$in":result["address_ids"]}})
```

[MongoDB PHP](#)

[MongoDB 数据库引用](#)

[MongoDB PHP](#)

[MongoDB 数据库引用](#)

反馈/建议

Copyright©2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号: 闽ICP备15012807号-1

MongoDB 关系

MongoDB 覆盖索引查询

MongoDB 关系

MongoDB 覆盖索引查询

MongoDB 数据库引用

在上一章节MongoDB关系中我们提到了MongoDB的引用来规范数据结构文档。

MongoDB 引用有两种:

手动引用 (Manual References)

DBRefs

DBRefs vs 手动引用

考虑这样的一个场景，我们在不同的集合中 (address_home, address_office, address_mailing, 等) 存储不同的地址（住址，办公室地址，邮件地址等）。

这样，我们在调用不同地址时，也需要指定集合，一个文档从多个集合引用文档，我们应该使用 DBRefs。

使用 DBRefs

DBRef的形式:

三个字段表示的意义为:

\$ref: 集合名称

\$id: 引用的id

\$db:数据库名称, 可选参数

以下实例中用户数据文档使用了 DBRef, 字段 address:

```
{
  "_id": ObjectId("53402597d852426020000002"),
```



```
"address": {  
  
  "$ref": "address_home",  
  
  "$id": ObjectId("534009e4d852427820000002"),  
  
  "$db": "runoob"},  
  
  "contact": "987654321",  
  
  "dob": "01-01-1991",  
  
  "name": "Tom Benzamin"  
  
}
```

address DBRef 字段指定了引用的地址文档是在 **runoob** 数据库下的 **address_home** 集合，id 为 **534009e4d852427820000002**。

以下代码中，我们通过指定 **\$ref** 参数（**address_home** 集合）来查找集合中指定id的用户地址信息：

```
>var user = db.users.findOne({"name":"Tom Benzamin"})  
  
>var dbRef = user.address  
  
>db[dbRef.$ref].findOne({"_id":(dbRef.$id)})
```

以上实例返回了 **address_home** 集合中的地址数据：

```
{  
  
  "_id" : ObjectId("534009e4d852427820000002"),  
  
  "building" : "22 A, Indiana Apt",  
  
  "pincode" : 123456,  
  
  "city" : "Los Angeles",  
  
  "state" : "California"  
  
}
```

[MongoDB 关系](#)

[MongoDB 覆盖索引查询](#)

[点我分享笔记](#)

反馈/建议

MongoDB 覆盖索引查询

官方的MongoDB的文档中说明，覆盖查询是以下的查询：

所有的查询字段是索引的一部分

所有的查询返回字段在同一个索引中

由于所有出现在查询中的字段是索引的一部分， MongoDB 无需在整个数据文档中检索匹配查询条件和返回使用相同索引的查询结果。因为索引存在于RAM中，从索引中获取数据比通过扫描文档读取数据要快得多。

使用覆盖索引查询

为了测试覆盖索引查询，使用以下 `users` 集合：

```
{
  "_id": ObjectId("53402597d852426020000002"),
  "contact": "987654321",
  "dob": "01-01-1991",
  "gender": "M",
  "name": "Tom Benzamin",
  "user_name": "tombenzamin"
}
```

我们在 `users` 集合中创建联合索引，字段为 `gender` 和 `user_name`：

```
>db.users.ensureIndex({gender:1,user_name:1})
```

现在，该索引会覆盖以下查询：

```
>db.users.find({gender:"M"},{user_name:1,_id:0})
```

也就是说，对于上述查询，MongoDB的不会去数据库文件中查找。相反，它会从索引中提取数据，这是非常快速的数据查询。

由于我们的索引中不包括 `_id` 字段，`_id`在查询中会默认返回，我们可以在MongoDB的查询结果集中排除它。

下面的实例没有排除 `_id`，查询就不会被覆盖：

```
>db.users.find({gender:"M"},{user_name:1})
```

最后，如果是以下的查询，不能使用覆盖索引查询：

所有索引字段是一个数组

所有索引字段是一个子文档

[MongoDB 数据库引用](#)

[MongoDB 查询分析](#)

[点我分享笔记](#)

[反馈/建议](#)

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[MongoDB 覆盖索引查询](#)

[MongoDB 原子操作](#)

MongoDB 查询分析

MongoDB 查询分析可以确保我们建议的索引是否有效，是查询语句性能分析的重要工具。

MongoDB 查询分析常用函数有：`explain()` 和 `hint()`。

使用 `explain()`

`explain` 操作提供了查询信息，使用索引及查询统计等。有利于我们对索引的优化。

接下来我们在 `users` 集合中创建 `gender` 和 `user_name` 的索引：

```
>db.users.ensureIndex({gender:1,user_name:1})
```

现在在查询语句中使用 `explain`：

```
>db.users.find({gender:"M"},{user_name:1,_id:0}).explain()
```

以上的 `explain()` 查询返回如下结果：

```
{
  "cursor" : "BtreeCursor gender_1_user_name_1",
  "isMultiKey" : false,
  "n" : 1,
  "nscannedObjects" : 0,
  "nscanned" : 1,
```

```
"nscannedObjectsAllPlans" : 0,

"nscannedAllPlans" : 1,

"scanAndOrder" : false,

"indexOnly" : true,

"nYields" : 0,

"nChunkSkips" : 0,

"millis" : 0,

"indexBounds" : {

  "gender" : [

    [

      "M",

      "M"

    ]

  ],

  "user_name" : [

    [

      {

        "$minElement" : 1

      },

      {

        "$maxElement" : 1

      }

    ]

  ]

}

}
```

现在，我们看看这个结果集的字段：

indexOnly: 字段为 **true**，表示我们使用了索引。

cursor: 因为这个查询使用了索引，MongoDB 中索引存储在B树结构中，所以这是也使用了 **BtreeCursor** 类型的游标。如果没有使用索引，游标的类型是 **BasicCursor**。这个键还会给出你所使用的索引的名称，你通过这个名称可以查看当前数据库下的**system.indexes**集合（系统自动创建，由于存储索引信息，这个稍微会提到）来得到索引的详细信息。

n: 当前查询返回的文档数量。

nscanned/nscannedObjects: 表明当前这次查询一共扫描了集合中多少个文档，我们的目的是，让这个数值和返回文档的数量越接近越好。

millis: 当前查询所需时间，毫秒数。

indexBounds: 当前查询具体使用的索引。

使用 hint()

虽然MongoDB查询优化器一般工作的很不错，但是也可以使用 **hint** 来强制 MongoDB 使用一个指定的索引。
这种方法某些情形下会提升性能。 一个有索引的 **collection** 并且执行一个多字段的查询(一些字段已经索引了)。
如下查询实例指定了使用 **gender** 和 **user_name** 索引字段来查询：

```
>db.users.find({gender:"M"},{user_name:1,_id:0}).hint({gender:1,user_name:1})
```

可以使用 **explain()** 函数来分析以上查询：

```
>db.users.find({gender:"M"},{user_name:1,_id:0}).hint({gender:1,user_name:1}).explain()
```

□ MongoDB 覆盖索引查询

MongoDB 原子操作 □

□ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

□ MongoDB 查询分析

MongoDB 高级索引 □

MongoDB 原子操作

mongodb不支持事务，所以，在你的项目中应用时，要注意这点。无论什么设计，都不要要求**mongodb**保证数据的完整性。
但是**mongodb**提供了许多原子操作，比如文档的保存，修改，删除等，都是原子操作。
所谓原子操作就是要么这个文档保存到**Mongodb**，要么没有保存到**Mongodb**，不会出现查询到的文档没有保存完整的情况。

原子操作数据模型

考虑下面的例子，图书馆的书籍及结账信息。

实例说明了在一个相同的文档中如何确保嵌入字段关联原子操作（**update**: 更新）的字段是同步的。

```
book = {  
  
  _id: 123456789,  
  
  title: "MongoDB: The Definitive Guide",  
  
  author: [ "Kristina Chodorow", "Mike Dirolf" ],
```

```
published_date: ISODate("2010-09-24"),

pages: 216,

language: "English",

publisher_id: "oreilly",

available: 3,

checkout: [ { by: "joe", date: ISODate("2012-10-15") } ]

}
```

你可以使用 `db.collection.findAndModify()` 方法来判断书籍是否可结算并更新新的结算信息。

在同一个文档中嵌入的 `available` 和 `checkout` 字段来确保这些字段是同步更新的：

```
db.books.findAndModify ( {

  query: {

    _id: 123456789,

    available: { $gt: 0 }

  },

  update: {

    $inc: { available: -1 },

    $push: { checkout: { by: "abc", date: new Date() } }

  }

} )
```

原子操作常用命令

\$set

用来指定一个键并更新键值，若键不存在并创建。

```
{ $set : { field : value } }
```

\$unset

用来删除一个键。

```
{ $unset : { field : 1 } }
```

\$inc

`$inc`可以对文档的某个值为数字型（只能为满足要求的数字）的键进行增减的操作。

```
{ $inc : { field : value } }
```

\$push

用法：

```
{ $push : { field : value } }
```

把value追加到field里面去，field一定要是数组类型才行，如果field不存在，会新增一个数组类型加进去。

\$pushAll

同\$push,只是一次可以追加多个值到一个数组字段内。

```
{ $pushAll : { field : value_array } }
```

\$pull

从数组field内删除一个等于value值。

```
{ $pull : { field : _value } }
```

\$addToSet

增加一个值到数组内，而且只有当这个值不在数组内才增加。

\$pop

删除数组的第一个或最后一个元素

```
{ $pop : { field : 1 } }
```

\$rename

修改字段名称

```
{ $rename : { old_field_name : new_field_name } }
```

\$bit

位操作，integer类型

```
{ $bit : { field : { and : 5 } } }
```

偏移操作符

```
> t.find() { "_id" : ObjectId("4b97e62bf1d8c7152c9ccb74"), "title" : "ABC", "comments" : [ { "by" : "joe", "votes" : 3 }, { "by" : "jane", "votes" : 7 } ] }
```

```
> t.update( {'comments.by':'joe'}, {$inc:{'comments.$.votes':1}}, false, true )
```

```
> t.find() { "_id" : ObjectId("4b97e62bf1d8c7152c9ccb74"), "title" : "ABC", "comments" : [ { "by" : "joe", "votes" : 4 }, { "by" : "jane", "votes" : 7 } ] }
```

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)[MongoDB 原子操作](#)[MongoDB 索引限制](#)

MongoDB 高级索引

考虑以下文档集合（**users**）：

```
{

  "address": {

    "city": "Los Angeles",

    "state": "California",

    "pincode": "123"

  },

  "tags": [

    "music",

    "cricket",

    "blogs"

  ],

  "name": "Tom Benzamin"

}
```

以上文档包含了 **address** 子文档和 **tags** 数组。

索引数组字段

假设我们基于标签来检索用户，为此我们需要对集合中的数组 **tags** 建立索引。

在数组中创建索引，需要对数组中的每个字段依次建立索引。所以在我们为数组 **tags** 创建索引时，会为 **music**、**cricket**、**blogs**三个值建立单独的索引。

使用以下命令创建数组索引：

```
>db.users.ensureIndex({"tags":1})
```

创建索引后，我们可以这样检索集合的 **tags** 字段：


```
>db.users.find({tags:"cricket"})
```

为了验证我们使用使用了索引，可以使用 **explain** 命令：

```
>db.users.find({tags:"cricket"}).explain()
```

以上命令执行结果中会显示 **"cursor": "BtreeCursor tags_1"**，则表示已经使用了索引。

索引子文档字段

假设我们需要通过 **city**、**state**、**pincode** 字段来检索文档，由于这些字段是子文档的字段，所以我们需要对子文档建立索引。

为子文档的三个字段创建索引，命令如下：

```
>db.users.ensureIndex({"address.city":1,"address.state":1,"address.pincode":1})
```

一旦创建索引，我们可以使用子文档的字段来检索数据：

```
>db.users.find({"address.city":"Los Angeles"})
```

查询表达不一定遵循指定的索引的顺序，**mongodb** 会自动优化。所以上面创建的索引将支持以下查询：

```
>db.users.find({"address.state":"California","address.city":"Los Angeles"})
```

同样支持以下查询：

```
>db.users.find({"address.city":"Los Angeles","address.state":"California","address.pincode":"123"})
```

[MongoDB 原子操作](#)

[MongoDB 索引限制](#)

[点我分享笔记](#)

[反馈/建议](#)

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



MongoDB 索引限制

额外开销

每个索引占据一定的存储空间，在进行插入，更新和删除操作时也需要对索引进行操作。所以，如果你很少对集合进行读取操作，建议不使用索引。

内存(RAM)使用

由于索引是存储在内存(RAM)中,你应该确保该索引的大小不超过内存的限制。

如果索引的大小大于内存的限制，MongoDB会删除一些索引，这将导致性能下降。

查询限制

索引不能被以下的查询使用：

正则表达式及非操作符，如 `$nin`, `$not`, 等。

算术运算符，如 `$mod`, 等。

`$where` 子句

所以，检测你的语句是否使用索引是一个好的习惯，可以用`explain`来查看。

索引键限制

从2.6版本开始，如果现有的索引字段的值超过索引键的限制，MongoDB中不会创建索引。

插入文档超过索引键限制

如果文档的索引字段值超过了索引键的限制，MongoDB不会将任何文档转换成索引的集合。与`mongorestore`和`mongoimport`工具类似。

最大范围

集合中索引不能超过64个

索引名的长度不能超过128个字符

一个复合索引最多可以有31个字段

MongoDB ObjectId

在前面几个章节中我们已经使用了MongoDB 的对象 `Id(ObjectId)`。

在本章节中，我们将了解的`ObjectId`的结构。

`ObjectId` 是一个12字节 `BSON` 类型数据，有以下格式：

前4个字节表示时间戳

接下来的3个字节是机器标识码

紧接的两个字节由进程id组成（PID）

最后三个字节是随机数。

MongoDB中存储的文档必须有一个"`_id`"键。这个键的值可以是任何类型的，默认是个`ObjectId`对象。

在一个集合里面，每个文档都有唯一的"`_id`"值，来确保集合里面每个文档都能被唯一标识。

MongoDB采用`ObjectId`，而不是其他比较常规的做法（比如自动增加的主键）的主要原因，因为在多个 服务器上同步自动增加主键值既费力还费时。

创建新的ObjectId

使用以下代码生成新的`ObjectId`：

```
>newObjectId = ObjectId()
```

上面的语句返回以下唯一生成的id：

```
ObjectId("5349b4ddd2781d08c09890f3")
```

你也可以使用生成的id来取代MongoDB自动生成的`ObjectId`：

```
>myObjectId = ObjectId("5349b4ddd2781d08c09890f4")
```

创建文档的时间戳

由于 `ObjectId` 中存储了 4 个字节的时间戳，所以你不需要为你的文档保存时间戳字段，你可以通过 `getTimestamp` 函数来获取文档的创建时间：

```
>ObjectId("5349b4ddd2781d08c09890f4").getTimestamp()
```

以上代码将返回 `ISO` 格式的文档创建时间：

```
ISODate("2014-04-12T21:49:17Z")
```

ObjectId 转换为字符串

在某些情况下，您可能需要将`ObjectId`转换为字符串格式。你可以使用下面的代码：

```
>new ObjectId().str
```

以上代码将返回`Guid`格式的字符串：：

5349b4ddd2781d08c09890f3

[MongoDB 索引限制](#)

[MongoDB Map Reduce](#)

[点我分享笔记](#)

[反馈/建议](#)

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号: 闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[MongoDB ObjectId](#)

[MongoDB 全文检索](#)

MongoDB Map Reduce

Map-Reduce是一种计算模型,简单的说就是将大量的工作(数据)分解(MAP)执行,然后再将结果合并成最终结果(REDUCE)。

MongoDB提供的Map-Reduce非常灵活,对于大规模数据分析也相当实用。

MapReduce 命令

以下是MapReduce的基本语法:

```
>db.collection.mapReduce(  
  
    function() {emit(key,value);},    //map 函数  
  
    function(key,values) {return reduceFunction},    //reduce 函数  
  
    {  
  
        out: collection,  
  
        query: document,  
  
        sort: document,  
  
        limit: number  
  
    }  
  
)
```

使用 MapReduce 要实现两个函数 Map 函数和 Reduce 函数,Map 函数调用 emit(key, value), 遍历 collection 中所有的记录, 将 key 与 value 传递给 Reduce 函数进行处理。

Map 函数必须调用 emit(key, value) 返回键值对。

参数说明:

map : 映射函数 (生成键值对序列,作为 **reduce** 函数参数)。

reduce 统计函数, **reduce**函数的任务就是将**key-values**变成**key-value**, 也就是把**values**数组变成一个单一的值**value**。。

out 统计结果存放集合 (不指定则使用临时集合,在客户端断开后自动删除)。

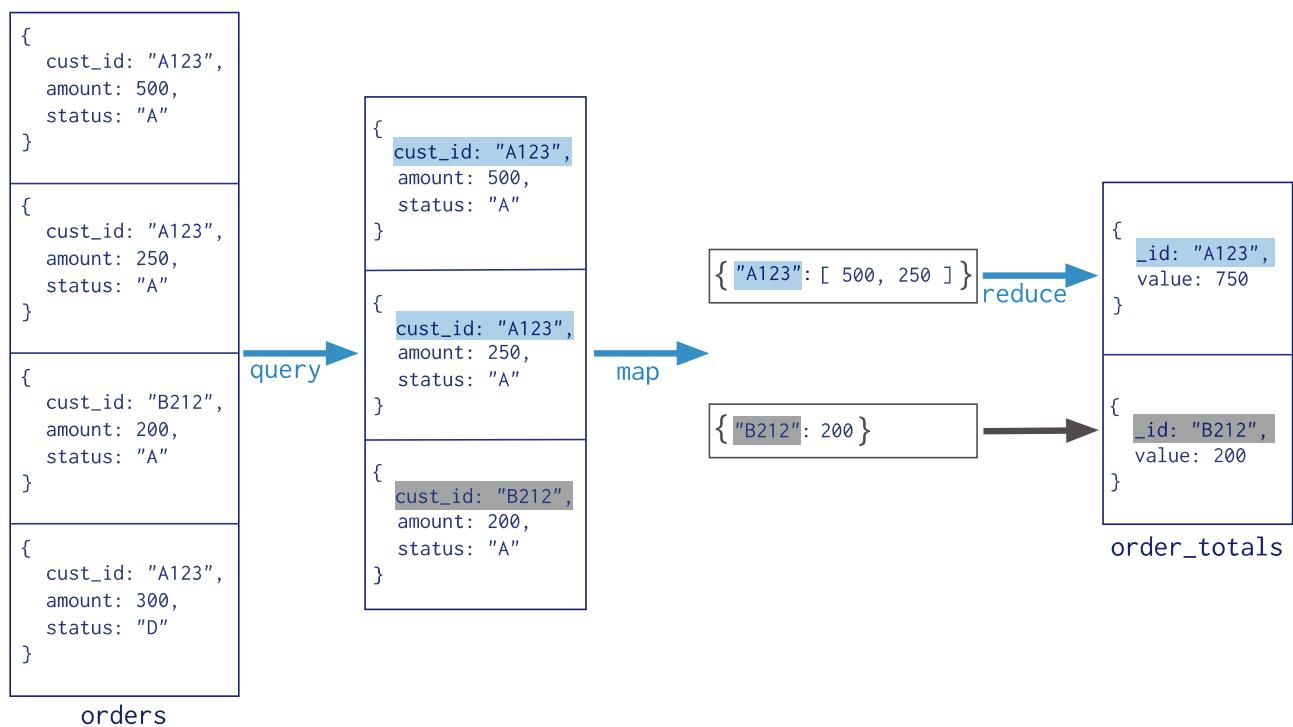
query 一个筛选条件, 只有满足条件的文档才会调用**map**函数。(**query.limit**, **sort**可以随意组合)

sort 和**limit**结合的**sort**排序参数 (也是在发往**map**函数前给文档排序), 可以优化分组机制

limit 发往**map**函数的文档数量的上限 (要是没有**limit**, 单独使用**sort**的用处不大)

以下实例在集合 **orders** 中查找 **status:"A"** 的数据, 并根据 **cust_id** 来分组, 并计算 **amount** 的总和。

```
Collection
↓
db.orders.mapReduce(
  map    → function() { emit( this.cust_id, this.amount ); },
  reduce → function(key, values) { return Array.sum( values ) },
  {
    query: { status: "A" },
    out: "order_totals"
  }
)
```



使用 MapReduce

考虑以下文档结构存储用户的文章, 文档存储了用户的 **user_name** 和文章的 **status** 字段:

```
>db.posts.insert({
  "post_text": "菜鸟教程, 最全的技术文档。",
  "user_name": "mark",
  "status": "active"
})

WriteResult({ "nInserted" : 1 })

>db.posts.insert({
```

```
"post_text": "菜鸟教程，最全的技术文档。",

"user_name": "mark",

"status": "active"

})

WriteResult({ "nInserted" : 1 })

>db.posts.insert({

    "post_text": "菜鸟教程，最全的技术文档。",

    "user_name": "mark",

    "status": "active"

})

WriteResult({ "nInserted" : 1 })

>db.posts.insert({

    "post_text": "菜鸟教程，最全的技术文档。",

    "user_name": "mark",

    "status": "active"

})

WriteResult({ "nInserted" : 1 })

>db.posts.insert({

    "post_text": "菜鸟教程，最全的技术文档。",

    "user_name": "mark",

    "status": "disabled"

})

WriteResult({ "nInserted" : 1 })

>db.posts.insert({

    "post_text": "菜鸟教程，最全的技术文档。",

    "user_name": "runoob",

    "status": "disabled"

})

WriteResult({ "nInserted" : 1 })

>db.posts.insert({

    "post_text": "菜鸟教程，最全的技术文档。",

    "user_name": "runoob",
```

```

        "status": "disabled"

    })

    WriteResult({ "nInserted" : 1 })

    >db.posts.insert({

        "post_text": "菜鸟教程，最全的技术文档。",

        "user_name": "runoob",

        "status": "active"

    })

    WriteResult({ "nInserted" : 1 })

```

现在，我们将在 `posts` 集合中使用 `mapReduce` 函数来选取已发布的文章(`status:"active"`)，并通过`user_name`分组，计算每个用户的文章数：

```

>db.posts.mapReduce(

    function() { emit(this.user_name,1); },

    function(key, values) {return Array.sum(values)},

    {

        query:{status:"active"},

        out:"post_total"

    }

)

```

以上 `mapReduce` 输出结果为：

```

{

    "result" : "post_total",

    "timeMillis" : 23,

    "counts" : {

        "input" : 5,

        "emit" : 5,

        "reduce" : 1,

        "output" : 2

    },

    "ok" : 1

}

```

结果表明，共有 5 个符合查询条件（`status:"active"`）的文档，在`map`函数中生成了 5 个键值对文档，最后使用`reduce`函数将相同的键值分为 2 组。
具体参数说明：

result: 储存结果的`collection`的名字,这是个临时集合，MapReduce的连接关闭后自动就被删除了。

timeMillis: 执行花费的时间，毫秒为单位

input: 满足条件被发送到`map`函数的文档个数

emit: 在`map`函数中`emit`被调用的次数，也就是所有集合中的数据总量

output: 结果集合中的文档个数（**count对调试非常有帮助**）

ok: 是否成功，成功为1

err: 如果失败，这里可以有失败原因，不过从经验上来看，原因比较模糊，作用不大

使用 `find` 操作符来查看 `mapReduce` 的查询结果：

```
>db.posts.mapReduce(

  function() { emit(this.user_name,1); },

  function(key, values) {return Array.sum(values)},

  {

    query:{status:"active"},

    out:"post_total"

  }

).find()
```

以上查询显示如下结果，两个用户 `tom` 和 `mark` 有两个发布的文章：

```
{ "_id" : "mark", "value" : 4 }

{ "_id" : "runoob", "value" : 1 }
```

用类似的方式，`MapReduce`可以被用来构建大型复杂的聚合查询。
`Map`函数和`Reduce`函数可以使用 `JavaScript` 来实现，使得`MapReduce`的使用非常灵活和强大。

☐

1 篇笔记

#1

☐

☐ 写笔记

临时集合参数是这样写的

```
out: { inline: 1 }
```

设置了 `{inline:1}` 将不会创建集合，整个 `Map/Reduce` 的操作将会在内存中进行。
注意，这个选项只有在结果集单个文档大小在16MB限制范围内时才有效。


```
db.users.mapReduce(map,reduce,{out:{inline:1}});
```

forthxu2个月前 [07-20]

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[MongoDB Map Reduce](#)

[MongoDB 正则表达式](#)

MongoDB 全文检索

全文检索对每一个词建立一个索引，指明该词在文章中出现的次数和位置，当用户查询时，检索程序就根据事先建立的索引进行查找，并将查找的结果反馈给用户的检索方式。

这个过程类似于通过字典中的检索字表查字的过程。

MongoDB 从 2.4 版本开始支持全文检索，目前支持15种语言的全文索引。

danish

dutch

english

finnish

french

german

hungarian

italian

norwegian

portuguese

romanian

russian

spanish

swedish

turkish

启用全文检索

MongoDB 在 2.6 版本以后是默认开启全文检索的，如果你使用之前的版本，你需要使用以下代码来启用全文检索：

```
>db.adminCommand({setParameter:true,textSearchEnabled:true})
```

或者使用命令：

```
mongod --setParameter textSearchEnabled=true
```

创建全文索引

考虑以下 `posts` 集合的文档数据，包含了文章内容（`post_text`）及标签(`tags`):

```
{
  "post_text": "enjoy the mongodb articles on Runoob",
  "tags": [
    "mongodb",
    "runoob"
  ]
}
```

我们可以对 `post_text` 字段建立全文索引，这样我们可以搜索文章内的内容：

```
>db.posts.ensureIndex({post_text:"text"})
```

使用全文索引

现在我们已经对 `post_text` 建立了全文索引，我们可以搜索文章中的关键词 `runoob`:

```
>db.posts.find({$text:{$search:"runoob"}})
```

以下命令返回了如下包含 `runoob` 关键词的文档数据：

```
{
  "_id" : ObjectId("53493d14d852429c10000002"),
  "post_text" : "enjoy the mongodb articles on Runoob",
  "tags" : [ "mongodb", "runoob" ]
}
```

如果你使用的是旧版本的 `MongoDB`，你可以使用以下命令：

```
>db.posts.runCommand("text",{search:"runoob"})
```

使用全文索引可以提高搜索效率。

删除全文索引

删除已存在的全文索引，可以使用 **find** 命令查找索引名：

```
>db.posts.getIndexes()
```

通过以上命令获取索引名，本例的索引名为**post_text_text**，执行以下命令来删除索引：

```
>db.posts.dropIndex("post_text_text")
```



MongoDB 正则表达式

正则表达式是使用单个字符串来描述、匹配一系列符合某个句法规则的字符串。

许多程序设计语言都支持利用正则表达式进行字符串操作。

MongoDB 使用 **\$regex** 操作符来设置匹配字符串的正则表达式。

MongoDB使用PCRE (Perl Compatible Regular Expression) 作为正则表达式语言。

不同于全文检索，我们使用正则表达式不需要做任何配置。

考虑以下 **posts** 集合的文档结构，该文档包含了文章内容和标签：

```
{

  "post_text": "enjoy the mongodb articles on runoob",

  "tags": [

    "mongodb",

    "runoob"

  ]

}
```

使用正则表达式

以下命令使用正则表达式查找包含 **runoob** 字符串的文章：

```
>db.posts.find({post_text:{$regex:"runoob"}})
```

以上查询也可以写为：

```
>db.posts.find({post_text:/runoob/})
```

不区分大小写的正则表达式

如果检索需要不区分大小写，我们可以设置 **\$options** 为 **\$i**。

以下命令将查找不区分大小写的字符串 **runoob**：

```
>db.posts.find({post_text:{$regex:"runoob",$options:"$i"}})
```

集合中会返回所有包含字符串 **runoob** 的数据，且不区分大小写：

```
{
  "_id" : ObjectId("53493d37d852429c10000004"),
  "post_text" : "hey! this is my post on runoob",
  "tags" : [ "runoob" ]
}
```

数组元素使用正则表达式

我们还可以在数组字段中使用正则表达式来查找内容。这在标签的实现上非常有用，如果你需要查找包含以 **run** 开头的标签数据(**ru** 或 **run** 或 **runoob**)，你可以使用以下代码：

```
>db.posts.find({tags:{$regex:"run"}})
```

优化正则表达式查询

如果你的文档中字段设置了索引，那么使用索引相比于正则表达式匹配查找所有的数据查询速度更快。

如果正则表达式是前缀表达式，所有匹配的数据将以指定的前缀字符串为开始。例如：如果正则表达式为 **^tut**，查询语句将查找以 **tut** 为开头的字符串。

这里面使用正则表达式有两点需要注意：

正则表达式中使用变量。一定要使用**eval**将组合的字符串进行转换，不能直接将字符串拼接后传入给表达式。否则没有报错信息，只是结果为空！实例如下：

```
var name=eval("/" + 变量值key +"/i");
```

以下是模糊查询包含**title**关键词, 且不区分大小写:

```
title:eval("/"+title+"/i")    // 等同于  title:{$regex:title,$option:"$i"}
```

MongoDB 全文检索

MongoDB 管理工具: Rockmongo



1 篇笔记
#1

写笔记



regex操作符的介绍

MongoDB使用\$regex操作符来设置匹配字符串的正则表达式, 使用PCRE(Pert Compatible Regular Expression)作为正则表达式语言。

regex操作符

```
{<field>:{$regex/pattern/, $options:'<options>'}}
```

```
{<field>:{$regex'pattern', $options:'<options>'}}
```

```
{<field>:{$regex/pattern/<options>'}}
```

正则表达式对象

```
{<field>:/pattern/<options>'}}
```

\$regex与正则表达式对象的区别:

在\$in操作符中只能使用正则表达式对象, 例如:{name:{\$in:['/joe/i','/jack/i]}}

在使用隐式的\$and操作符中, 只能使用\$regex, 例如:{name:{\$regex:/jo/i, \$nin:['john']}}

当option选项中包含X或S选项时, 只能使用\$regex. 例如:{name:{\$regex/m.*line/, \$options:"si"}}

\$regex操作符的使用

\$regex操作符中的option选项可以改变正则匹配的默认行为, 它包括i, m, x以及S四个选项, 其含义如下

i 忽略大小写, {<field>:{\$regex/pattern/i}}, 设置i选项后, 模式中的字母会进行大小写不敏感匹配。

m 多行匹配模式, {<field>:{\$regex/pattern/, \$options:'m'}}, m选项会更改^和\$元字符的默认行为, 分别使用与行的开头和结尾匹配, 而不是与输入字符串的开头和结尾匹配。

x 忽略非转义的空白字符, {<field>:{\$regex/pattern/, \$options:'m'}}, 设置x选项后, 正则表达式中的非转义的空白字符将被忽略, 同时井号(#)被解释为注释的开头注, 只能显式位于option选项中。

s 单行匹配模式{<field>:{\$regex/pattern/, \$options:'s'}}, 设置s选项后, 会改变模式中的点号(.)元字符的默认行为, 它会匹配所有字符, 包括换行符(\n), 只能显式位于option选项中。

使用\$regex操作符时, 需要注意下面几个问题:

i, m, x, s可以组合使用, 例如:{name:{\$regex/j*k/, \$options:"si"}}

在设置索引的字段上进行正则匹配可以提高查询速度, 而且当正则表达式使用的是前缀表达式时, 查询速度会进一步提高, 例如:{name:{\$regex /joe/}}

牛牛9个月前 (01-03)

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号: 闽ICP备15012807号-1



首页 HTML CSS JS 本地书签

MongoDB 正则表达式

MongoDB GridFS

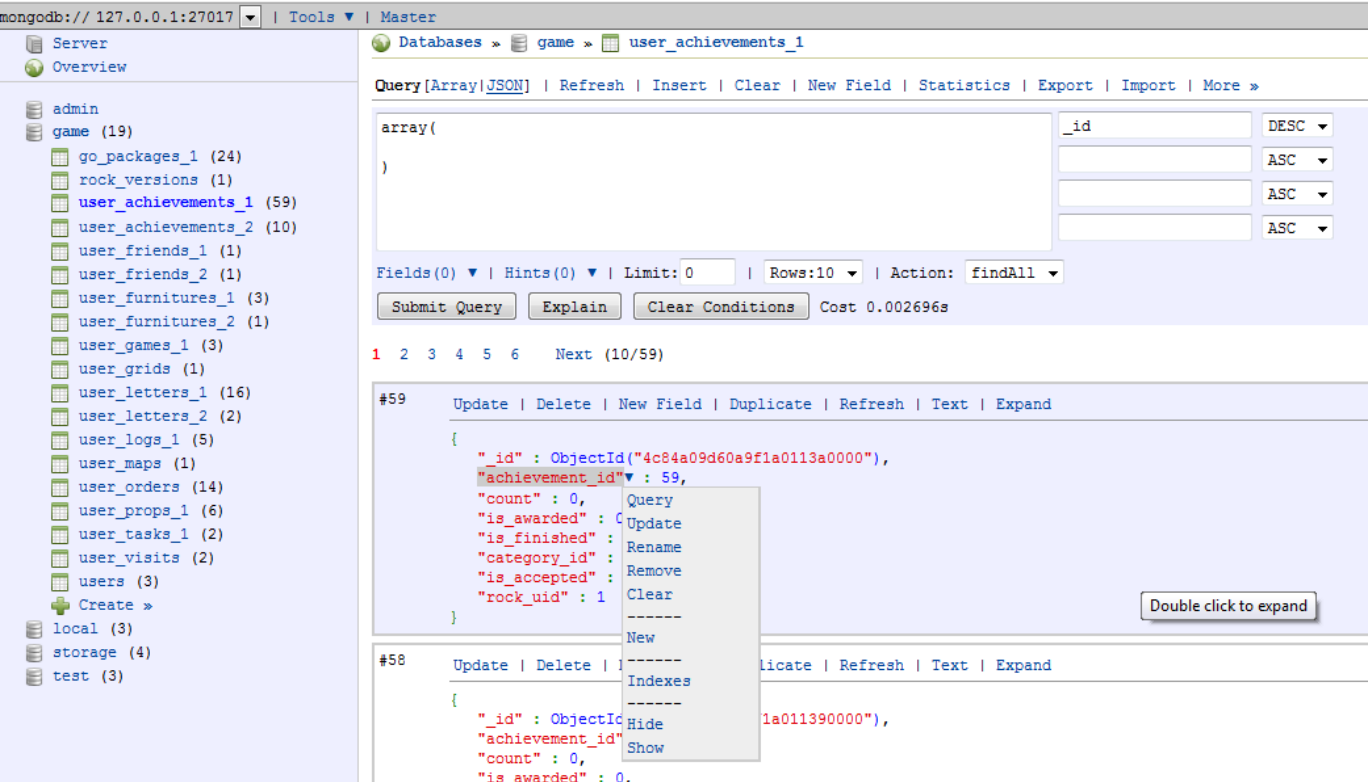
MongoDB 管理工具: Rockmongo

RockMongo是PHP5写的一个MongoDB管理工具。

通过 Rockmongo 你可以管理 MongoDB服务, 数据库, 集合, 文档, 索引等等。

它提供了非常人性化的操作。类似 phpMyAdmin (PHP开发的MySQL管理工具)。

Rockmongo 下载地址: <http://rockmongo.com/downloads>



简介

主要特征:

使用宽松的New BSD License协议

速度快，安装简单

支持多语言（目前提供中文、英文、日文、巴西葡萄牙语、法语、德语、俄语、意大利语）

系统

可以配置多个主机，每个主机可以有多个管理员

需要管理员密码才能登入操作，确保数据库的安全性

服务器

服务器信息 (WEB服务器, PHP, PHP.ini相关指令 ...)

状态

数据库信息

数据库

查询，创建和删除

执行命令和Javascript代码

统计信息

集合（相当于表）

强大的查询工具

读数据，写数据，更改数据，复制数据，删除数据

查询、创建和删除索引

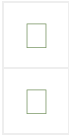
清空数据

批量删除和更改数据

统计信息

GridFS

查看分块



安装

需求

一个能运行PHP的Web服务器，比如Apache Httpd, Nginx ...

PHP - 需要PHP v5.1.6或更高版本，需要支持SESSION

为了能连接MongoDB，你需要安装php_mongo扩展

快速安装

下载安装包

解压到你的网站目录下

用编辑器打开config.php，修改host, port, admins等参数

在浏览器中访问index.php，比如说：<http://localhost/rockmongo/index.php>

使用用户名和密码登录，默认为"admin"和"admin"

开始玩转MongoDB!

参考文章：http://rockmongo.com/wiki/introduction?lang=zh_cn

MongoDB 正则表达式

MongoDB GridFS



1 篇笔记
#1

写笔记



Navicat for MongoDB也是一个MongoDB管理工具，是客户端，需要安装。
通过Navicat for MongoDB你同样可以管理 MongoDB服务，数据库，集合，文档，索引等等。
它提供了非常人性化的操作。类似 Navicat for MySQL（MySQL管理工具）。
了解Navicat for MongoDB：<https://www.navicat.com.cn/products/navicat-for-mongodb>
Navicat for MongoDB 下载地址：<https://www.navicat.com.cn/products>

吴鹏飞6天前

反馈/建议



MongoDB 管理工具: Rockmongo

MongoDB 固定集合（Capped Collections）

MongoDB GridFS

GridFS 用于存储和恢复那些超过16M（BSON文件限制）的文件(如：图片、音频、视频等)。

GridFS 也是文件存储的一种方式，但是它是存储在MongoDB的集合中。

GridFS 可以更好的存储大于16M的文件。

GridFS 会将大文件对象分割成多个小的chunk(文件片段),一般为256k/个,每个chunk将作为MongoDB的一个文档(document)被存储在chunks集合中。

GridFS 用两个集合来存储一个文件：`fs.files`与`fs.chunks`。

每个文件的实际内容被存在chunks(二进制数据)中,和文件有关的meta数据(filename,content_type,还有用户自定义的属性)将会被存在files集合中。

以下是简单的 **fs.files** 集合文档：

```
{

  "filename": "test.txt",

  "chunkSize": NumberInt(261120),

  "uploadDate": ISODate("2014-04-13T11:32:33.557Z"),

  "md5": "7b762939321e146569b07f72c62cca4f",

  "length": NumberInt(646)

}
```

以下是简单的 **fs.chunks** 集合文档：

```
{

  "files_id": ObjectId("534a75d19f54bfec8a2fe44b"),

  "n": NumberInt(0),

  "data": "Mongo Binary Data"

}
```

GridFS 添加文件

现在我们使用 **GridFS** 的 **put** 命令来存储 **mp3** 文件。调用 **MongoDB** 安装目录下bin的 **mongofiles.exe**工具。

打开命令提示符，进入到**MongoDB**的安装目录的bin目录中，找到**mongofiles.exe**，并输入下面的代码：

```
>mongofiles.exe -d gridfs put song.mp3
```

GridFS 是存储文件的数据名称。如果不存在该数据库，**MongoDB**会自动创建。**Song.mp3** 是音频文件名。

使用以下命令来查看数据库中文件的文档：

```
>db.fs.files.find()
```

以上命令执行后返回以下文档数据：

```
{

  _id: ObjectId('534a811bf8b4aa4d33fdf94d'),

  filename: "song.mp3",

  chunkSize: 261120,

  uploadDate: new Date(1397391643474), md5: "e4f53379c909f7bed2e9d631e15c1c41",
```



```
length: 10401959
```

```
}
```

我们可以看到 **fs.chunks** 集合中所有的区块，以下我们得到了文件的 **_id** 值，我们可以根据这个 **_id** 获取区块(chunk)的数据：

```
>db.fs.chunks.find({files_id:ObjectId('534a811bf8b4aa4d33fdf94d')})
```

以上实例中，查询返回了 **40** 个文档的数据，意味着**mp3**文件被存储在**40**个区块中。

[MongoDB 管理工具: Rockmongo](#)

[MongoDB 固定集合（Capped Collections）](#)

[点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[MongoDB GridFS](#)

[MongoDB 自动增长](#)

MongoDB 固定集合（Capped Collections）

MongoDB 固定集合（Capped Collections）是性能出色且有着固定大小的集合，对于大小固定，我们可以想象其就像一个环形队列，当集合空间用完后，再插入的元素就会覆盖最初部的元素！

创建固定集合

我们通过**createCollection**来创建一个固定集合，且**capped**选项设置为**true**：

```
>db.createCollection("cappedLogCollection",{capped:true,size:10000})
```

还可以指定文档个数,加上**max:1000**属性：

```
>db.createCollection("cappedLogCollection",{capped:true,size:10000,max:1000})
```

判断集合是否为固定集合：

```
>db.cappedLogCollection.isCapped()
```

如果需要将已存在的集合转换为固定集合可以使用以下命令：

```
>db.runCommand({"convertToCapped":"posts",size:10000})
```

以上代码将我们已存在的 `posts` 集合转换为固定集合。

固定集合查询

固定集合文档按照插入顺序储存的,默认情况下查询就是按照插入顺序返回的,也可以使用`$natural`调整返回顺序。

```
>db.cappedLogCollection.find().sort({$natural:-1})
```

固定集合的功能特点

可以插入及更新,但更新不能超出`collection`的大小,否则更新失败,不允许删除,但是可以调用`drop()`删除集合中的所有行,但是`drop`后需要显式地重建集合。

在32位机子上一个`capped collection`的最大值约为482.5M,64位上只受系统文件大小的限制。

固定集合属性及用法

属性

属性1:对固定集合进行插入速度极快

属性2:按照插入顺序的查询输出速度极快

属性3:能够在插入最新数据时,淘汰最早的数据

用法

用法1:储存日志信息

用法2:缓存一些少量的文档

MongoDB GridFS

MongoDB 自动增长



1 篇笔记
#1

写笔记

```
db.createCollection("cappedLogCollection",{capped:true,size:10000,max:1000})
```

size 是整个集合空间大小，单位为【KB】

max 是集合文档个数上线，单位是【个】

如果空间大小到达上限，则插入下一个文档时，会覆盖第一个文档；如果文档个数到达上限，同样插入下一个文档时，会覆盖第一个文档。两个参数上限判断取的是【与】的逻辑。

好汉张飞10个月前 (12-12)

反馈/建议

MongoDB 自动增长

MongoDB 没有像 SQL 一样有自动增长的功能，MongoDB 的 `_id` 是系统自动生成的12字节唯一标识。

但在某些情况下，我们可能需要实现 `ObjectId` 自动增长功能。

由于 MongoDB 没有实现这个功能，我们可以通过编程的方式来实现，以下我们将在 `counters` 集合中实现 `_id` 字段自动增长。

使用 `counters` 集合

考虑以下 `products` 文档。我们希望 `_id` 字段实现 从 1,2,3,4 到 n 的自动增长功能。

```
{
  "_id":1,
  "product_name": "Apple iPhone",
  "category": "mobiles"
}
```

为此，创建 `counters` 集合，序列字段值可以实现自动长：

```
>db.createCollection("counters")
```

现在我们向 `counters` 集合中插入以下文档，使用 `productid` 作为 `key`:

```
{
  "_id":"productid",
  "sequence_value": 0
}
```

`sequence_value` 字段是序列通过自动增长后的一个值。

使用以下命令插入 `counters` 集合的序列文档中：

```
>db.counters.insert({_id:"productid",sequence_value:0})
```

创建 Javascript 函数

现在，我们创建函数 `getNextSequenceValue` 来作为序列名的输入，指定的序列会自动增长 1 并返回最新序列值。在本文的实例中序列名为 `productid`。

```
>function getNextSequenceValue(sequenceName){

    var sequenceDocument = db.counters.findAndModify(

        {

            query:{_id: sequenceName },

            update: {$inc:{sequence_value:1}},

            "new":true

        });

    return sequenceDocument.sequence_value;

}
```

使用 Javascript 函数

接下来我们将使用 `getNextSequenceValue` 函数创建一个新的文档，并设置文档 `_id` 自动为返回的序列值：

```
>db.products.insert({

    "_id":getNextSequenceValue("productid"),

    "product_name":"Apple iPhone",

    "category":"mobiles"})

>db.products.insert({

    "_id":getNextSequenceValue("productid"),

    "product_name":"Samsung S3",

    "category":"mobiles"})
```

就如你所看到的，我们使用 `getNextSequenceValue` 函数来设置 `_id` 字段。

为了验证函数是否有效，我们可以使用以下命令读取文档：

```
>db.products.find()
```

以上命令将返回以下结果，我们发现 `_id` 字段是自增长的：

```
{ "_id" : 1, "product_name" : "Apple iPhone", "category" : "mobiles" }

{ "_id" : 2, "product_name" : "Samsung S3", "category" : "mobiles" }
```

[MongoDB 固定集合（Capped Collections）](#)

[MongoDB 创建数据库](#)

[点我分享笔记](#)

[反馈/建议](#)