**MangoDB**

**学习手册**

**V1.0**

**张镇**

**目 录**

[1 简介 1](#_Toc490472466)

[1.1 MongoDB的重要特性 1](#_Toc490472467)

[1.2 应该在哪些场景使用MongoDB 2](#_Toc490472468)

[1.3 MongoDB和RDBMS的性能分析 2](#_Toc490472469)

[2 安装MangoDB 3](#_Toc490472470)

[2.1 Windows下安装配置MangoDB 3](#_Toc490472471)

[2.1.1 前提条件 3](#_Toc490472472)

[2.1.2 MongoDB安装 4](#_Toc490472473)

[2.1.3 运行MongoDB 7](#_Toc490472474)

[2.1.4 启动MongoDB 7](#_Toc490472475)

[2.1.5 MongoDB帮助 9](#_Toc490472476)

[2.1.6 MongoDB统计 11](#_Toc490472477)

[2.2 Ubuntu下安装配置MangoDB 11](#_Toc490472478)

[2.2.1 安装MongoDB社区版 11](#_Toc490472479)

[2.2.1.1 第一步：导入包管理系统使用的公钥 12](#_Toc490472480)

[2.2.1.2 第二步：为MongoDB创建一个列表文件 12](#_Toc490472481)

[2.2.1.3 第三步：重新加载本地包数据库 13](#_Toc490472482)

[2.2.1.4 第四步：安装MongoDB包 13](#_Toc490472483)

[2.2.2 运行MongoDB 13](#_Toc490472484)

[2.2.3 启动 MongoDB 13](#_Toc490472485)

[2.2.4 停止MongoDB 15](#_Toc490472486)

[2.2.5 重新启动MongoDB 15](#_Toc490472487)

[2.3 RedHat/CentOS下安装配置MangoDB 15](#_Toc490472488)

[2.3.1 安装MongoDB社区版 15](#_Toc490472489)

[2.3.2 运行MongoDB 17](#_Toc490472490)

[2.3.3 启动 MongoDB 17](#_Toc490472491)

[2.3.4 停止 MongoDB 18](#_Toc490472492)

[2.3.5 重启 MongoDB 19](#_Toc490472493)

[2.3.6 开始使用 MongoDB 19](#_Toc490472494)

[3 MangoDB知识 21](#_Toc490472495)

[3.1 MangoDB基本概念 21](#_Toc490472496)

[3.1.1 数据库 21](#_Toc490472497)

[3.1.2 集合 21](#_Toc490472498)

[3.1.3 文档 21](#_Toc490472499)

[3.1.4 示例 21](#_Toc490472500)

[3.2 MangoDB数据类型 22](#_Toc490472501)

[3.3 MangoDB数据建模 22](#_Toc490472502)

[4 各语言连接MangoDB 25](#_Toc490472503)

[4.1 Java连接MangoDB 25](#_Toc490472504)

[4.1.1 安装 MongoDB JDBC驱动程序 25](#_Toc490472505)

[4.1.2 连接到数据库 25](#_Toc490472506)

[4.1.3 创建和列出集合 27](#_Toc490472507)

[4.1.4 插入文档 28](#_Toc490472508)

[4.1.5 更新文档 31](#_Toc490472509)

[4.1.6 删除文档 32](#_Toc490472510)

[4.2 Python连接MangoDB 34](#_Toc490472511)

[4.2.1 安装PyMongo 34](#_Toc490472512)

[4.2.2 使用MongoClient建立连接 35](#_Toc490472513)

[4.2.3 获取数据库 35](#_Toc490472514)

[4.2.4 获取集合 35](#_Toc490472515)

[4.2.5 文档 36](#_Toc490472516)

[4.2.6 插入文档 36](#_Toc490472517)

[4.2.7 使用find\_one()获取单个文档 37](#_Toc490472518)

[4.2.8 通过ObjectId查询 39](#_Toc490472519)

[4.2.9 关于Unicode字符串的注释 40](#_Toc490472520)

[4.2.10 批量插入 40](#_Toc490472521)

[4.2.11 查询多个文档 41](#_Toc490472522)

[4.2.12 计数统计 43](#_Toc490472523)

[4.2.13 范围查询 43](#_Toc490472524)

[4.2.14 索引 44](#_Toc490472525)

[4.3 PHP连接MangoDB 44](#_Toc490472526)

[4.3.1 进行连接并选择数据库 45](#_Toc490472527)

[4.3.2 创建一个集合 45](#_Toc490472528)

[4.3.3 插入文档 46](#_Toc490472529)

[4.3.4 查找所有文件 47](#_Toc490472530)

[4.3.5 更新文档 47](#_Toc490472531)

[4.3.6 删除文档 48](#_Toc490472532)

[4.4 Ruby连接MangoDB 49](#_Toc490472533)

[4.4.1 安装驱动程序 49](#_Toc490472534)

[4.4.2 前提条件 49](#_Toc490472535)

[4.4.3 Ruby连接MongoDB 50](#_Toc490472536)

[4.4.4 访问数据库和集合 50](#_Toc490472537)

[4.4.4.1 插入文档 50](#_Toc490472538)

[4.4.4.2 查询集合 51](#_Toc490472539)

[4.4.4.3 更新文件 51](#_Toc490472540)

[4.4.4.4 删除文档 52](#_Toc490472541)

[4.4.4.5 创建索引 52](#_Toc490472542)

[5 使用MangoDB 53](#_Toc490472543)

[5.1 创建数据库：use 命令 53](#_Toc490472544)

[5.2 删除数据库：db.dropDatabase() 方法 53](#_Toc490472545)

[5.3 创建集合：createCollection()方法 54](#_Toc490472546)

[5.4 删除集合：drop()方法 55](#_Toc490472547)

[5.5 插入文档：insert()方法、save()方法 56](#_Toc490472548)

[5.6 其它插入文档的方法 58](#_Toc490472549)

[5.6.1 db.collection.insertOne()方法 58](#_Toc490472550)

[5.6.2 db.collection.insertMany()方法 58](#_Toc490472551)

[5.7 查询文档 59](#_Toc490472552)

[5.7.1 find()方法 59](#_Toc490472553)

[5.7.2 pretty()方法 59](#_Toc490472554)

[5.7.3 MongoDB 与 RDBMS的等效 Where 子句 60](#_Toc490472555)

[5.7.3.1 MongoDB中的AND操作符 60](#_Toc490472556)

[5.7.3.2 MongoDB中的OR操作符 61](#_Toc490472557)

[5.7.3.3 使用 AND 和 OR 条件一起 61](#_Toc490472558)

[5.7.3.4 查询嵌入/嵌套文档 62](#_Toc490472559)

[5.7.3.4.1 匹配嵌入/嵌套文档 62](#_Toc490472560)

[5.7.3.4.2 查询嵌套字段 63](#_Toc490472561)

[5.7.3.4.3 使用查询运算符指定匹配 63](#_Toc490472562)

[5.7.3.4.4 指定AND条件 63](#_Toc490472563)

[5.8 更新文档 63](#_Toc490472564)

[5.8.1 MongoDB Update()方法 63](#_Toc490472565)

[5.8.2 MongoDB Save()方法 64](#_Toc490472566)

[5.9 删除文档：remove()方法 64](#_Toc490472567)

[5.9.1 只删除一条文档记录 65](#_Toc490472568)

[5.9.2 删除所有文档记录 65](#_Toc490472569)

[5.10 投影（选择字段）：find()方法 66](#_Toc490472570)

[5.11 限制记录数 67](#_Toc490472571)

[5.11.1 MongoDB limit()方法 67](#_Toc490472572)

[5.11.2 MongoDB skip()方法 67](#_Toc490472573)

[5.12 排序文档 68](#_Toc490472574)

[5.12.1 MongoDB sort()方法 68](#_Toc490472575)

[6 MangoDB进阶 70](#_Toc490472576)

[6.1 MongoDB索引 70](#_Toc490472577)

[6.1.1 ensureIndex()方法 70](#_Toc490472578)

[6.2 MongoDB聚合 71](#_Toc490472579)

[6.2.1 aggregate()方法 71](#_Toc490472580)

[6.2.2 管道概念 73](#_Toc490472581)

[6.3 MongoDB复制 74](#_Toc490472582)

[6.3.1 为什么复制？ 74](#_Toc490472583)

[6.3.2 MongoDB复制的工作原理 74](#_Toc490472584)

[6.3.3 副本集功能 75](#_Toc490472585)

[6.3.4 设置副本集 75](#_Toc490472586)

[6.3.5 将会员添加到副本集 76](#_Toc490472587)

[6.4 MongoDB分片 77](#_Toc490472588)

[6.4.1 为什么要分片？ 77](#_Toc490472589)

[6.4.2 MongoDB中的分片 77](#_Toc490472590)

[6.5 MongoDB备份与恢复 78](#_Toc490472591)

[6.5.1 导出转储MongoDB数据 78](#_Toc490472592)

[6.5.2 恢复数据 81](#_Toc490472593)

[6.6 MongoDB部署 81](#_Toc490472594)

[7 MangoDB高级应用 84](#_Toc490472595)

[7.1 MongoDB关联关系 84](#_Toc490472596)

[7.1.1 嵌入式关系建模 84](#_Toc490472597)

[7.1.2 建模参考关系 85](#_Toc490472598)

[7.2 MongoDB数据库引用 85](#_Toc490472599)

[7.2.1 DBRefs vs 手动参考 86](#_Toc490472600)

[7.2.2 使用DBRefs 86](#_Toc490472601)

[7.3 MongoDB覆盖查询 87](#_Toc490472602)

[7.3.1 什么是覆盖查询？ 87](#_Toc490472603)

[7.3.2 使用覆盖查询 87](#_Toc490472604)

[7.4 MongoDB分析查询 88](#_Toc490472605)

[7.4.1 使用 $explain 操作符 88](#_Toc490472606)

[7.4.2 使用 $hint 89](#_Toc490472607)

[7.5 MongoDB原子操作 89](#_Toc490472608)

[7.5.1 原子操作模型数据 89](#_Toc490472609)

[7.6 MongoDB高级索引 91](#_Toc490472610)

[7.6.1 索引数组字段 91](#_Toc490472611)

[7.6.2 索引子文档字段 91](#_Toc490472612)

[7.7 MongoDB索引限制 92](#_Toc490472613)

[7.7.1 索引额外开销 92](#_Toc490472614)

[7.7.2 内存的使用 92](#_Toc490472615)

[7.7.3 查询限制 92](#_Toc490472616)

[7.7.4 索引键限制 92](#_Toc490472617)

[7.7.5 插入超过索引键限制的文档 93](#_Toc490472618)

[7.7.6 索引最大范围 93](#_Toc490472619)

[7.8 MongoDB对象ID（ObjectId） 93](#_Toc490472620)

[7.8.1 创建文档的时间戳 93](#_Toc490472621)

[7.8.2 将ObjectId转换为String 94](#_Toc490472622)

[7.9 MongoDB Map Reduce 94](#_Toc490472623)

[7.9.1 MapReduce命令 94](#_Toc490472624)

[7.9.2 使用MapReduce 95](#_Toc490472625)

[7.10 MongoDB文本搜索 96](#_Toc490472626)

[7.10.1 启用文本搜索 96](#_Toc490472627)

[7.10.2 创建文本索引 96](#_Toc490472628)

[7.10.3 使用文本索引 96](#_Toc490472629)

[7.10.4 删除文本索引 97](#_Toc490472630)

[7.11 MongoDB正则表达式 97](#_Toc490472631)

[7.11.1 使用正则表达式 98](#_Toc490472632)

[7.11.2 使用不区分大小写的正则表达式 98](#_Toc490472633)

[7.11.3 对数组元素使用正则表达式 98](#_Toc490472634)

[7.11.4 化正则表达式查询 98](#_Toc490472635)

[7.12 MongoDB GridFS 99](#_Toc490472636)

[7.12.1 将文件添加到GridFS 99](#_Toc490472637)

[7.13 MongoDB固定循环集合 100](#_Toc490472638)

[7.13.1 创建固定集合 100](#_Toc490472639)

[7.13.2 查询固定集合 101](#_Toc490472640)

[7.14 MongoDB自动递增序列 101](#_Toc490472641)

[7.14.1 使用计数器集合 101](#_Toc490472642)

[7.14.2 使用Javascript函数 102](#_Toc490472643)

[8 用户及安全 103](#_Toc490472644)

[8.1 MongoDB安全检查表 103](#_Toc490472645)

[8.1.1 启用访问控制和强制验证 103](#_Toc490472646)

[8.1.2 配置基于角色的访问控制 103](#_Toc490472647)

[8.1.3 加密通信 103](#_Toc490472648)

[8.1.4 加密和保护数据 103](#_Toc490472649)

[8.1.5 限制网络曝光 104](#_Toc490472650)

[8.1.6 审计系统活动 104](#_Toc490472651)

[8.1.7 使用专用用户运行MongoDB 104](#_Toc490472652)

[8.1.8 使用安全配置选项运行MongoDB 104](#_Toc490472653)

[8.1.9 请求安全技术实施指南(如适用) 104](#_Toc490472654)

[8.2 MongoDB认证 105](#_Toc490472655)

[8.2.1 认证方法 105](#_Toc490472656)

[8.2.2 认证机制 105](#_Toc490472657)

[8.2.3 内部认证 105](#_Toc490472658)

[8.2.4 分片集群的认证 106](#_Toc490472659)

[8.3 MongoDB用户 106](#_Toc490472660)

[8.3.1 用户管理接口 106](#_Toc490472661)

[8.3.2 验证数据库 106](#_Toc490472662)

[8.3.3 认证用户 106](#_Toc490472663)

[8.3.4 分片集群用户 107](#_Toc490472664)

[8.4 MongoDB添加用户 107](#_Toc490472665)

[8.4.1 前提条件 108](#_Toc490472666)

[8.4.2 例子 108](#_Toc490472667)

[8.4.2.1 创建管理员用户 108](#_Toc490472668)

[8.4.2.2 添加数据库用户 109](#_Toc490472669)

[8.4.2.3 删除数据库用户 110](#_Toc490472670)

[8.4.2.4 创建带角色的用户 110](#_Toc490472671)

[8.4.2.5 创建没有角色的用户 110](#_Toc490472672)

[8.4.2.6 创建具有角色的管理用户 111](#_Toc490472673)

[8.5 MongoDB启用身份验证 111](#_Toc490472674)

[8.5.1 用户管理员 111](#_Toc490472675)

[8.5.2 过程 112](#_Toc490472676)

[8.5.2.1 第一步：启动MongoDB无需访问控制 112](#_Toc490472677)

[8.5.2.2 第二步：连接到实例 112](#_Toc490472678)

[8.5.2.3 第三步：创建用户管理员 112](#_Toc490472679)

[8.5.2.4 第四步：重新启动具有访问控制的MongoDB实例 113](#_Toc490472680)

[8.5.2.5 第五步：以用户管理员身份进行连接和验证 113](#_Toc490472681)

[8.5.2.6 第六步：根据需要创建其他用户 113](#_Toc490472682)

[8.5.2.7 第七步：连接并验证为myTester 114](#_Toc490472683)

[8.6 MongoDB管理用户和角色 115](#_Toc490472684)

[8.6.1 创建用户定义的角色 115](#_Toc490472685)

[8.6.2 创建角色来管理当前操作 115](#_Toc490472686)

[8.6.3 创建角色用来运行 mongostat 116](#_Toc490472687)

[8.6.4 修改现有用户的访问权限 117](#_Toc490472688)

[8.6.5 修改现有用户的密码 118](#_Toc490472689)

[8.6.6 查看用户的角色 118](#_Toc490472690)

[8.6.7 查看角色的权限 119](#_Toc490472691)

[8.7 MongoDB更改用户密码和自定义数据 120](#_Toc490472692)

[8.7.1 注意事项 120](#_Toc490472693)

[8.7.2 先决条件 120](#_Toc490472694)

[8.7.3 执行过程 121](#_Toc490472695)

# 简介

MongoDB是一个NoSQL数据库。 它是一个使用C++编写的开源，跨平台，面向文档的数据库。

MongoDB的官方网站是：<http://www.mongodb.com/> ，可以从官方上找到大部分有关数据库的相关资料，如：各种版本的安装包下载，文档，最新的 MongoDB 资讯，社区以及教程等等。

首先应该知道什么是面向文档的数据库？MongoDB是面向文档的数据库，这是MongoDB的主要功能，它提供面向文档的存储。MongoDB将数据存储为文档，因此被称为面向文档的数据库，如下例。

FirstName = "Max",

Address = "Haikou City",

Spouse = [{Name: "Maxsu"}].

FirstName ="Kobe",

Address = "LAC"

## MongoDB的重要特性

下面列出的是MongoDB的一些重要功能特性：

1.支持特别查询：在MongoDB中，可以通过字段、范围查询进行搜索，并且还支持正则表达式搜索。

2.索引：可以索引文档中的任何字段。

3.复制：MongoDB支持主从复制。主机可以执行读写操作，从机从主机复制数据，只能用于读取或备份(不写入)。

4.复制数据：MongoDB可以在多台服务器上运行。复制数据以保持系统正常运行，并在硬件故障的情况下保持其运行状态。

5.负载均衡：由于数据放在碎片中，因此具有自动负载均衡配置。

6.支持映射缩减和聚合工具。

7.使用JavaScript而不是Procedure。

8.它是一个用C++编写的无模式数据库。

9.提供高性能的数据库服务。

10.轻松存储任何大小的文件，而不会使您的堆栈复杂化。

11.在故障的情况下易于管理。

12.它还支持：

* 具有动态模式的JSON数据模型。
* 自动分片，用于水平扩展。
* 内置复制，提供高可用性。

## 应该在哪些场景使用MongoDB

* 数据大而复杂。
* 移动和社会基础设施数据。
* 内容管理和交付。
* 用户数据管理。
* 数据中心。

## MongoDB和RDBMS的性能分析

* 在关系数据库(RDBMS)中，表用作存储元素，而在 MongoDB 中使用的是集合。
* 在RDBMS中有多个模式，在每个模式中，可创建用于存储数据的表，而 MongoDB 是面向文档的数据库，数据是以类似[JSON](http://www.yiibai.com/json/)格式的BSON格式编写的存储的。
* MongoDB几乎比传统数据库系统快100倍。

# 安装MangoDB

## Windows下安装配置MangoDB

要在Windows上安装 MongoDB，首先打开： <http://www.mongodb.org/downloads> 下载最新版本的MongoDB。确保根据您的Windows版本获得正确版本的MongoDB。

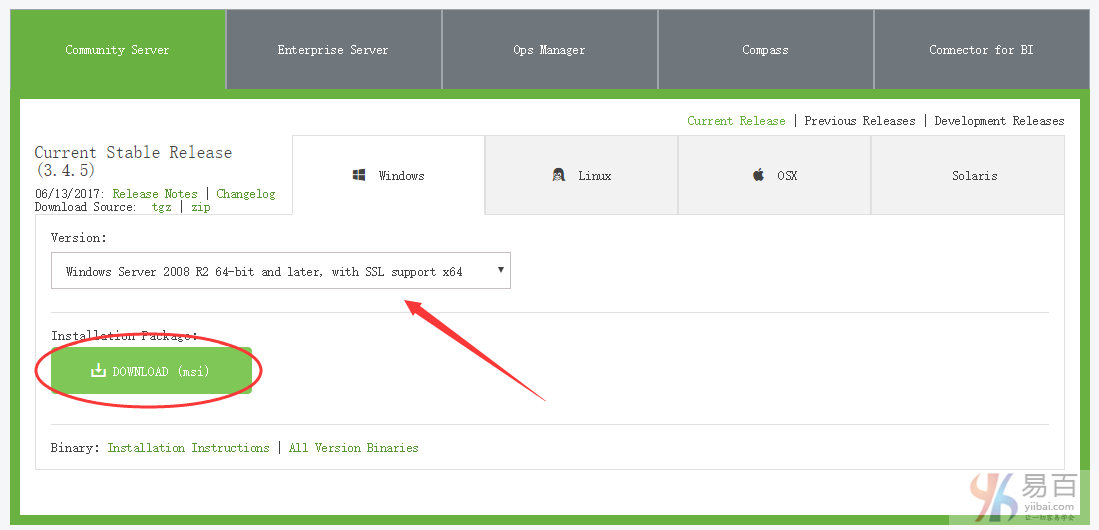
要获取 Windows 版本，请打开命令提示符并执行以下命令。

C:\Users\Administrator> wmic os get osarchitecture

OSArchitecture

64 位

如上所示，这里 Windows 版本是 64位的操作系统，所以下载对应的 MongoDB-for-Windows 64位版本，如下图所示 -

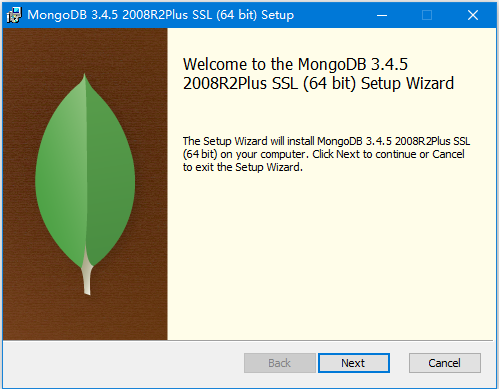


### 前提条件

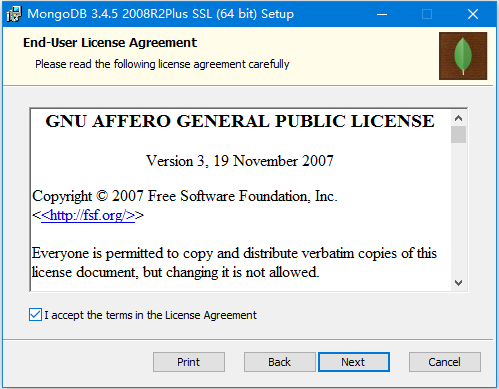
MongoDB社区版需要Windows Server 2008 R2，Windows Vista或更高版本(注：本教程将演示在Windows10 46位系统上安装MongoDB)。 .msi安装程序包括所有其他软件相关性，并将自动升级任何使用.msi文件安装的旧版本的 MongoDB。

### MongoDB安装

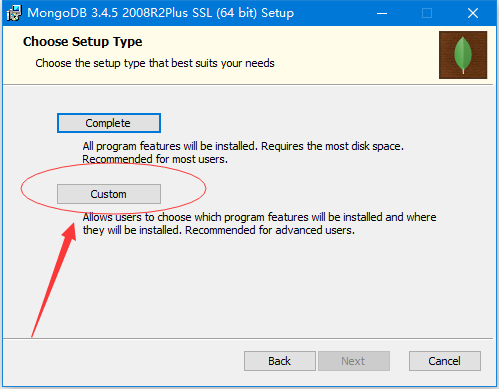
在Windows资源管理器中，找到下载的MongoDB.msi文件，通常位于默认的“下载”文件夹中。 双击mongodb-win32-x86\_64-2008plus-ssl-3.4.5-signed.msi文件。 将出现一组屏幕，指导您完成安装过程。



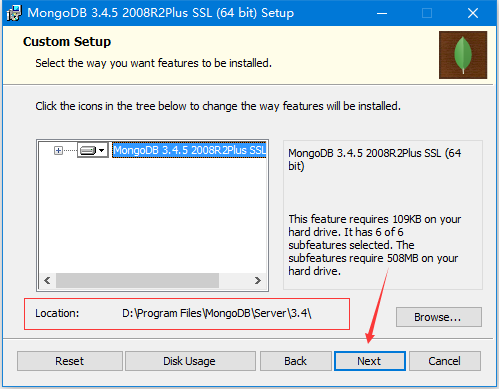
下一步(*Next>*)，在弹出的界面中，打上勾以表示“同意安装协议” -



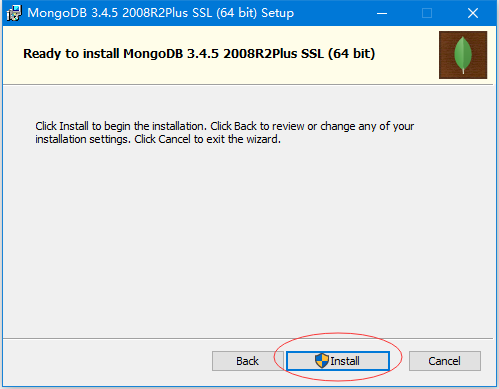
下一步(*Next>*)，选择自定义(*Custom*)安装，如下图所示 -



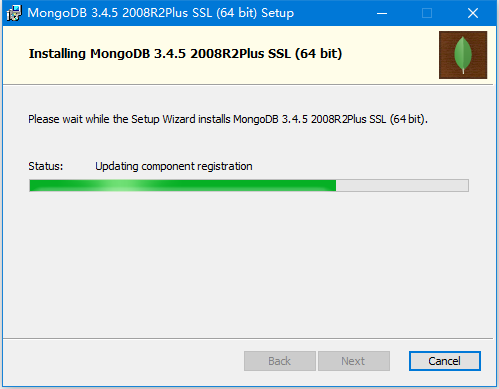
下一步(*Next>*)，选择安装目录为：*D:\Program Files\MongoDB\Server\3.4\* ，如下图所示 -



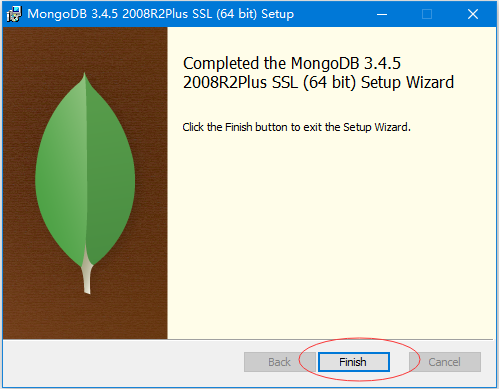
下一步(*Next>*)，开始安装 -



安装过程中，如下图所示 -



安装完成，点击：*Finish*，如下图所示 -



### 运行MongoDB

注意：不要让未使用“安全模式”进行认证的 mongod.exe 在公共网络上可见！！这样是非常危险的！！MongoDB一般在受信任的环境中运行，默认情况下数据库不启用“安全模式”。

MongoDB需要一个数据目录来存储所有数据。MongoDB的默认数据目录路径是启动 MongoDB 的驱动器上的绝对路径，例如D:\Program Files\MongoDB\Server\3.4\data\db。通过在命令提示符中运行以下命令来创建此文件夹：

D:\Program Files\MongoDB\Server\3.4> "md D:\Program Files\MongoDB\Server\3.4\data"

可以使用--dbpath选项为mongod.exe指定数据文件的备用路径，例如：

“D:\Program Files\MongoDB\Server\3.4\bin\mongod.exe" --dbpath "D:\Program Files\MongoDB\Server\3.4\data\db"

### 启动MongoDB

要启动 *MongoDB*，请运行 mongod.exe 。 例如，从命令提示符：

“D:\Program Files\MongoDB\Server\3.4\bin\mongod.exe"

注意：如果不使用 --dbpath 指定数据存储的目录，那么 MongoDB 默认使用的是 “C:\data\db“ 目录，所以在启动 MongoDB 数据库之前要确保 “C:\data\db“ 目录已经创建了。

在执行上面命令启动后，应该会看到类似下面的输出结果：

C:\Users\Administrator>"D:\Program Files\MongoDB\Server\3.4\bin\mongod.exe"

2017-06-29T02:37:46.688+0800 I CONTROL [initandlisten] MongoDB starting : pid=10652 port=27017 dbpath=C:\data\db\ 64-bit host=MY-PC

2017-06-29T02:37:46.691+0800 I CONTROL [initandlisten] targetMinOS: Windows

... ...

... ...

2017-06-29T02:37:46.733+0800 I STORAGE [initandlisten] wiredtiger\_open config: create,cache\_size=1491M,session\_max=20000,eviction=(threads\_min=4,threads\_max=4),config\_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file\_manager=(close\_idle\_time=100000),checkpoint=(wait=60,log\_size=2GB),statistics\_log=(wait=0),

2017-06-29T02:37:47.738+0800 I CONTROL [initandlisten]

2017-06-29T02:37:47.742+0800 I CONTROL [initandlisten] \*\* WARNING: Access control is not enabled for the database.

2017-06-29T02:37:47.751+0800 I CONTROL [initandlisten] \*\* Read and write access to data and configuration is unrestricted.

2017-06-29T02:37:47.761+0800 I CONTROL [initandlisten]

2017-06-29T02:37:48.315+0800 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory 'C:/data/db/diagnostic.data'

2017-06-29T02:37:48.659+0800 I INDEX [initandlisten] build index on: admin.system.version properties: { v: 2, key: { version: 1 }, name: "incompatible\_with\_version\_32", ns: "admin.system.version" }

2017-06-29T02:37:48.669+0800 I INDEX [initandlisten] building index using bulk method; build may temporarily use up to 500 megabytes of RAM

2017-06-29T02:37:48.710+0800 I INDEX [initandlisten] build index done. scanned 0 total records. 0 secs

2017-06-29T02:37:48.718+0800 I COMMAND [initandlisten] setting featureCompatibilityVersion to 3.4

2017-06-29T02:37:48.732+0800 I NETWORK [thread1] waiting for connections on port 27017

2017-06-29T02:40:54.820+0800 I NETWORK [thread1] connection accepted from 127.0.0.1:60285 #1 (1 connection now open)

2017-06-29T02:40:54.822+0800 I NETWORK [conn1] received client metadata from 127.0.0.1:60285 conn1: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", version: "3.4.5" }, os: { type: "Windows", name: "Microsoft Windows 8", architecture: "x86\_64", version: "6.2 (build 9200)" } }

打开 MongoDB 客户端测试使用：

D:\Program Files\MongoDB\Server\3.4\bin> mongo.exe

MongoDB shell version v3.4.5

connecting to: mongodb://127.0.0.1:27017

MongoDB server version: 3.4.5

Welcome to the MongoDB shell.

For interactive help, type "help".

For more comprehensive documentation, see

http://docs.mongodb.org/

Questions? Try the support group

http://groups.google.com/group/mongodb-user

Server has startup warnings:

2017-06-29T02:37:47.738+0800 I CONTROL [initandlisten]

2017-06-29T02:37:47.742+0800 I CONTROL [initandlisten] \*\* WARNING: Access control is not enabled for the database.

2017-06-29T02:37:47.751+0800 I CONTROL [initandlisten] \*\* Read and write access to data and configuration is unrestricted.

2017-06-29T02:37:47.761+0800 I CONTROL [initandlisten]

>

> db

test

> show dbs

admin 0.000GB

local 0.000GB

>

### MongoDB帮助

要获取命令列表，请在MongoDB客户端中键入：db.help()。 这将列出一个命令列表，如下所示：

> db.help()

DB methods:

db.adminCommand(nameOrDocument) - switches to 'admin' db, and runs command [ just calls db.runCommand(...) ]

db.auth(username, password)

db.cloneDatabase(fromhost)

db.commandHelp(name) returns the help for the command

db.copyDatabase(fromdb, todb, fromhost)

db.createCollection(name, { size : ..., capped : ..., max : ... } )

db.createView(name, viewOn, [ { $operator: {...}}, ... ], { viewOptions } )

db.createUser(userDocument)

db.currentOp() displays currently executing operations in the db

db.dropDatabase()

db.eval() - deprecated

db.fsyncLock() flush data to disk and lock server for backups

db.fsyncUnlock() unlocks server following a db.fsyncLock()

db.getCollection(cname) same as db['cname'] or db.cname

db.getCollectionInfos([filter]) - returns a list that contains the names and options of the db's collections

db.getCollectionNames()

db.getLastError() - just returns the err msg string

db.getLastErrorObj() - return full status object

db.getLogComponents()

db.getMongo() get the server connection object

db.getMongo().setSlaveOk() allow queries on a replication slave server

db.getName()

db.getPrevError()

db.getProfilingLevel() - deprecated

db.getProfilingStatus() - returns if profiling is on and slow threshold

db.getReplicationInfo()

db.getSiblingDB(name) get the db at the same server as this one

db.getWriteConcern() - returns the write concern used for any operations on this db, inherited from server object if set

db.hostInfo() get details about the server's host

db.isMaster() check replica primary status

db.killOp(opid) kills the current operation in the db

db.listCommands() lists all the db commands

db.loadServerScripts() loads all the scripts in db.system.js

db.logout()

db.printCollectionStats()

db.printReplicationInfo()

db.printShardingStatus()

db.printSlaveReplicationInfo()

db.dropUser(username)

db.repairDatabase()

db.resetError()

db.runCommand(cmdObj) run a database command. if cmdObj is a string, turns it into { cmdObj : 1 }

db.serverStatus()

db.setLogLevel(level,<component>)

db.setProfilingLevel(level,<slowms>) 0=off 1=slow 2=all

db.setWriteConcern( <write concern doc> ) - sets the write concern for writes to the db

db.unsetWriteConcern( <write concern doc> ) - unsets the write concern for writes to the db

db.setVerboseShell(flag) display extra information in shell output

db.shutdownServer()

db.stats()

db.version() current version of the server

>

### MongoDB统计

要获取有关MongoDB服务器的统计信息，请在 MongoDB 客户端中键入命令：db.stats()。 这个命令将显示数据库名称，数据库中的集合和文档数量。命令的输出如下所示：

> db.stats()

{

"db" : "test",

"collections" : 0,

"views" : 0,

"objects" : 0,

"avgObjSize" : 0,

"dataSize" : 0,

"storageSize" : 0,

"numExtents" : 0,

"indexes" : 0,

"indexSize" : 0,

"fileSize" : 0,

"ok" : 1

}

>

## Ubuntu下安装配置MangoDB

本节将介绍和演示如何在.deb包上的LTS Ubuntu Linux系统上安装 MongoDB 社区版。 虽然 Ubuntu 包含自己的 MongoDB 软件包，但官方 MongoDB 社区版软件包通常是最新版本的。

*mongodb-org-server* 包提供了一个使用 /etc/mongod.conf 配置文件启动 mongod 的初始化脚本。

这些包与Ubuntu提供的 *mongodb*，*mongodb-server* 和 *mongodb-clients* 包冲突。由软件包提供的默认的 */etc/mongod.conf* 配置文件的 bind\_ip 默认设置为 *127.0.0.1* 。 在初始化副本集之前，根据环境的需要修改此设置。

MongoDB仅提供 64位LTS(长期支持)Ubuntu版本的软件包。 例如，*12.04 LTS*，*14.04 LTS*，*16.04 LTS(xenial)*等。这些软件包可能与其他 Ubuntu 版本一起使用，但是它们不受支持。

### 安装MongoDB社区版

下面我们来看看在 Ubuntu(16.04 LTS) 上安装 MongoDB 的步骤：

#### 第一步：导入包管理系统使用的公钥

Ubuntu软件包管理工具(即dpkg和apt)通过要求分销商使用GPG密钥对软件包进行签名来确保软件包的一致性和真实性。发出以下命令导入[MongoDB公共GPG密钥](http://www.mongodb.org/static/pgp/server-3.4.asc?_ga=2.211916818.749835483.1498599570-1870842856.1498599569)：

$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 0C49F3730359A14518585931BC711F9BA15703C6

#### 第二步：为MongoDB创建一个列表文件

首先要确定所使用的Ubuntu系统的版本，要查看当前的内核版本，请打开终端并键入uname -r命令以查看内核版本：

查看内核版本命令如下：

yiibai@ubuntu:~$ uname -r

4.4.0-31-generic

yiibai@ubuntu:~$

查看操作系统是32位还是64位：

yiibai@ubuntu:~$ sudo uname --m

[sudo] password for yiibai:

x86\_64

yiibai@ubuntu:~$

查看 Ubuntu 版本 -

yiibai@ubuntu:~$ sudo lsb\_release -a

No LSB modules are available.

Distributor ID: Ubuntu

Description: Ubuntu 16.04.1 LTS

Release: 16.04

Codename: xenial

yiibai@ubuntu:~$

使用适合您的Ubuntu版本的命令创建 /etc/apt/sources.list.d/mongodb-org-3.4.list 列表文件：

***Ubuntu 14.04***

$ echo "deb [ arch=amd64 ] http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list

***Ubuntu 16.04***

$ echo "deb [ arch=amd64,arm64 ] http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list

#### 第三步：重新加载本地包数据库

发出以下命令重新加载本地软件包数据库：

$ sudo apt-get update

注意：上面的安装过程可能会比较久，如果你的网络比较慢，请耐心一点等待吧。

遇到问题：

* Could not get lock /var/lib/dpkg/lock - open (11: Resource temporarily unava 解决办法：<http://blog.csdn.net/u011596455/article/details/60322568>

#### 第四步：安装MongoDB包

安装最新的稳定版本的MongoDB。发出以下命令：

$ sudo apt-get install -y mongodb-org

### 运行MongoDB

大多数类Unix操作系统限制会话，因为它们可能使用的系统资源。这些限制可能会对MongoDB操作产生负面影响。有关详细信息，请参阅[UNIX ulimit设置](http://docs.mongodb.com/manual/reference/ulimit/)。

MongoDB 实例默认将其数据文件存储在/var/lib/mongodb目录中，日志文件存储在/var/log/mongodb目录中，并且使用 mongodb 帐户运行。可以在/etc/mongod.conf中指定备用日志目录和数据文件目录。有关其他信息，请参阅[systemLog.path](http://docs.mongodb.com/manual/reference/configuration-options/#systemLog.path)和[storage.dbPath](http://docs.mongodb.com/manual/reference/configuration-options/#storage.dbPath)。

如果更改运行 MongoDB 进程的用户，则可通过修改/var/lib/mongodb和/var/log/mongodb目录的访问控制权限，以使该用户能够访问这些目录。

### 启动 MongoDB

发出以下命令启动 mongodb：

$ sudo service mongod start

**验证MongoDB是否已成功启动**

通过检查 /var/log/mongodb/mongod.log 中的日志文件的内容来验证 mongod 进程是否已成功启动，日记文件中应该有类似下面的行内容：

...

2017-06-28T21:17:26.292-0700 I CONTROL [initandlisten]

2017-06-28T21:17:26.341-0700 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory '/var/lib/mongodb/diagnostic.data'

2017-06-28T21:17:26.366-0700 I INDEX [initandlisten] build index on: admin.system.version properties: { v: 2, key: { version: 1 }, name: "incompatible\_with\_version\_32", ns: "admin.system.version" }

2017-06-28T21:17:26.366-0700 I INDEX [initandlisten] building index using bulk method; build may temporarily use up to 500 megabytes of RAM

2017-06-28T21:17:26.367-0700 I INDEX [initandlisten] build index done. scanned 0 total records. 0 secs

2017-06-28T21:17:26.368-0700 I COMMAND [initandlisten] setting featureCompatibilityVersion to 3.4

2017-06-28T21:17:26.368-0700 I NETWORK [thread1] waiting for connections on port 27017

**使用 MongoDB 命令测试**

yiibai@ubuntu:~$ mongo

MongoDB shell version v3.4.5

connecting to: mongodb://127.0.0.1:27017

MongoDB server version: 3.4.5

Welcome to the MongoDB shell.

For interactive help, type "help".

For more comprehensive documentation, see

http://docs.mongodb.org/

Questions? Try the support group

http://groups.google.com/group/mongodb-user

Server has startup warnings:

2017-06-28T21:17:26.133-0700 I STORAGE [initandlisten]

2017-06-28T21:17:26.133-0700 I STORAGE [initandlisten] \*\* WARNING: Using ..... .....

mm/transparent\_hugepage/defrag is 'always'.

2017-06-28T21:17:26.292-0700 I CONTROL [initandlisten] \*\* We suggest set ting it to 'never'

2017-06-28T21:17:26.292-0700 I CONTROL [initandlisten]

## 使用几个命令

> show dbs

admin 0.000GB

local 0.000GB

>

> db

test

> use test

switched to db test

>

### 停止MongoDB

根据需要，可以通过发出以下命令来停止 mongod 进程：

$ sudo service mongod stop

### 重新启动MongoDB

发出以下命令重新启动 mongod：

$ sudo service mongod restart

## RedHat/CentOS下安装配置MangoDB

本节介绍如何在Red Hat Enterprise Linux或CentOS Linux版本6和7上使用.rpm软件包来安装 MongoDB 社区版。 虽然Red Hat包含自己的MongoDB软件包，但安装最新 MongoDB社区版还要自己手动安装。

平台支持： 本安装指南仅支持64位系统。有关详细信息，请参阅[平台支持](http://docs.mongodb.com/manual/release-notes/3.0-compatibility/#compatibility-platform-support)。MongoDB 3.2不支持Red Hat Enterprise Linux 5。

mongodb-org-server包提供了一个使用/etc/mongod.conf配置文件启动 mongod 的初始化脚本。

由软件包提供的默认的 /etc/mongod.conf 配置文件的 bind\_ip 默认设置为 127.0.0.1。 在初始化副本集合之前，根据环境的需要修改此设置。

### 安装MongoDB社区版

本安装指南仅支持64位系统。有关详细信息，请参阅[平台支持](http://docs.mongodb.com/manual/release-notes/3.0-compatibility/#compatibility-platform-support)。请参考以下步骤一步一步地来完成。

**第一步：配置包管理系统(yum)**

创建一个 /etc/yum.repos.d/mongodb-org-3.4.repo 文件，以便可以使用 yum 命令直接安装 MongoDB。

MongoDB版本3.0中更改：MongoDB Linux软件包位于以 3.0 开头的新存储库中。

对于最新稳定发布的MongoDB

创建存储库文件(使用命令：vi /etc/yum.repos.d/mongodb-org-3.4.repo)，写入如下内容：

[mongodb-org-3.4]

name=MongoDB Repository

baseurl=http://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/3.4/x86\_64/

gpgcheck=1

enabled=1

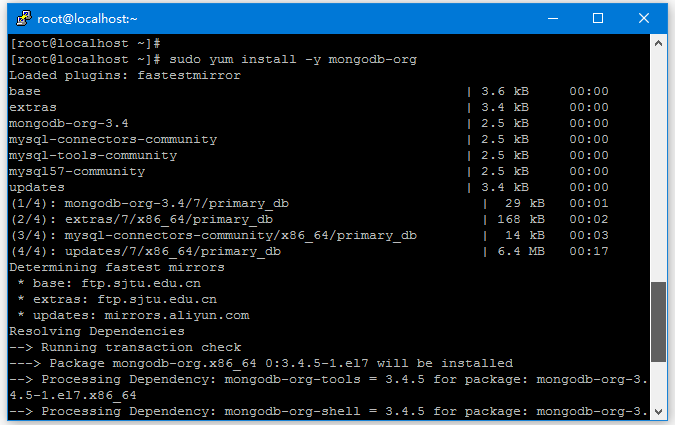
gpgkey=http://www.mongodb.org/static/pgp/server-3.4.asc

**第二步：安装MongoDB软件包和相关工具**

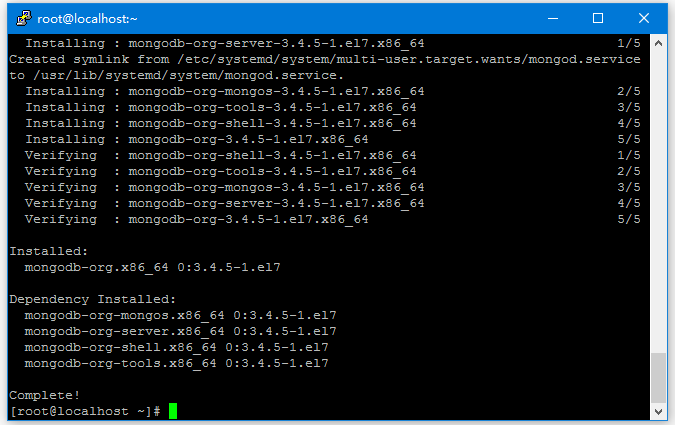
要安装最新的稳定版本(3.4)的MongoDB，请执行以下命令：

# sudo yum install -y mongodb-org

执行过程如下所示：



安装完成后，如下图所示：



### 运行MongoDB

大多数类Unix操作系统限制会话，以减少系统资源的使用。 这些限制可能会对 MongoDB 操作产生负面影响。有关详细信息，请参阅[UNIX ulimit设置](http://docs.mongodb.com/manual/reference/ulimit/)。

**配置SELinux**

如果您使用SELinux，则必须配置SELinux以允许MongoDB在基于Red Hat Linux的系统(Red Hat Enterprise Linux或CentOS Linux)上启动。

要配置SELinux，管理员有三个选项：

* 如果SELinux处于强制模式，则可以访问MongoDB部署将使用的相关端口(例如27017)。 有关MongoDB默认端口的更多信息，请参阅[默认MongoDB端口](http://docs.mongodb.com/manual/reference/default-mongodb-port/)。 对于默认设置，可以通过运行下面命令来实现 -

# semanage port -a -t mongod\_port\_t -p tcp 27017

* 通过在/etc/selinux/config中将 SELINUX 设置为 disabled 来禁用SELinux。

SELINUX=disabled

以学习为目的，建议使用此项设置，但是必须重新启动系统才能使更改生效。

* 在/etc/selinux/config 中设置为许可模式，即，将SELinux设置为 permissive 。

SELINUX=permissive

上述设置完成后，必须重新启动系统才能使更改生效。可以使用 setenforce 更改为允许模式。 setenforce 不需要重启，但不持续。或者，可以选择在安装Linux操作系统时不要安装SELinux软件包，或选择删除相关软件包。这个选项是最具侵入性的，所以不推荐。

MongoDB实例默认将其数据文件存储在/var/lib/mongo目录，及其日志文件/var/log/mongodb中，并使用 mongod 用户帐户运行。可以在 /etc/mongod.conf 中指定备用日志和数据文件目录。 有关其他信息，请参阅[systemLog.path](http://docs.mongodb.com/manual/reference/configuration-options/#systemLog.path)和[storage.dbPath](http://docs.mongodb.com/manual/reference/configuration-options/#storage.dbPath)。

如果更改运行 MongoDB 进程的用户，则必须修改 */var/lib/mongo* 和 */var/log/mongodb* 目录的访问控制权限，以使此用户能够访问这些目录。

### 启动 MongoDB

可以通过发出以下命令来启动 mongod 进程：

# sudo service mongod start

**验证MongoDB是否已成功启动**

可以通过检查 /var/log/mongodb/mongod.log 中的日志文件的内容来验证 mongod 进程是否成功启动，查看此文件应该会看到类似的内容：

[root@localhost ~]# cat /var/log/mongodb/mongod.log

2017-06-29T04:36:57.410-0400 I CONTROL [main] \*\*\*\*\* SERVER RESTARTED \*\*\*\*\*

2017-06-29T04:36:57.695-0400 I CONTROL [initandlisten] options: { command: [ "run" ], config: "/etc/mongod.conf", net: { bindIp: "127.0.0.1", port: 27017 }, processManagement: { fork: true, pidFilePath: "/var/run/mongodb/mongod.pid" }, storage: { dbPath: "/var/lib/mongo", journal: { enabled: true } }, systemLog: { destination: "file", logAppend: true, path: "/var/log/mongodb/mongod.log", quiet: true } }

2017-06-29T04:36:57.809-0400 I STORAGE [initandlisten] wiredtiger\_open config: create,cache\_size=256M,session\_max=20000,eviction=(threads\_min=4,threads\_max=4),config\_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file\_manager=(close\_idle\_time=100000),checkpoint=(wait=60,log\_size=2GB),statistics\_log=(wait=0),

... ...

2017-06-29T04:36:59.529-0400 I INDEX [initandlisten] build index on: admin.system.version properties: { v: 2, key: { version: 1 }, name: "incompatible\_with\_version\_32", ns: "admin.system.version" }

2017-06-29T04:36:59.529-0400 I INDEX [initandlisten] building index using bulk method; build may temporarily use up to 500 megabytes of RAM

2017-06-29T04:36:59.541-0400 I INDEX [initandlisten] build index done. scanned 0 total records. 0 secs

2017-06-29T04:36:59.542-0400 I COMMAND [initandlisten] setting featureCompatibilityVersion to 3.4

2017-06-29T04:36:59.548-0400 I NETWORK [thread1] waiting for connections on port 27017

[root@localhost ~]#

或者，使用以下命令：

[root@localhost ~]# ps -aux|grep mongod

mongod 2169 2.7 3.9 971208 39804 ? Sl 04:36 0:03 /usr/bin/mongod --quiet -f /etc/mongod.conf run

root 2376 0.0 0.0 112648 964 pts/0 R+ 04:38 0:00 grep --color=auto mongod

[root@localhost ~]#

可以选择确保通过发出以下命令，MongoDB 将在系统启动后启动 mongod 服务：

# sudo chkconfig mongod on

### 停止 MongoDB

根据需要，可以通过发出以下命令来停止 mongod 进程：

# sudo service mongod stop

### 重启 MongoDB

根据需要，可以通过发出以下命令来重启 mongod 进程：

# sudo service mongod restart

可以通过查看 */var/log/mongodb/mongod.log* 文件中的输出来跟踪进程状态的错误或重要信息。

### 开始使用 MongoDB

进入 MongoDB 命令行，可以对MongoDB进行操作了：

[root@localhost ~]# mongo

## 查看数据库

> show dbs;

## 查看数据库版本

> db.version();

## 常用命令帮助

> db.help();

[root@localhost ~]# mongo

MongoDB shell version v3.4.5

connecting to: mongodb://127.0.0.1:27017

MongoDB server version: 3.4.5

Welcome to the MongoDB shell.

For interactive help, type "help".

For more comprehensive documentation, see

http://docs.mongodb.org/

Questions? Try the support group

http://groups.google.com/group/mongodb-user

Server has startup warnings:

2017-06-29T04:36:58.888-0400 I CONTROL [initandlisten]

2017-06-29T04:36:58.888-0400 I CONTROL [initandlisten] \*\* WARNING: Access contr ol is not enabled for the database.

2017-06-29T04:36:58.888-0400 I CONTROL [initandlisten] \*\* Read and wri te access to data and configuration is unrestricted.

2017-06-29T04:36:58.888-0400 I CONTROL [initandlisten]

2017-06-29T04:36:58.892-0400 I CONTROL [initandlisten]

2017-06-29T04:36:58.892-0400 I CONTROL [initandlisten] \*\* WARNING: /sys/kernel/ mm/transparent\_hugepage/enabled is 'always'.

2017-06-29T04:36:58.892-0400 I CONTROL [initandlisten] \*\* We suggest set ting it to 'never'

2017-06-29T04:36:58.892-0400 I CONTROL [initandlisten]

2017-06-29T04:36:58.892-0400 I CONTROL [initandlisten] \*\* WARNING: /sys/kernel/ mm/transparent\_hugepage/defrag is 'always'.

2017-06-29T04:36:58.892-0400 I CONTROL [initandlisten] \*\* We suggest set ting it to 'never'

2017-06-29T04:36:58.892-0400 I CONTROL [initandlisten]

>

> show dbs;

admin 0.000GB

local 0.000GB

> use local

switched to db local

> db.version()

3.4.5

# MangoDB知识

## MangoDB基本概念

MongoDB是一个跨平台，面向文档的数据库，提供高性能，高可用性和易于扩展。MongoDB是工作在集合和文档上一种概念。

### 数据库

数据库是一个集合的物理容器。每个数据库获取其自己设定在文件系统上的文件。一个单一的MongoDB服务器通常有多个数据库。

### 集合

集合是一组MongoDB的文件。它与一个RDBMS表是等效的。一个集合存在于数据库中。集合不强制执行模式。集合中的文档可以有不同的字段。通常情况下，在一个集合中的所有文件都是类似或相关的。

### 文档

文档是一组键值对。文档具有动态模式。动态模式是指在同一个集合的文件不必具有相同一组集合的文档字段或结构，并且相同的字段可以保持不同类型的数据。

### 示例

下面给出的示例显示了一个博客网站的存储信息，存储的“文档”内容是用逗号分隔的键值对。

{

\_id: ObjectId(7df78ad8902c)

title: 'MongoDB Overview',

description: 'MongoDB is no sql database',

by: 'yiibai tutorial',

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

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

likes: 100,

comments: [

{

user:'user1',

message: 'My first comment',

dateCreated: new Date(2011,1,20,2,15),

like: 0

},

{

user:'user2',

message: 'My second comments',

dateCreated: new Date(2011,1,25,7,45),

like: 5

}

]

}

## MangoDB数据类型

MongoDB支持许多数据类型，包括：

* **字符串** - 这是用于存储数据的最常用的数据类型。MongoDB中的字符串必须为UTF-8。
* **整型** - 此类型用于存储数值。 整数可以是32位或64位，具体取决于服务器。
* **布尔类型** - 此类型用于存储布尔值(true / false)值。
* **双精度浮点数** - 此类型用于存储浮点值。
* **最小/最大键** - 此类型用于将值与最小和最大BSON元素进行比较。
* **数组** - 此类型用于将数组或列表或多个值存储到一个键中。
* **时间戳** - ctimestamp，当文档被修改或添加时，可以方便地进行录制。
* **对象** - 此数据类型用于嵌入式文档。
* **Null** - 此类型用于存储Null值。
* **符号** - 该数据类型与字符串相同; 但是，通常保留用于使用特定符号类型的语言。
* **日期** - 此数据类型用于以UNIX时间格式存储当前日期或时间。您可以通过创建日期对象并将日，月，年的日期进行指定自己需要的日期时间。
* **对象ID** - 此数据类型用于存储文档的ID。
* **二进制数据** - 此数据类型用于存储二进制数据。
* **代码** - 此数据类型用于将JavaScript代码存储到文档中。
* **正则表达式** - 此数据类型用于存储正则表达式。

## MangoDB数据建模

MongoDB中的数据具有灵活的模式。同一集合中的文档不需要具有相同的字段或结构；文档中的公共字段可以包含不同类型的数据。与SQL数据库不同，SQL数据库必须在插入数据之前确定和声明表的模式，MongoDB的集合不会强制统一文档结构。这种灵活性有助于将文档映射到实体或对象。即使数据具有实质性的变化，每个文档也可以匹配表示实体的数据字段。

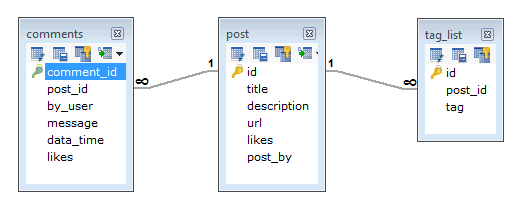
数据建模中的关键挑战是平衡应用程序的需求、数据库引擎的性能特征和数据检索模式。 在设计数据模型时，请始终考虑应用程序使用情况(即数据的查询，更新和处理)，以及数据本身的固有结构。

**实例**

假设客户需要他的博客/网站的数据库设计，并查看RDBMS和MongoDB架构设计之间的区别。网站有以下要求。

* 每个帖子都有唯一的标题，描述和网址。
* 每个帖子都可以有一个或多个标签。
* 每个帖子都有其发布者的名称和总人数。
* 每个帖子都有用户给出的评论以及他们的姓名，消息，数据时间和喜好。
* 每个帖子可以有零个或多个评论。

在RDBMS架构中，上述要求的设计将具有最少的三个表。表与表之间的关系如下：



而在MongoDB模式中，设计一个集合“post”，其结构以下：

{

\_id: POST\_ID

title: TITLE\_OF\_POST,

description: POST\_DESCRIPTION,

by: POST\_BY,

url: URL\_OF\_POST,

tags: [TAG1, TAG2, TAG3],

likes: TOTAL\_LIKES,

comments: [

{

user:'COMMENT\_BY',

message: TEXT,

dateCreated: DATE\_TIME,

like: LIKES

},

{

user:'COMMENT\_BY',

message: TEXT,

dateCreated: DATE\_TIME,

like: LIKES

}

]

}

通过上面的示例可以知道，显示数据时，在RDBMS中需要连接三个表，而在MongoDB中，数据将仅显示一个集合中的内容即可。

# 各语言连接MangoDB

## Java连接MangoDB

### 安装 MongoDB JDBC驱动程序

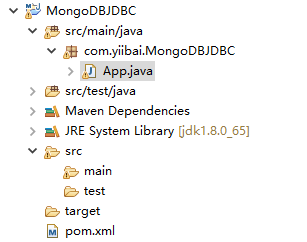
要在Java程序中使用MongoDB，需要确保在机器上设置了MongoDB JDBC驱动程序。我们来看一下如何设置MongoDB JDBC驱动。

* 下载 mongo.jar([http://mongodb.github.io/mongo-java-driver/)，确保下载最新版本。](http://mongodb.github.io/mongo-java-driver/)%EF%BC%8C%E7%A1%AE%E4%BF%9D%E4%B8%8B%E8%BD%BD%E6%9C%80%E6%96%B0%E7%89%88%E6%9C%AC%E3%80%82)
* 将 mongo.jar 包含到类路径中。

### 连接到数据库

要连接数据库，需要指定数据库名称，如果数据库不存在，那么MongoDB会自动创建它。

使用 Eclipse 创建一个 Maven 工程 - *MongoDBJDBC* ，其目录结构如下所示：



pom.xml文件的内容如下：

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.yiibai</groupId>

<artifactId>MongoDBJDBC</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>MongoDBJDBC</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mongodb</groupId>

<artifactId>mongodb-driver</artifactId>

<version>3.4.2</version>

</dependency>

</dependencies>

<build>

<defaultGoal>compile</defaultGoal>

</build>

</project>

以下是连接到数据库的代码片段：

package com.yiibai.MongoDBJDBC;

/\*\*

\* Hello world!

\*

\*/

import com.mongodb.MongoClient;

import com.mongodb.client.MongoDatabase;

public class App {

public static void main(String args[]) {

try {

// To connect to mongodb server

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

// Now connect to your databases

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

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

System.out.println("MongoDatabase inof is : "+mgdb.getName());

// If MongoDB in secure mode, authentication is required.

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

// System.out.println("Authentication: "+auth);

} catch (Exception e) {

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

}

}

}

默认认证机制：

String user; // the user name

String database; // the name of the database in which the user is defined

char[] password; // the password as a character array

// ...

MongoCredential credential = MongoCredential.createCredential(user, database, password);

MongoClient mongoClient = new MongoClient(new ServerAddress("host1", 27017),

Arrays.asList(credential));

或者使用连接字符串而不明确指定认证机制：

MongoClientURI uri = new MongoClientURI("mongodb://user1:pwd1@host1/?authSource=db1");

MongoClient mongoClient = new MongoClient(uri);

执行上面代码，得到以下结果：

com.mongodb.diagnostics.logging.JULLogger log

信息: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=UNKNOWN, serverSelectionTimeout='30000 ms', maxWaitQueueSize=500}

Connect to database successfully!

MongoDatabase inof is : mycol

### 创建和列出集合

要创建集合，可使用 com.mongodb.DB 类的 createCollection()方法。

以下是创建集合的代码片段：

package com.yiibai.MongoDBJDBC;

import org.bson.Document;

import com.mongodb.DBCollection;

import com.mongodb.client.model.CreateCollectionOptions;

/\*\*

\* Hello world!

\*

\*/

import com.mongodb.MongoClient;

import com.mongodb.client.MongoCollection;

import com.mongodb.client.MongoDatabase;

public class CreateCollection {

public static void main(String args[]) {

try {

// To connect to mongodb server

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

// Now connect to your databases

MongoDatabase database = mongoClient.getDatabase("test");

//database.createCollection("NewCollection",new CreateCollectionOptions().capped(true).sizeInBytes(0x100000));

MongoCollection<Document> coll = database.getCollection("myTestCollection");

System.out.println("Collection created successfully");

System.out.println("当前数据库中的所有集合是：");

for (String name : database.listCollectionNames()) {

System.out.println(name);

}

} catch (Exception e) {

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

}

}

}

执行上面代码，得到以下结果：

Collection created successfully

当前数据库中的所有集合是：

mycol

inventory

post

mycollection

newcollection

### 插入文档

要将文档插入到MongoDB中，使用com.mongodb.DBCollection类的insertOne()方法。

以下是插入文档的代码片段：

package com.yiibai.MongoDBJDBC;

import java.util.Arrays;

import org.bson.Document;

import com.mongodb.BasicDBObject;

import com.mongodb.Block;

import com.mongodb.DBCollection;

import com.mongodb.DBCursor;

/\*\*

\* Hello world!

\*

\*/

import com.mongodb.MongoClient;

import com.mongodb.client.MongoCollection;

import com.mongodb.client.MongoCursor;

import com.mongodb.client.MongoDatabase;

import com.mongodb.client.model.CreateCollectionOptions;

public class InsertDocument {

public static void main(String args[]) {

try {

// To connect to mongodb server

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

// Now connect to your databases

MongoDatabase database = mongoClient.getDatabase("test");

// database.createCollection("NewCollection", new

// CreateCollectionOptions().capped(true).sizeInBytes(0x100000));

System.out.println("Collection created successfully");

System.out.println("当前数据库中的所有集合是：");

for (String name : database.listCollectionNames()) {

System.out.println(name);

}

MongoCollection coll = database.getCollection("mycol");

System.out.println("Collection mycol selected successfully");

MongoCollection<Document> collection = database.getCollection("mycol");

Document document = new Document("\_id", 1999).append("title", "MongoDB Insert Demo")

.append("description","database")

.append("likes", 30)

.append("by", "yiibai point")

.append("url", "http://www.yiibai.com/mongodb/");

collection.insertOne(document);

collection.find().forEach(printBlock);

System.out.println("Document inserted successfully");

} catch (Exception e) {

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

}

}

static Block<Document> printBlock = new Block<Document>() {

public void apply(final Document document) {

System.out.println(document.toJson());

}

};

}

执行上面代码，得到以下结果：

NewCollection

mycol

inventory

post

Collection

mycollection

newcollection

Collection mycol selected successfully

{ "\_id" : 123.0, "title" : "MongoDB Overview", "description" : "MongoDB is no sql database", "by" : "yiibai tutorials", "url" : "http://www.yiibai.com", "tags" : ["mongodb", "database", "NoSQL"], "likes" : 100.0 }

{ "\_id" : 100.0, "title" : "MongoDB Overview", "description" : "MongoDB is no sql database", "by" : "yiibai tutorials", "url" : "http://www.yiibai.com", "tags" : ["mongodb", "database", "NoSQL"], "likes" : 100.0 }

{ "\_id" : 1999, "title" : "MongoDB Insert Demo", "description" : "database", "likes" : 30, "by" : "yiibai point", "url" : "http://www.yiibai.com/mongodb/" }

Document inserted successfully

**如何插入多行？**

Document doc1 = new Document("name", "Amarcord Pizzeria")

.append("contact", new Document("phone", "264-555-0193")

.append("email", "amarcord.pizzeria@example.net")

.append("location",Arrays.asList(-73.88502, 40.749556)))

.append("stars", 2)

.append("categories", Arrays.asList("Pizzeria", "Italian", "Pasta"));

Document doc2 = new Document("name", "Blue Coffee Bar")

.append("contact", new Document("phone", "604-555-0102")

.append("email", "bluecoffeebar@example.com")

.append("location",Arrays.asList(-73.97902, 40.8479556)))

.append("stars", 5)

.append("categories", Arrays.asList("Coffee", "Pastries"));

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

documents.add(doc1);

documents.add(doc2);

collection.insertMany(documents);

### 更新文档

要从集合更新文档，使用com.mongodb.DBCollection类的update()和updateMany()方法。以下是选择第一个文档的代码片段：

package com.yiibai.MongoDBJDBC;

import org.bson.Document;

import com.mongodb.Block;

/\*\*

\* Hello world!

\*

\*/

import com.mongodb.MongoClient;

import com.mongodb.MongoClientURI;

import com.mongodb.ServerAddress;

import com.mongodb.client.MongoDatabase;

import com.mongodb.client.MongoCollection;

import org.bson.Document;

import com.mongodb.client.MongoCursor;

import static com.mongodb.client.model.Filters.\*;

import com.mongodb.client.result.DeleteResult;

import static com.mongodb.client.model.Updates.\*;

import com.mongodb.client.result.UpdateResult;

import java.util.ArrayList;

import java.util.List;

public class App {

public static void main(String args[]) {

try {

// To connect to mongodb server

MongoClient mongoClient = new MongoClient("127.0.0.1", 27017);

// Now connect to your databases

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

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

System.out.println("MongoDatabase inof is : "+database.getName());

MongoCollection<Document> collection = database.getCollection("mycol");

//collection.updateOne(eq("\_id", 1999),new Document("$set", new Document("title", "更新了标题2")));

Document document = new Document("\_id", 9999).append("title", "MongoDB Insert Demo")

.append("description","database")

.append("likes", 30)

.append("by", "yiibai point")

.append("url", "http://www.yiibai.com/mongodb/");

collection.insertOne(document);

collection.find().forEach(printBlock);

collection.updateOne(eq("\_id", 9999),new Document("$set", new Document("title", "更新了标题2")));

collection.find().forEach(printBlock);

} catch (Exception e) {

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

}

}

static Block<Document> printBlock = new Block<Document>() {

public void apply(final Document document) {

System.out.println(document.toJson());

}

};

}

执行上面代码，得到以下结果 -

{ "\_id" : 1999, "title" : "更新了标题2", "description" : "database", "likes" : 30, "by" : "yiibai point", "url" : "http://www.yiibai.com/mongodb/" }

{ "\_id" : 9999, "title" : "MongoDB Insert Demo", "description" : "database", "likes" : 30, "by" : "yiibai point", "url" : "http://www.yiibai.com/mongodb/" }

{ "\_id" : 1999, "title" : "更新了标题2", "description" : "database", "likes" : 30, "by" : "yiibai point", "url" : "http://www.yiibai.com/mongodb/" }

{ "\_id" : 9999, "title" : "更新了标题2", "description" : "database", "likes" : 30, "by" : "yiibai point", "url" : "http://www.yiibai.com/mongodb/" }

### 删除文档

要从集合中删除第一个文档，需要首先使用findOne()方法选择文档，然后使用com.mongodb.DBCollection类的remove()方法。

以下是删除第一个文档的代码片段 -

package com.yiibai.MongoDBJDBC;

import org.bson.Document;

import com.mongodb.Block;

/\*\*

\* Hello world!

\*

\*/

import com.mongodb.MongoClient;

import com.mongodb.MongoClientURI;

import com.mongodb.ServerAddress;

import com.mongodb.client.MongoDatabase;

import com.mongodb.client.MongoCollection;

import org.bson.Document;

import com.mongodb.client.MongoCursor;

import static com.mongodb.client.model.Filters.\*;

import com.mongodb.client.result.DeleteResult;

import static com.mongodb.client.model.Updates.\*;

import com.mongodb.client.result.UpdateResult;

import java.util.ArrayList;

import java.util.List;

public class App {

public static void main(String args[]) {

try {

// To connect to mongodb server

MongoClient mongoClient = new MongoClient("127.0.0.1", 27017);

// Now connect to your databases

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

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

System.out.println("MongoDatabase inof is : "+database.getName());

// 更新文档

MongoCollection<Document> collection = database.getCollection("mycol");

/\*\*

Document document = new Document("\_id", 9999).append("title", "MongoDB Insert Demo")

.append("description","database")

.append("likes", 30)

.append("by", "yiibai point")

.append("url", "http://www.yiibai.com/mongodb/");

collection.insertOne(document);

//collection.find().forEach(printBlock);

//collection.updateOne(eq("\_id", 9999),new Document("$set", new Document("title", "更新了标题2")));

\*/

collection.find().forEach(printBlock);

// 删除文档

collection.deleteOne(eq("\_id", 9999));

System.out.println("After Delete Document:");

collection.find().forEach(printBlock);

} catch (Exception e) {

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

}

}

static Block<Document> printBlock = new Block<Document>() {

public void apply(final Document document) {

System.out.println(document.toJson());

}

};

}

执行上面代码，得到以下结果 -

{ "\_id" : 1999, "title" : "更新了标题2", "description" : "database", "likes" : 30, "by" : "yiibai point", "url" : "http://www.yiibai.com/mongodb/" }

{ "\_id" : 9999, "title" : "更新了标题2", "description" : "database", "likes" : 30, "by" : "yiibai point", "url" : "http://www.yiibai.com/mongodb/" }

After Delete Document:

{ "\_id" : 1999, "title" : "更新了标题2", "description" : "database", "likes" : 30, "by" : "yiibai point", "url" : "http://www.yiibai.com/mongodb/" }

## Python连接MangoDB

### 安装PyMongo

注意：请勿安装“bson”软件包。 PyMongo配有自己的bson包; 执行“pip install bson”或“easy\_install bson”则会安装与PyMongo不兼容的第三方软件包。

**使用pip安装**

我们建议在所有平台上使用pip来安装pymongo：

C:\Users\Administrator>python -m pip install pymongo

Collecting pymongo

Downloading pymongo-3.4.0.tar.gz (583kB)

... ....

Installing collected packages: pymongo

Running setup.py install for pymongo ... done

Successfully installed pymongo-3.4.0

要获得pymongo的特定版本：

$ python -m pip install pymongo==3.1.1

要升级pymongo的版本：

$ python -m pip install --upgrade pymongo

**Python版本依赖**

PyMongo支持CPython 2.6,2.7,3.3+，PyPy和PyPy3。

GSSAPI和TLS的可选依赖关系：

GSSAPI认证需要Windows上的Unix或WinKerberos上的pykerberos。PyMongo可以自动安装正确的依赖关系：

$ python -m pip install pymongo[gssapi]

### 使用MongoClient建立连接

使用PyMongo时，第一步是运行 mongod 实例创建一个MongoClient。如下：

from pymongo import MongoClient

client = MongoClient()

上述代码将连接默认主机和端口。 也可以明确指定主机和端口，如下所示：

from pymongo import MongoClient

#client = MongoClient()

client = MongoClient('localhost', 27017)

或使用MongoDB URI格式：

client = MongoClient('mongodb://localhost:27017/')

### 获取数据库

MongoDB的一个实例可以支持多个独立的数据库。 在使用PyMongo时，可以使用MongoClient实例上的属性的方式来访问数据库：

db = client.pythondb

如果数据库名称使用属性方式访问无法正常工作(如：python-db)，则可以使用字典样式访问：

db = client['python-db']

### 获取集合

集合是存储在MongoDB中的一组文档，可以类似于关系数据库中的表。 在PyMongo中获取集合的工作方式与获取数据库相同：

collection = db.python\_collection

或(使用字典方式访问)：

collection = db['python-collection']

MongoDB中关于集合(和数据库)的一个重要注意事项是它们是懒创建的 - 上述任何命令都没有在MongoDB服务器上实际执行任何操作。当第一个文档插入集合时才创建集合和数据库。

集合是存储在MongoDB中的一组文档，可以被认为大致相当于关系数据库中的表。 在PyMongo中获取集合的工作方式与获取数据库相同：

### 文档

MongoDB中的数据使用JSON方式来表示文档(并存储)。 在PyMongo中使用字典来表示文档。例如，以下字典可能用于表示博客文章：

import datetime

from pymongo import MongoClient

client = MongoClient()

post = {"author": "Mike",

"text": "My first blog post!",

"tags": ["mongodb", "python", "pymongo"],

"date": datetime.datetime.utcnow()}

### 插入文档

要将文档插入到集合中，可以使用insert\_one()方法：

#!/usr/bin/python3

#coding=utf-8

import datetime

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

post = {"author": "Maxsu",

"text": "My first blog post!",

"tags": ["mongodb", "python", "pymongo"],

"date": datetime.datetime.utcnow()}

posts = db.posts

post\_id = posts.insert\_one(post).inserted\_id

print ("post id is ", post\_id)

执行上面代码，得到以下结果 -

post id is 595965fe4959eb09c4451091

插入文档时，如果文档尚未包含“\_id”键，则会自动添加“\_id”。 “\_id”的值在集合中必须是唯一的。 insert\_one()返回一个InsertOneResult的实例。 有关“\_id”的更多信息，请参阅[有关\_id文档](http://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.insert_one)。

插入第一个文档后，实际上已经在服务器上创建了帖子(posts)集合。可以列出数据库中的所有集合：

#!/usr/bin/python3

#coding=utf-8

import datetime

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

"""

post = {"author": "Maxsu",

"text": "My first blog post!",

"tags": ["mongodb", "python", "pymongo"],

"date": datetime.datetime.utcnow()}

posts = db.posts

post\_id = posts.insert\_one(post).inserted\_id

print ("post id is ", post\_id)

"""

cur\_collection = db.collection\_names(include\_system\_collections=False)

print("cur\_collection is :", cur\_collection)

执行上面代码，得到以下结果 -

cur\_collection is : ['posts']

### 使用find\_one()获取单个文档

MongoDB中执行的最基本的查询类型是find\_one()。 此方法返回与查询匹配的单个文档(如果没有匹配，则返回None)。 当知道只有一个匹配的文档，或只对第一个匹配感兴趣时则可考虑使用find\_one()方法。下面示例中使用find\_one()从帖子(posts)集中获取第一个文档：

#!/usr/bin/python3

#coding=utf-8

import datetime

import pprint

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

post = {"author": "Maxsu",

"text": "My first blog post!",

"tags": ["mongodb", "python", "pymongo"],

"date": datetime.datetime.utcnow()}

posts = db.posts

#post\_id = posts.insert\_one(post).inserted\_id

#print ("post id is ", post\_id)

pprint.pprint(posts.find\_one())

执行上面代码，得到以下结果 -

{'\_id': ObjectId('595965fe4959eb09c4451091'),

'author': 'Maxsu',

'date': datetime.datetime(2017, 7, 2, 21, 30, 38, 402000),

'tags': ['mongodb', 'python', 'pymongo'],

'text': 'My first blog post!'}

结果是匹配之前插入的字典格式(Json)。注意: 返回的文档包含一个“\_id”，它是在插入时自动添加的。

find\_one()方法还支持查询结果文档必须匹配的特定元素。要查询作者是“Maxsu”的文档，可以指定查询的条件，如下所示：

#!/usr/bin/python3

#coding=utf-8

import datetime

import pprint

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

post = {"author": "Minsu",

"text": "This blog post belong to Minsu!",

"tags": ["MySQL", "Oracle", "pymongo"],

"date": datetime.datetime.utcnow()}

posts = db.posts

post\_id = posts.insert\_one(post).inserted\_id

post = posts.find\_one({"author": "Maxsu"})

pprint.pprint(post)

#print (post)

执行上面代码，得到以下结果 -

{'\_id': ObjectId('595965fe4959eb09c4451091'),

'author': 'Maxsu',

'date': datetime.datetime(2017, 7, 2, 21, 30, 38, 402000),

'tags': ['mongodb', 'python', 'pymongo'],

'text': 'My first blog post!'}

### 通过ObjectId查询

也可以通过它的\_id找到一个帖子(post)，下面的示例子中演示如何根据给定的一个ObjectId查询数据：

#!/usr/bin/python3

#coding=utf-8

import datetime

import pprint

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

post = {"\_id": 100,

"author": "Kuber",

"text": "This is is my first post!",

"tags": ["Docker", "Shell", "pymongo"],

"date": datetime.datetime.utcnow()}

posts = db.posts

post\_id = posts.insert\_one(post).inserted\_id

print("post\_id is :", post\_id)

post = posts.find\_one({"\_id": post\_id})

print("Find By Post ID:")

pprint.pprint(post)

#print (post)

执行上面代码，得到以下结果 -

post\_id is : 100

Find By Post ID:

{'\_id': 100,

'author': 'Kuber',

'date': datetime.datetime(2017, 7, 3, 14, 14, 8, 28000),

'tags': ['Docker', 'Shell', 'pymongo'],

'text': 'This is is my first post!'}

Web应用程序中的常见任务是从请求URL获取ObjectId并找到匹配的文档。 在这种情况下，必须将ObjectId从一个字符串转换到find\_one()：

from bson.objectid import ObjectId

# The web framework gets post\_id from the URL and passes it as a string

def get(post\_id):

# Convert from string to ObjectId:

document = client.db.collection.find\_one({'\_id': ObjectId(post\_id)})

### 关于Unicode字符串的注释

您可能已经注意到，我们先前存储的常规Python字符串在从服务器检索时看起来是不同的(例如，u’Mike而不是“Mike”)。一个简短的解释是有序的字符串。

MongoDB以BSON格式存储数据。BSON字符串是UTF-8编码的，所以PyMongo必须确保它存储的任何字符串只包含有效的UTF-8数据。 常规字符串(<type'str'>)被验证并保存不变。 Unicode字符串(<type'unicode'>)首先被编码为UTF-8。 我们的示例字符串在Python shell中表示为u'Mike而不是“Mike”的原因是PyMongo将每个BSON字符串解码为Python unicode字符串，而不是常规str。

### 批量插入

为了执行更复杂一些的查询，我们再插入一些文档。 除了插入单个文档外，还可以通过将列表作为第一个参数传递给insert\_many()来执行批量插入操作。 这将在列表中插入每个文档，只向服务器发送一个命令：

#!/usr/bin/python3

#coding=utf-8

import datetime

import pprint

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

new\_posts = [{"\_id": 1000,

"author": "Curry",

"text": "Another post!",

"tags": ["bulk", "insert"],

"date": datetime.datetime(2017, 11, 12, 11, 14)},

{"\_id": 1001,"author": "Maxsu",

"title": "MongoDB is fun",

"text": "and pretty easy too!",

"date": datetime.datetime(2019, 11, 10, 10, 45)}]

posts = db.posts

result = posts.insert\_many(new\_posts)

print("Bulk Inserts Result is :", result.inserted\_ids)

#print (post)

执行上面代码，得到以下结果 -

Bulk Inserts Result is : [1000, 1001]

有几个有趣的事情要注意这个例子：

* insert\_many()的结果现在返回两个ObjectId实例，每个ID表示插入的一个文档。
* new\_posts[1]具有与其他帖子不同的“形状”(数据结构) - 没有“tags”字段，添加了一个新字段“title”。MongoDB是无模式的，表示的就是这个意思。

### 查询多个文档

要查询获得超过单个文档作为查询的结果，可使用find()方法。find()返回一个Cursor实例，它允许遍历所有匹配的文档。如下示例，遍历帖子集合中的每个文档：

#!/usr/bin/python3

#coding=utf-8

import datetime

import pprint

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

posts = db.posts

for post in posts.find():

pprint.pprint(post)

执行上面代码，得到以下结果 -

{'\_id': ObjectId('595965fe4959eb09c4451091'),

'author': 'Maxsu',

'date': datetime.datetime(2017, 7, 2, 21, 30, 38, 402000),

'tags': ['mongodb', 'python', 'pymongo'],

'text': 'My first blog post!'}

{'\_id': 100,

'author': 'Kuber',

'date': datetime.datetime(2017, 7, 3, 14, 14, 8, 28000),

'tags': ['Docker', 'Shell', 'pymongo'],

'text': 'This is is my first post!'}

{'\_id': 1000,

'author': 'Curry',

'date': datetime.datetime(2017, 11, 12, 11, 14),

'tags': ['bulk', 'insert'],

'text': 'Another post!'}

{'\_id': 1001,

'author': 'Maxsu',

'date': datetime.datetime(2019, 11, 10, 10, 45),

'text': 'and pretty easy too!',

'title': 'MongoDB is fun'}

类似使用find\_one()一样，我们可以将文档传递给find()来限制返回的结果。 在这里，只希望得到作者是“Maxsu”的文档：

#!/usr/bin/python3

#coding=utf-8

import datetime

import pprint

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

posts = db.posts

for post in posts.find({"author": "Maxsu"}):

pprint.pprint(post)

执行上面的代码，得到以下结果 -

{'\_id': ObjectId('595965fe4959eb09c4451091'),

'author': 'Maxsu',

'date': datetime.datetime(2017, 7, 2, 21, 30, 38, 402000),

'tags': ['mongodb', 'python', 'pymongo'],

'text': 'My first blog post!'}

{'\_id': 1001,

'author': 'Maxsu',

'date': datetime.datetime(2019, 11, 10, 10, 45),

'text': 'and pretty easy too!',

'title': 'MongoDB is fun'}

### 计数统计

如果只想知道有多少文档匹配查询，可以执行count()方法操作，而不是一个完整的查询。 可以得到一个集合中的所有文档的计数：

#!/usr/bin/python3

#coding=utf-8

import datetime

import pprint

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

posts = db.posts

print("posts count is = ", posts.count())

print("posts's author is Maxsu count is =", posts.find({"author": "Maxsu"}).count())

执行上面代码，得到以下结果 -

posts count is = 4

posts's author is Maxsu count is = 2

### 范围查询

MongoDB支持许多不同类型的高级查询。例如，可以执行一个查询，将结果限制在比特定日期更早的帖子，而且还可以按作者对结果进行排序：

#!/usr/bin/python3

#coding=utf-8

import datetime

import pprint

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

posts = db.posts

d = datetime.datetime(2019, 11, 12, 12)

for post in posts.find({"date": {"$lt": d}}).sort("author"):

pprint.pprint(post)

这里使用特殊的“$lt”运算符做范围查询，并且还可以调用sort()来按作者对结果进行排序。

### 索引

添加索引可以帮助加速某些查询，并且还可以添加额外的功能来查询和存储文档。在这个例子中，将演示如何在一个键上创建一个唯一的索引，该索引将拒绝已经存在值的文档插入。

首先，我们创建索引：

result = db.profiles.create\_index([('user\_id', pymongo.ASCENDING)], unique=True)

sorted(list(db.profiles.index\_information()))

请注意，现在有两个索引：一个是MongoDB自动创建的在\_id索引，另一个是刚刚创建在user\_id上的索引。

现在来设置一些用户配置文件：

user\_profiles = [{'user\_id': 211, 'name': 'Luke'},{'user\_id': 212, 'name': 'Ziltoid'}]

result = db.profiles.insert\_many(user\_profiles)

该索引将阻止 user\_id 已经在集合中的文档插入：

new\_profile = {'user\_id': 213, 'name': 'Drew'}

duplicate\_profile = {'user\_id': 212, 'name': 'Tommy'}

result = db.profiles.insert\_one(new\_profile) # This is fine.

result = db.profiles.insert\_one(duplicate\_profile)

## 出现错误提示...

Traceback (most recent call last):

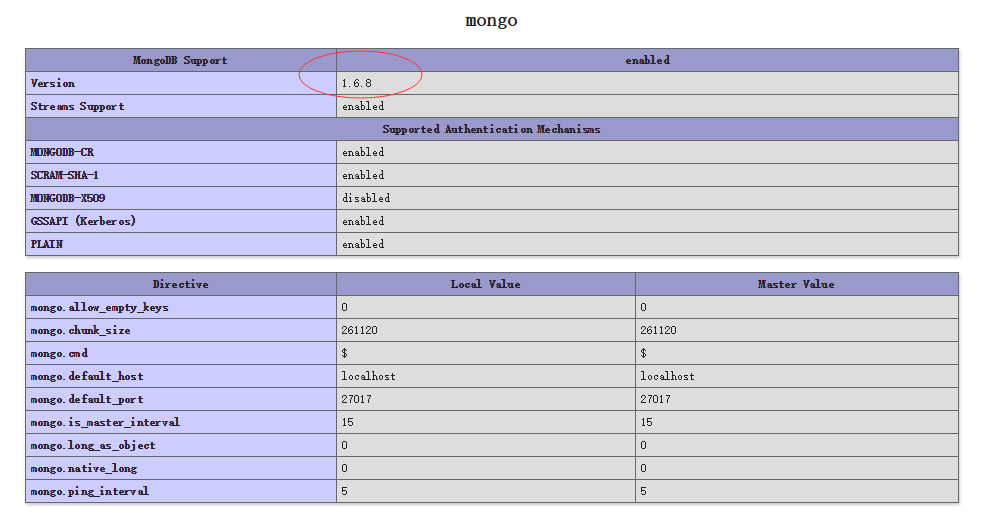
DuplicateKeyError: E11000 duplicate key error index: test\_database.profiles.$user\_id\_1 dup key: { : 212 }

## PHP连接MangoDB

要使用PHP与MongoDB交互存储数据，需要使用MongoDB PHP驱动程序([http://pecl.php.net/package/mongo)。](http://pecl.php.net/package/mongo)%E3%80%82) 从url下载驱动程序下载PHP驱动程序并确保下载的是正确的版本(如在本示例中：Win10 64位下载的版本是：php\_mongo-1.6.8-5.6-ts-vc11-x64.zip)。 现在解压缩存档并将php\_mongo.dll放入PHP扩展目录(默认为“ext”)，并将以下行添加到php.ini文件中 -

extension = php\_mongo.dll

然后重新启动 Apache 服务器，查看 phpinfo() 函数的输出结果 -



### 进行连接并选择数据库

要进行连接，需要指定数据库名称，如果数据库不存在，那么MongoDB会自动创建它。

以下是连接到数据库的代码片段 -

<?php

// connect to mongodb

$m = new MongoClient();

echo "Connection to database successfully<br/>";

// select a database

$db = $m->mydb;

echo "Database mydb selected";

?>

执行程序时，会产生以下结果 -

Connection to database successfully

Database mydb selected

### 创建一个集合

以下是创建集合的代码片段 -

<?php

// connect to mongodb

$m = new MongoClient();

echo "Connection to database successfully";

// select a database

$db = $m->mydb;

echo "Database mydb selected";

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

echo "Collection created succsessfully";

?>

执行程序时，会产生以下结果 -

Connection to database successfully

Database mydb selected

Collection created succsessfully

### 插入文档

要将文档插入到MongoDB中，请使用insert()方法。

以下是插入文档的代码片段 -

<?php

// connect to mongodb

$m = new MongoClient();

echo "Connection to database successfully";

// select a database

$db = $m->mydb;

echo "Database mydb selected";

$collection = $db->phpcol;

echo "Collection selected succsessfully";

$document = array(

"title" => "MongoDB",

"description" => "database",

"likes" => 100,

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

"by", "tutorials point"

);

$collection->insert($document);

echo "Document inserted successfully";

?>

执行程序时，会产生以下结果 -

Connection to database successfully

Database mydb selected

Collection selected succsessfully

Document inserted successfully

### 查找所有文件

要从集合中选择所有文档，请使用find()方法。

<?php

// connect to mongodb

$m = new MongoClient();

echo "Connection to database successfully";

// select a database

$db = $m->mydb;

echo "Database mydb selected";

$collection = $db->phpcol;

echo "Collection selected succsessfully";

$cursor = $collection->find();

// iterate cursor to display title of documents

foreach ($cursor as $document) {

echo $document["title"]. ', URL is=> ' .$document["url"] . "

";

}

?>

以下是选择所有文档的代码片段 -

Connection to database successfully

Database mydb selected

Collection selected succsessfully

MongoDB, URL is=> http://www.yiibai.com/mongodb/

### 更新文档

要更新文档，需要使用update()方法。

在下面的例子中，将把插入的文档的标题更新为：*MongoDB教程* 。 以下是更新文档的代码段 -

<?php

// connect to mongodb

$m = new MongoClient();

echo "Connection to database successfully";

// select a database

$db = $m->mydb;

echo "Database mydb selected";

$collection = $db->mycol;

echo "Collection selected succsessfully";

// now update the document

$collection->update(array("title"=>"MongoDB"),

array('$set'=>array("title"=>"MongoDB教程")));

echo "Document updated successfully";

// now display the updated document

$cursor = $collection->find();

// iterate cursor to display title of documents

echo "Updated document";

foreach ($cursor as $document) {

echo $document["title"] .', URL => '.$document["url"] . "

";

}

?>

以下是选择所有文档的代码片段 -

Connection to database successfully

Database mydb selected

Collection selected succsessfully

MongoDB, URL is=> http://www.yiibai.com/mongodb/

### 删除文档

要删除文档，可使用remove()方法。

在下面的示例中，将删除标题为：*MongoDB教程* 的文档。 以下是删除文档的代码片段 -

<?php

// connect to mongodb

$m = new MongoClient();

echo "Connection to database successfully";

// select a database

$db = $m->mydb;

echo "Database mydb selected";

$collection = $db->phpcol;

echo "Collection selected succsessfully";

// now remove the document

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

echo "Documents deleted successfully";

// now display the available documents

$cursor = $collection->find();

foreach ($cursor as $document) {

echo $document["title"] . "

";

}

?>

以下是选择所有文档的代码片段 -

Connection to database successfully

Database mydb selected

Collection selected succsessfully

Documents deleted successfully

在上面的例子中，第二个参数是布尔类型，用于remove()方法的justOne字段。

其余的MongoDB方法findOne()，save()，limit()，skip()，sort()等与上述相同。

## Ruby连接MangoDB

MongoDB Ruby驱动程序是MongoDB官方支持的Ruby驱动程序。它是用纯Ruby编写的，为了简化而进行了优化。它可以自己使用，但它也可以作为几个对象映射库的基础。

### 安装驱动程序

Ruby驱动程序是作为一个gem绑定的，并且托管在[Rubygems](http://rubygems.org/gems/mongo)上。

**安装gem**

驱动程序可以手动或捆绑式安装。手动安装gem：

gem install mongo

要使用捆绑安装gem，请在Gemfile中包含以下内容：

gem 'mongo', '~> 2.4'

### 前提条件

* MongoDB实例在localhost上运行使用默认端口27017。
* Ruby MongoDB驱动程序。有关如何安装MongoDB驱动程序的说明，请参阅安装。
* 代码顶部的以下语句：

require 'mongo'

### Ruby连接MongoDB

使用Mongo::Client建立与正在运行的MongoDB实例的连接。

require 'mongo'

client = Mongo::Client.new([ '127.0.0.1:27017' ], :database => 'test')

还可以使用URI连接字符串：

require 'mongo'

client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')

### 访问数据库和集合

以下示例演示如何访问指定数据库并显示其集合：

client = Mongo::Client.new([ '127.0.0.1:27017' ], :database => 'test')

db = client.database

db.collections # returns a list of collection objects

db.collection\_names # returns a list of collection names

要访问一个集合，请按名称查看。

collection = client[:restaurants]

如果集合不存在，服务器将在第一次放入数据时创建。

#### 插入文档

要将单个文档插入到集合中，请使用insert\_one方法。

client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')

collection = client[:people]

doc = { name: 'Steve', hobbies: [ 'hiking', 'tennis', 'fly fishing' ] }

result = collection.insert\_one(doc)

result.n # returns 1, because one document was inserted

要将多个文档插入到集合中，请使用insert\_many方法。

docs = [ { \_id: 1, name: 'Steve', hobbies: [ 'hiking', 'tennis', 'fly fishing' ] },

{ \_id: 2, name: 'Sally', hobbies: ['skiing', 'stamp collecting' ] } ]

result = collection.insert\_many(docs)

result.inserted\_count # returns 2 because two documents were inserted

#### 查询集合

使用find方法查询集合。一个空的查询过滤器返回集合中的所有文档。

client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')

collection = client[:people]

collection.find.each do |document|

#=> Yields a BSON::Document.

end

使用查询过滤器只找到匹配的文档。

client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')

collection = client[:people]

puts collection.find( { name: 'Sally' } ).first

该示例应该会打印以下内容：

{"\_id" => 2, "name" => "Sally", "hobbies" => ["skiing", "stamp collecting"]}

#### 更新文件

有几种更新方法，包括：update\_one和update\_many。 update\_one更新单个文档，而update\_many会一次更新多个文档。

这两种方法都将查询过滤器作为第一个参数，以及更新文档的数据作为第二个参数。 使用$set来添加或更新特定的字段。如果没有$set，则会将所有文档更新为给定的更新数据。

client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')

collection = client[:people]

result = collection.update\_one( { 'name' => 'Sally' }, { '$set' => { 'phone\_number' => "555-555-5555" } } )

puts collection.find( { 'name' => 'Sally' } ).first

#### 删除文档

使用delete\_one或delete\_many方法从集合中删除文档(单个或多个)。

client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')

collection = client[:people]

result = collection.delete\_one( { name: 'Steve' } )

puts result.deleted\_count # returns 1 because one document was deleted

以下示例将两个记录插入到集合中，然后使用与正则表达式匹配name字段查询，删除那些以“S”开头的字符串的所有文档。

client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')

collection = client[:people]

collection.insert\_many([ { \_id: 3, name: "Arnold" }, { \_id: 4, name: "Susan" } ])

puts collection.count # counts all documents in collection

result = collection.delete\_many({ name: /$S\*/ })

puts result.deleted\_count # returns the number of documents deleted

#### 创建索引

使用create\_one或create\_many方法一次创建一个索引或多个索引。

client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')

collection = client[:people]

collection.indexes.create\_one({ name: 1 }, unique: true)

使用create\_many方法使用一个语句创建多个索引。 请注意，使用create\_many时，语法与create\_one不同。

client = Mongo::Client.new('mongodb://127.0.0.1:27017/test')

collection = client[:people]

collection.indexes.create\_many([

{ key: { name: 1 } , unique: true },

{ key: { hobbies: 1 } },

])

# 使用MangoDB

## 创建数据库：use 命令

MongoDB使用 use DATABASE\_NAME 命令来创建数据库。如果指定的数据库DATABASE\_NAME不存在，则该命令将创建一个新的数据库，否则返回现有的数据库。

**语法**

use DATABASE 语句的基本语法如下：

use DATABASE\_NAME

**示例**

如果要创建一个名称为<newdb>的数据库，那么使用 use DATABASE 语句将如下所示：

> use newdb

switched to db newdb

要检查当前选择的数据库，请使用 db 命令：

>db

newdb

如果要检查数据库列表，请使用命令 show dbs：

>show dbs

local 0.000025GB

test 0.00002GB

创建的数据库(newdb)不在列表中。要显示数据库，需要至少插入一个文档，空的数据库是不显示出来的。

>db.items.insert({"name":"yiibai tutorials"})

>show dbs

local 0.00005GB

test 0.00002GB

newdb 0.00002GB

在 MongoDB 中默认数据库是：test。 如果您还没有创建过任何数据库，则集合/文档将存储在test数据库中。

## 删除数据库：db.dropDatabase() 方法

MongoDB中的 db.dropDatabase()命令用于删除现有的数据库。

**语法**

db.dropDatabase()

这将删除当前所选数据库。 如果没有选择任何数据库，那么它将删除默认的“test”数据库。

**示例**

首先，使用命令show dbs检查可用数据库的列表。

>show dbs

local 0.00025GB

newdb 0.0002GB

test 0.00012GB

如果要删除新数据库<newdb>，那么dropDatabase()命令将如下所示：

>use newdb

switched to db newdb

>db.dropDatabase()

>{ "dropped" : "newdb", "ok" : 1 }

现在检查数据库列表，如下：

>show dbs

local 0.00025GB

test 0.0002GB

## 创建集合：createCollection()方法

MongoDB 的 db.createCollection(name，options) 方法用于在MongoDB 中创建集合。

**语法**

createCollection()命令的基本语法如下：

db.createCollection(name, options)

在命令中，name 是要创建的集合的名称。 options是一个文档，用于指定集合的配置。

| **参数** | **类型** | **描述** |
| --- | --- | --- |
| name | String | 要创建的集合的名称 |
| options | Document | (可选)指定有关内存大小和索引的选项 |

options参数是可选的，因此只需要指定集合的名称。 以下是可以使用的选项列表：

| **字段** | **类型** | **描述** |
| --- | --- | --- |
| capped | Boolean | (可选)如果为true，则启用封闭的集合。上限集合是固定大小的集合，它在达到其最大大小时自动覆盖其最旧的条目。 如果指定true，则还需要指定size参数。 |
| autoIndexId | Boolean | (可选)如果为true，则在\_id字段上自动创建索引。默认值为false。 |
| size | 数字 | (可选)指定上限集合的最大大小(以字节为单位)。 如果capped为true，那么还需要指定此字段的值。 |
| max | 数字 | (可选)指定上限集合中允许的最大文档数。 |

在插入文档时，MongoDB首先检查上限集合capped字段的大小，然后检查max字段。

**示例**

没有使用选项的createCollection()方法的基本语法如下：

>use test

switched to db test

>db.createCollection("mycollection")

{ "ok" : 1 }

可以使用命令show collections检查创建的集合。

>show collections

Mycollection

以下示例显示了createCollection()方法的语法，其中几个重要选项：

> db.createCollection("mycol", {capped : true, autoIndexId : true, size : 6142800, max : 10000 })

{ "ok" : 1 }

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

>db.newcollection.insert({"name" : "yiibaitutorials"})

>show collections

mycol

newcollection

mycollection

## 删除集合：drop()方法

MongoDB 的 db.collection.drop() 用于从数据库中删除集合。

**语法**

drop()命令的基本语法如下

db.COLLECTION\_NAME.drop()

**示例**

首先，检查数据库 test 中可用的集合。

>use test

switched to db test

> show collections

mycol

mycollection

newcollection

现在删除名称为 mycollection 的集合。

>db.mycollection.drop()

true

再次检查当前数据库的集合列表，如下：

> show collections

mycol

newcollection

如果选定的集合成功删除，drop()方法将返回true，否则返回false。

## 插入文档：insert()方法、save()方法

要将数据插入到 MongoDB 集合中，需要使用MongoDB的insert()或save()方法。

**语法**

insert()命令的基本语法如下：

>db.COLLECTION\_NAME.insert(document)

**示例**

>db.mycol.insert({

\_id: 100,

title: 'MongoDB Overview',

description: 'MongoDB is no sql database',

by: 'yiibai tutorials',

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

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

likes: 100,

})

这里mycol是集合的名称，在前一章中所创建的。如果数据库中不存在集合，则MongoDB将创建此集合，然后将文档插入到该集合中。

在插入的文档中，如果不指定\_id参数，那么 MongoDB 会为此文档分配一个唯一的ObjectId。

\_id为集合中的每个文档唯一的12个字节的十六进制数。12字节划分如下：

\_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id,

3 bytes incrementer)

要在单个查询中插入多个文档，可以在insert()命令中传递文档数组。如下所示：

> db.mycol.insert([

{

\_id: 101,

title: 'MongoDB Guide',

description: 'MongoDB is no sql database',

by: 'yiibai tutorials',

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

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

likes: 100

},

{

\_id: 102,

title: 'NoSQL Database',

description: "NoSQL database doesn't have tables",

by: 'yiibai tutorials',

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

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

likes: 210,

comments: [

{

user:'user1',

message: 'My first comment',

dateCreated: new Date(2017,11,10,2,35),

like: 0

}

]

},

{

\_id: 104,

title: 'Python Quick Guide',

description: "Python Quick start ",

by: 'yiibai tutorials',

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

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

likes: 30,

comments: [

{

user:'user1',

message: 'My first comment',

dateCreated: new Date(2018,11,10,2,35),

like: 590

}

]

}

])

要插入文档，也可以使用db.post.save(document)。 如果不在文档中指定\_id，那么save()方法将与insert()方法一样自动分配ID的值。如果指定\_id，则将以save()方法的形式替换包含\_id的文档的全部数据。

## 其它插入文档的方法

### db.collection.insertOne()方法

db.collection.insertOne()方法将单个文档插入到集合中。以下示例将新文档插入到库存集合中。 如果文档没有指定\_id字段，MongoDB会自动将\_id字段与ObjectId值添加到新文档。

db.inventory.insertOne(

{ item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, uom: "cm" } }

)

db.collection.insertOne()方法返回包含新插入的文档的\_id字段值的文档。

执行结果如下：

> db.inventory.insertOne(

... { item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, uom: "cm" } }

... )

{

"acknowledged" : true,

"insertedId" : ObjectId("5955220846be576f199feb55")

}

### db.collection.insertMany()方法

db.collection.insertMany()方法将多个文档插入到集合中，可将一系列文档传递给db.collection.insertMany()方法。以下示例将三个新文档插入到库存集合中。如果文档没有指定\_id字段，MongoDB会向每个文档添加一个ObjectId值的\_id字段。

db.inventory.insertMany([

{ item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21, uom: "cm" } },

{ item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" } },

{ item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85, uom: "cm" } }

])

insertMany()返回包含新插入的文档\_id字段值的文档。执行结果如下：

> db.inventory.insertMany([

{ item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21, uom: "cm" } },

{ item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" } },

{ item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85, uom: "cm" } }

])

{

"acknowledged" : true,

"insertedIds" : [

ObjectId("59552c1c46be576f199feb56"),

ObjectId("59552c1c46be576f199feb57"),

ObjectId("59552c1c46be576f199feb58")

]

}

## 查询文档

### find()方法

要从MongoDB集合查询数据，需要使用MongoDB的find()方法。

**语法**

find()命令的基本语法如下：

>db.COLLECTION\_NAME.find(document)

find()方法将以非结构化的方式显示所有文档。

### pretty()方法

要以格式化的方式显示结果，可以使用pretty()方法。

**语法**

> db.mycol.find().pretty()

**示例**

>db.mycol.find().pretty()

{

"\_id": 100,

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"by": "yiibai tutorials",

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

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

"likes": "100"

}

>

除了find()方法外，还有一个findOne()方法，它只返回一个文档。

### MongoDB 与 RDBMS的等效 Where 子句

要在一些条件的基础上查询文档，可以使用以下操作。

| **操作** | **语法** | **示例** | **RDBMS等效语句** |
| --- | --- | --- | --- |
| 相等 | {<key>:<value>} | db.mycol.find({"by":"yiibai"}).pretty() | where by = 'yiibai' |
| 小于 | {<key>:{$lt:<value>}} | db.mycol.find({"likes":{$lt:50}}).pretty() | where likes < 50 |
| 小于等于 | {<key>:{$lte:<value>}} | db.mycol.find({"likes":{$lte:50}}).pretty() | where likes <= 50 |
| 大于 | {<key>:{$gt:<value>}} | db.mycol.find({"likes":{$gt:50}}).pretty() | where likes > 50 |
| 大于等于 | {<key>:{$gte:<value>}} | db.mycol.find({"likes":{$gte:50}}).pretty() | where likes >= 50 |
| 不等于 | {<key>:{$ne:<value>}} | db.mycol.find({"likes":{$ne:50}}).pretty() | where likes != 50 |

下面我们将对上表中的所有操作演示。

#### MongoDB中的AND操作符

**语法**

在find()方法中，如果通过使用“，”将它们分开传递多个键，则 MongoDB 将其视为AND条件。 以下是AND的基本语法：

>db.mycol.find(

{

$and: [

{key1: value1}, {key2:value2}

]

}

).pretty()

**示例**

以下示例将显示由“yiibai tutorials”编写并且标题为“*MongoDB Overview*”的所有教程。

> db.mycol.find({$and:[{"by":"yiibai tutorials"},{"title": "MongoDB Overview"}]}).pretty()

{

"\_id" : 100,

"title" : "MongoDB Overview",

"description" : "MongoDB is no sql database",

"by" : "yiibai tutorials",

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

"tags" : [

"mongodb",

"database",

"NoSQL"

],

"likes" : 100

}

对于上面给出的例子，等效的SQL where子句是：

SELECT \* FROM mycol where by ='yiibai tutorials' AND title ='MongoDB Overview'

可以在find子句中传递任意数量的键值。

#### MongoDB中的OR操作符

**语法**

在要根据OR条件查询文档，需要使用$or关键字。 以下是OR条件的基本语法：

>db.mycol.find(

{

$or: [

{key1: value1}, {key2:value2}

]

}

).pretty()

**示例**

以下示例将显示由“yiibai tutorials”编写或标题为“*MongoDB Overview*”的所有教程。

>db.mycol.find({$or:[{"by":"yiibai tutorials"},{"title": "MongoDB Overview"}]}).pretty()

{

"\_id": 100,

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"by": "yiibai tutorials",

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

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

"likes": "100"

}

#### 使用 AND 和 OR 条件一起

**示例**

以下示例将显示likes大于10以及标题是“MongoDB Overview”或者“yiibai tutorials”的所有文档。 等价SQL where子句为：

SELECT \* FROM mycol where likes> 10 AND(by ='yiibai tutorials' OR title ='MongoDB Overview')

>db.mycol.find({"likes": {$gt:10}, $or: [{"by": "yiibai tutorials"},

{"title": "MongoDB Overview"}]}).pretty()

{

"\_id": 100,

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"by": "yiibai tutorials",

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

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

"likes": "100"

}

#### 查询嵌入/嵌套文档

这里演示如何使用：db.collection.find()方法对嵌入/嵌套文档的查询操作的示例。 此页面上的示例使用inventory集合。要填充库存(inventory)集合以准备一些数据，请运行以下命令：

db.inventory.insertMany( [

{ item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },

{ item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },

{ item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },

{ item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },

{ item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }

]);

##### 匹配嵌入/嵌套文档

要在作为嵌入/嵌套文档的字段上指定相等条件，请使用查询过滤器文档{<field>：<value>}，其中<value>是要匹配的文档。

例如，以下查询选择字段size等于{ h: 14, w: 21, uom: "cm" }的所有文档：

db.inventory.find( { size: { h: 14, w: 21, uom: "cm" } } )

整个嵌入式文档中的相等匹配需要精确匹配指定的<value>文档，包括字段顺序。

例如，以下查询与库存(inventory)集合中的任何文档不匹配：

db.inventory.find( { size: { w: 21, h: 14, uom: "cm" } } )

##### 查询嵌套字段

要在嵌入/嵌套文档中的字段上指定查询条件，请使用点符号(“field.nestedField”)。

*在嵌套字段上指定等于匹配*

以下示例选择在size字段中嵌套的字段uom等于“in”的所有文档：

db.inventory.find( { "size.uom": "in" } )

##### 使用查询运算符指定匹配

查询过滤器文档可以使用查询运算符来指定，如以下形式的条件：

{ <field1>: { <operator1>: <value1> }, ... }

以下查询使用size字段中嵌入的字段h中的小于运算符($lt)：

db.inventory.find( { "size.h": { $lt: 15 } } )

##### 指定AND条件

以下查询选择嵌套字段h小于15的所有文档，嵌套字段uom等于“in”，status字段等于“D”：

db.inventory.find( { "size.h": { $lt: 15 }, "size.uom": "in", status: "D" } )

## 更新文档

MongoDB的update()和save()方法用于将集合中的文档更新。update()方法更新现有文档中的值，而save()方法使用save()方法中传递的文档数据替换现有文档。

### MongoDB Update()方法

update()方法更新现有文档中的值。

**语法**

update()方法的基本语法如下：

> db.COLLECTION\_NAME.update(SELECTION\_CRITERIA, UPDATED\_DATA)

**示例**

考虑mycol集合具有以下数据：

> db.mycol.find({}, {'\_id':1, 'title':1})

{ "\_id" : 100, "title" : "MongoDB Overview" }

{ "\_id" : 101, "title" : "MongoDB Guide" }

{ "\_id" : 102, "title" : "NoSQL Database" }

{ "\_id" : 104, "title" : "Python Quick Guide" }

以下示例将为标题为“MongoDB Overview”的文档设置为“New Update MongoDB Overview”。

> db.mycol.find({'title':'MongoDB Overview'},{'\_id':1, 'title':1})

{ "\_id" : 100, "title" : "MongoDB Overview" }

> # 更新操作

> db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New Update MongoDB Overview'}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> # 查询更新后的结果 -

> db.mycol.find({'\_id':100},{'\_id':1, 'title':1})

{ "\_id" : 100, "title" : "New Update MongoDB Overview" }

默认情况下，MongoDB只会更新一个文档。要更新多个文档，需要将参数’multi‘设置为true。

>db.mycol.update({'title':'MongoDB Overview'},

{$set:{'title':'New Update MongoDB Overview'}},{multi:true})

### MongoDB Save()方法

save()方法使用save()方法中传递的文档数据替换现有文档。

**语法**

MongoDB save()方法的基本语法如下所示：

>db.COLLECTION\_NAME.save({\_id:ObjectId(),NEW\_DATA})

**示例**

以下示例将\_id为100的文档使用新的文档替换。

db.mycol.save(

{

"\_id" : 100, "title":"Update By Save()Method.", "by":"yiibai.com"

}

)

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

db.mycol.find({'\_id':100}, {'\_id':1, 'title':1})

{ "\_id" : 100, "title" : "Update By Save()Method." }

## 删除文档：remove()方法

MongoDB中的 remove()方法用于从集合中删除文档。 remove()方法接受两个参数。 一个是删除条件，第二个是标志：justOne。

* *criteria* - (可选)符合删除条件的集合将被删除。
* *justOne* - (可选)如果设置为true或1，则只删除一个文档。

**语法**

remove()方法的基本语法如下 -

>db.COLLECTION\_NAME.remove(DELLETION\_CRITTERIA)

假设mycol集合具有以下数据。

> db.mycol.find({}, {'\_id':1, 'title':1})

{ "\_id" : 101, "title" : "MongoDB Guide" }

{ "\_id" : 102, "title" : "NoSQL Database" }

{ "\_id" : 104, "title" : "Python Quick Guide" }

{ "\_id" : 100, "title" : "Update By Save()Method." }

>

以下示例将删除\_id为“100”的文档。

> db.mycol.find({}, {'\_id':1, 'title':1})

{ "\_id" : 101, "title" : "MongoDB Guide" }

{ "\_id" : 102, "title" : "NoSQL Database" }

{ "\_id" : 104, "title" : "Python Quick Guide" }

{ "\_id" : 100, "title" : "Update By Save()Method." }

>

> db.mycol.remove({'\_id':100})

WriteResult({ "nRemoved" : 1 })

> db.mycol.find({}, {'\_id':1, 'title':1})

{ "\_id" : 101, "title" : "MongoDB Guide" }

{ "\_id" : 102, "title" : "NoSQL Database" }

{ "\_id" : 104, "title" : "Python Quick Guide" }

### 只删除一条文档记录

如果有多条记录，并且只想删除第一条记录，则在remove()方法中设置justOne参数。

>db.COLLECTION\_NAME.remove(DELETION\_CRITERIA,1)

### 删除所有文档记录

删除所有文档记录，可在remove()方法中设置justOne参数。

如果不指定删除条件，MongoDB 将删除集合中的所有文档。 这相当于SQL的truncate命令。

>db.mycol.remove()

>db.mycol.find()

## 投影（选择字段）：find()方法

在MongoDB中，投影表示仅选择所需要字段的数据，而不是选择整个文档字段的数据。如果某个文档有5个字段，但只要显示3个字段，那么就只选择3个字段吧，这样做是非常有好处的。

MongoDB的find()方法，在 MongoDB [查询文档](http://www.yiibai.com/mongodb/mongodb_query_document.html)中此方法接收的第二个可选参数是要检索的字段列表。 在MongoDB中，当执行find()方法时，它默认将显示文档的所有字段。为了限制显示的字段，需要将字段列表对应的值设置为1或0。1用于显示字段，而0用于隐藏字段。

find()方法的基本语法如下：

**语法**

>db.COLLECTION\_NAME.find({},{KEY:1})

**示例**

假设集合mycol有以下数据：

> db.mycol.find({}, {'\_id':1, 'title':1})

{ "\_id" : 101, "title" : "MongoDB Guide" }

{ "\_id" : 102, "title" : "NoSQL Database" }

{ "\_id" : 104, "title" : "Python Quick Guide" }

{ "\_id" : 100, "title" : "MongoDB Overview" }

以下示例将在查询文档时只显示文档的标题。

> db.mycol.find({}, {'title':1,'\_id':0})

{ "title" : "MongoDB Guide" }

{ "title" : "NoSQL Database" }

{ "title" : "Python Quick Guide" }

{ "title" : "MongoDB Overview" }

> db.mycol.find({}, {'title':1,'by':1, 'url':1})

{ "\_id" : 101, "title" : "MongoDB Guide", "by" : "yiibai tutorials", "url" : "http://www.yiibai.com" }

{ "\_id" : 102, "title" : "NoSQL Database", "by" : "yiibai tutorials", "url" : "http://www.yiibai.com" }

{ "\_id" : 104, "title" : "Python Quick Guide", "by" : "yiibai tutorials", "url" : "http://www.yiibai.com" }

{ "\_id" : 100, "title" : "MongoDB Overview", "by" : "yiibai tutorials", "url" : "http://www.yiibai.com" }

请注意，在执行find()方法时，始终都会显示\_id字段，如果不想要此字段，则需要将其设置为0。

## 限制记录数

### MongoDB limit()方法

要限制 MongoDB 中返回的记录数，需要使用limit()方法。 该方法接受一个数字类型参数，它是要显示的文档数。

**语法**

limit()方法的基本语法如下：

> db.COLLECTION\_NAME.find().limit(NUMBER)

**示例**

假设集合myycol有以下数据。

> db.mycol.find({},{'\_id':1, 'title':1})

{ "\_id" : 101, "title" : "MongoDB Guide" }

{ "\_id" : 102, "title" : "NoSQL Database" }

{ "\_id" : 104, "title" : "Python Quick Guide" }

{ "\_id" : 100, "title" : "MongoDB Overview" }

以下示例将在查询文档时仅显示两个文档。

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

{ "title" : "MongoDB Guide" }

{ "title" : "NoSQL Database" }

如果没有在limit()方法中指定number参数的值，那么它将显示集合中的所有文档。

### MongoDB skip()方法

除了limit()方法之外，还有一个方法skip()也接受数字类型参数，用于跳过文档数量。

**语法**

skip()方法的基本语法如下 -

>db.COLLECTION\_NAME.find().limit(NUMBER).skip(NUMBER)

**示例**

以下示例将仅显示第三个文档。

> db.mycol.find({},{"title":1,\_id:0}).limit(1).skip(2)

{ "title" : "Python Quick Guide" }

请注意，skip()方法中的默认值为0。

## 排序文档

### MongoDB sort()方法

要在MongoDB中排序文档，需要使用sort()方法。 该方法接受包含字段列表及其排序顺序的文档。使用指定排序顺序1和-1。 1用于升序，而-1用于降序。

**语法**

sort()方法的基本语法如下：

>db.COLLECTION\_NAME.find().sort({KEY:1})

**示例**

假设集合myycol有以下数据。

> db.mycol.find({},{'\_id':1, 'title':1})

{ "\_id" : 101, "title" : "MongoDB Guide" }

{ "\_id" : 102, "title" : "NoSQL Database" }

{ "\_id" : 104, "title" : "Python Quick Guide" }

{ "\_id" : 100, "title" : "MongoDB Overview" }

以下示例将按标题降序排序显示文档。

> ## 按`title`降序排序

> db.mycol.find({},{"title":1,\_id:0}).sort({"title":-1})

{ "title" : "Python Quick Guide" }

{ "title" : "NoSQL Database" }

{ "title" : "MongoDB Overview" }

{ "title" : "MongoDB Guide" }

> ## 按`title`升序排序

> db.mycol.find({},{"title":1,\_id:0}).sort({"title":1})

{ "title" : "MongoDB Guide" }

{ "title" : "MongoDB Overview" }

{ "title" : "NoSQL Database" }

{ "title" : "Python Quick Guide" }

以下示例将按“\_id”降序和升序排序显示文档。

> 按“\_id”升序排序

> db.mycol.find({},{"title":1,\_id:1}).sort({"\_id":1})

{ "\_id" : 100, "title" : "MongoDB Overview" }

{ "\_id" : 101, "title" : "MongoDB Guide" }

{ "\_id" : 102, "title" : "NoSQL Database" }

{ "\_id" : 104, "title" : "Python Quick Guide" }

> # 按“\_id”降序排序

> db.mycol.find({},{"title":1,\_id:1}).sort({"\_id":-1})

{ "\_id" : 104, "title" : "Python Quick Guide" }

{ "\_id" : 102, "title" : "NoSQL Database" }

{ "\_id" : 101, "title" : "MongoDB Guide" }

{ "\_id" : 100, "title" : "MongoDB Overview" }

# MangoDB进阶

## MongoDB索引

索引支持查询的有效地提高效率。没有索引，MongoDB必须扫描集合的每个文档，以选择与查询语句匹配的文档。这种扫描效率很低，需要 MongoDB 处理大量的数据。

索引是特殊的数据结构，以易于遍历的形式存储数据集的一小部分。 索引存储特定字段或一组字段的值，按照索引中指定的字段值排序。

### ensureIndex()方法

要创建索引，需要使用MongoDB的ensureIndex()方法。

**语法**

ensureIndex()方法的基本语法如下：

>db.COLLECTION\_NAME.ensureIndex({KEY:1})

这里的key是要在其上创建索引的字段的名称，1是升序。 要按降序创建索引，需要使用-1。

**示例**

>db.mycol.ensureIndex({"title":1})

在ensureIndex()方法中，可以传递多个字段，以在多个字段上创建索引。

>db.mycol.ensureIndex({"title":1,"description":-1})

ensureIndex()方法也接受选项列表(可选)。 以下是列表：

| **参数** | **类型** | **描述** |
| --- | --- | --- |
| background | Boolean | 在后台构建索引，以便构建索引不会阻止其他数据库活动，则指定background的值为true。默认值为false。 |
| unique | Boolean | 创建一个唯一的索引，使得集合不会接受索引键或键匹配索引中现有值的文档的插入。 指定true以创建唯一索引。 默认值为false。 |
| name | String | 索引的名称。如果未指定，则MongoDB通过连接索引字段的名称和排序顺序来生成索引名称。 |
| dropDups | Boolean | 在可能有重复项的字段上创建唯一索引。MongoDB仅索引第一次出现的键，并从集合中删除包含该键的后续出现的所有文档。指定true以创建唯一索引。 默认值为false。 |
| sparse | Boolean | 如果为true，则索引仅引用具有指定字段的文档。这些索引在某些情况下(特别是排序)使用的空间较小，但行为不同。默认值为false。 |
| expireAfterSeconds | integer | 指定一个值(以秒为单位)作为TTL，以控制MongoDB在此集合中保留文档的时间。 |
| v | 索引版本 | 索引版本号。默认索引版本取决于创建索引时运行的MongoDB的版本。 |
| weights | 文档 | 权重是从1到99999之间的数字，并且表示该字段相对于其他索引字段在分数方面的意义。 |
| default\_language | String | 对于文本索引，确定停止词列表的语言以及句柄和分词器的规则。 默认值为英文。 |
| language\_override | String | 对于文本索引，要指定文档中包含覆盖默认语言的字段名称。默认值为language。 |

## MongoDB聚合

聚合操作处理数据记录并返回计算结果。聚合操作将多个文档中的值组合在一起，并可对分组数据执行各种操作，以返回单个结果。在SQL中的count(\*)与group by组合相当于mongodb中的聚合功能。

### aggregate()方法

对于MongoDB中的聚合，应该使用aggregate()方法。

**语法**

aggregate()方法的基本语法如下：

>db.COLLECTION\_NAME.aggregate(AGGREGATE\_OPERATION)

**示例**

假设在集合中，有以下数据：

db.article.insert([

{

\_id: 100,

title: 'MongoDB Overview',

description: 'MongoDB is no sql database',

by\_user: 'Maxsu',

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

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

likes: 100

},

{

\_id: 101,

title: 'NoSQL Overview',

description: 'No sql database is very fast',

by\_user: 'Maxsu',

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

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

likes: 10

},

{

\_id: 102,

title: 'Neo4j Overview',

description: 'Neo4j is no sql database',

by\_user: 'Kuber',

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

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

likes: 750

},

{

\_id: 103,

title: 'MySQL Overview',

description: 'MySQL is sql database',

by\_user: 'Curry',

url: 'http://www.yiibai.com/mysql/',

tags: ['MySQL', 'database', 'SQL'],

likes: 350

}])

现在从上面的集合中，如果要显示一个列表，说明每个用户写入了多少个教程，那么可使用以下aggregate()方法：

> db.article.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : 1}}}])

{ "\_id" : "Curry", "num\_tutorial" : 1 }

{ "\_id" : "Kuber", "num\_tutorial" : 1 }

{ "\_id" : "Maxsu", "num\_tutorial" : 2 }

对于上述用例的Sql等效查询是：

select by\_user, count(\*) as num\_tutorial from `article` group by by\_user;

在上面的例子中，我们按字段by\_user分组了文档，并且每次发生的by\_user的前一个值的值都被递增。以下是可用聚合表达式的列表。

| **表达式** | **描述** | **示例** |
| --- | --- | --- |
| $sum | 从集合中的所有文档中求出定义的值。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : "$likes"}}}]) |
| $avg | 计算集合中所有文档的所有给定值的平均值。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$avg : "$likes"}}}]) |
| $min | 从集合中的所有文档获取相应值的最小值。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$min : "$likes"}}}]) |
| $max | 从集合中的所有文档获取相应值的最大值。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$max : "$likes"}}}]) |
| $push | 将值插入到生成的文档中的数组中。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", url : {$push: "$url"}}}]) |
| $addToSet | 将值插入生成的文档中的数组，但不会创建重复项。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", url : {$addToSet : "$url"}}}]) |
| $first | 根据分组从源文档获取第一个文档。 通常情况下，这只适用于以前应用的“$sort”阶段。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", first\_url : {$first : "$url"}}}]) |
| $last | 根据分组从源文档获取最后一个文档。通常情况下，这只适用于以前应用的“$sort”阶段。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", last\_url : {$last : "$url"}}}]) |

### 管道概念

在UNIX命令中，shell管道可以对某些输入执行操作，并将输出用作下一个命令的输入。 MongoDB也在聚合框架中支持类似的概念。每一组输出可作为另一组文档的输入，并生成一组生成的文档(或最终生成的JSON文档在管道的末尾)。这样就可以再次用于下一阶段等等。

以下是在聚合框架可能的阶段：

* $project：用于从集合中选择一些特定字段。
* $match：这是一个过滤操作，因此可以减少作为下一阶段输入的文档数量。
* $group：这是上面讨论的实际聚合。
* $sort：排序文档。
* $skip：通过这种方式，可以在给定数量的文档的文档列表中向前跳过。
* $limit：限制从当前位置开始的给定数量的文档数量。
* $unwind：用于展开正在使用数组的文档。使用数组时，数据是预先加入的，此操作将被撤销，以便再次单独使用文档。 因此，在这个阶段，将增加下一阶段的文件数量。

## MongoDB复制

复制是跨多个服务器同步数据的过程。复制提供冗余，并通过不同数据库服务器上的多个数据副本增加数据可用性。 复制保护数据库免受单个服务器的丢失。 复制还允许从硬件故障和服务中断中恢复。 使用其他数据副本，可以将其专用于灾难恢复，报告或备份。

### 为什么复制？

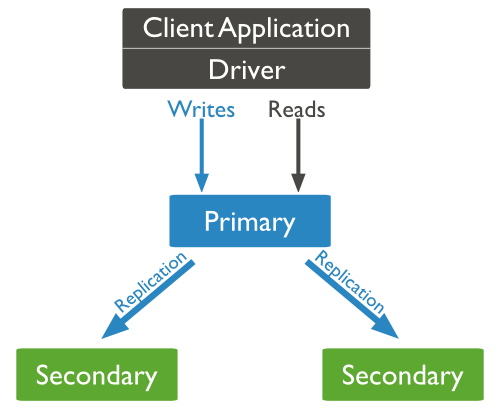
* 保持数据安全
* 数据的高可用性(24 \* 7)
* 灾难恢复
* 维护无停机(如备份，索引重建，压缩)
* 读取缩放(额外的副本可读)
* 副本集对应用程序是透明的

### MongoDB复制的工作原理

MongoDB通过使用副本集来实现复制。副本集是托管相同数据集的一组 mongod 实例。 在一个副本中，一个节点是接收所有写操作的主节点。所有其他实例(例如辅助节点)都应用主节点的操作，以便它们具有相同的数据集。 副本集可以只有一个主节点。

* 副本集是一组两个或多个节点(通常最少需要3个节点)。
* 在副本集中，一个节点是主节点，其余节点是次要节点。
* 所有数据从主节点复制到辅助节点。
* 在自动故障切换或维护时，选择为主节点建立，并选择新的主节点。
* 恢复故障节点后，它再次加入副本集，并作为辅助节点。

显示了MongoDB复制的典型图，客户端应用程序始终与主节点进行交互，然后主节点将数据复制到辅助节点。



### 副本集功能

* N个节点的集群
* 任何一个节点都可以是主节点
* 所有写入操作都转到主节点操作
* 自动故障切换
* 自动恢复
* 共识一般选举

### 设置副本集

在本教程中，我们将独立的 MongoDB 实例转换为副本集。要转换为副本集，以下是步骤：

* 关机正在运行 MongoDB 服务器。
* 通过指定 - replSet选项启动 MongoDB服 务器。 以下是--replSet的基本语法：

mongod --port "PORT" --dbpath "YOUR\_DB\_DATA\_PATH" --replSet "REPLICA\_SET\_INSTANCE\_NAME"

**示例**

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

* 它在端口27017上启动名称为rs0的 mongod 实例。
* 现在启动命令提示符并连接到这个 mongod 实例。
* 在Mongo客户端中，发出命令rs.initiate()以启动新的副本集。
* 要检查副本集配置，可使用命令rs.conf()。 要检查复制集的状态，请使用命令rs.status()。

### 将会员添加到副本集

要将成员添加到副本集，请在多台计算机上启动 mongod 实例。 现在启动一个 mongo 客户端并发出一个命令rs.add()。

**语法**

rs.add()命令的基本语法如下：

>rs.add(HOST\_NAME:PORT)

**示例**

假设您的 mongod 实例名称是 mongod1.net，它在端口 27017 上运行。要将此实例添加到副本集，请在 Mongo 客户端中发出命令 rs.add()。

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

只能在连接到主节点时，将 mongod 实例添加到副本集。要检查是否连接到主服务器，请在 mongo 客户端中发出命令db.isMaster()。

rs0:PRIMARY> db.isMaster()

{

"hosts" : [

"ubuntu:27017"

],

"setName" : "rs0",

"setVersion" : 1,

"ismaster" : true,

"secondary" : false,

"primary" : "ubuntu:27017",

"me" : "ubuntu:27017",

"electionId" : ObjectId("7fffffff0000000000000001"),

"lastWrite" : {

"opTime" : {

"ts" : Timestamp(1498896581, 1),

"t" : NumberLong(1)

},

"lastWriteDate" : ISODate("2017-07-01T08:09:41Z")

},

"maxBsonObjectSize" : 16777216,

"maxMessageSizeBytes" : 48000000,

"maxWriteBatchSize" : 1000,

"localTime" : ISODate("2017-07-01T08:09:50.365Z"),

"maxWireVersion" : 5,

"minWireVersion" : 0,

"readOnly" : false,

"ok" : 1

}

## MongoDB分片

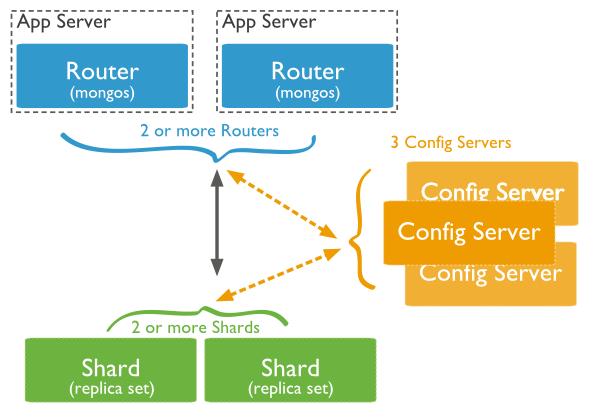
分片是在多台机器之间存储数据记录的过程，MongoDB是满足数据增长需求的方法。 随着数据的大小增加，单个机器可能不足以存储所有数据，也不能提供可接受的读写吞吐量。 分片解决了水平缩放的问题。 使用分片，可以添加更多的机器来支持数据增长和读写操作的需求。

### 为什么要分片？

* 在复制中，所有写入都转到主节点
* 延迟敏感查询到主节点
* 单个副本集合有12个节点的限制
* 当活动数据集较大时，内存不足够大
* 本地磁盘不够大
* 垂直扩缩太昂贵了

### MongoDB中的分片

下图显示了使用分片集群的 MongoDB 分片。



在上图中，有三个主要组件：

* **碎片(Shards)**：碎片用于存储数据。它们提供高可用性和数据一致性。在生产环境中，每个分片是一个单独的副本集。
* **配置服务器(Config Servers)**：配置服务器存储集群的元数据。该数据包含集群的数据集与分片的映射。查询路由器使用此元数据将操作定位到特定的分片。在生产环境中，分片集群正好有3个配置服务器。
* **查询路由器(Query Routers)**：查询路由器基本上是 mongo 实例，与客户端应用程序的接口和直接操作到适当的分片。 查询路由器处理并将操作定向到碎片，然后将结果返回给客户端。 分片集群可以包含多个查询路由器来分割客户端请求负载。客户端向一个查询路由器发送请求。一般来说，分片集群有许多查询路由器。

## MongoDB备份与恢复

### 导出转储MongoDB数据

要在MongoDB中创建数据库备份，应该使用mongodump命令。此命令将导出转储服务器的整个数据到转储目录。有许多选项可用于限制数据量或创建远程服务器的备份。

**语法**

mongodump命令的基本语法如下：

> mongodump

**示例**

启动 mongod 服务器 假设您的 mongod 服务器正在本地主机和端口 27017 上运行，请打开命令提示符并转到 mongodb 实例的 bin 目录(如示例安装路径：D:\Program Files\MongoDB\Server\3.4\bin)，然后键入命令：mongodump。

考虑 mycol 集合具有以下数据：

> db.mycol.find({}, {"\_id":1, "title":1})

{ "\_id" : 101, "title" : "MongoDB Guide" }

{ "\_id" : 102, "title" : "NoSQL Database" }

{ "\_id" : 104, "title" : "Python Quick Guide" }

{ "\_id" : 100, "title" : "MongoDB Overview" }

现在使用以下命令，创建备份：

> mongodump

该命令将连接到运行在 127.0.0.1 和端口 27017 的服务器，并将服务器的所有数据恢复到目录/bin/dump/。 以下是命令的输出：

yiibai@ubuntu:~$ mongodump

2017-07-02T17:31:51.115-0700 writing admin.system.version to

2017-07-02T17:31:51.118-0700 done dumping admin.system.version (1 document)

2017-07-02T17:31:51.119-0700 writing test.inventory to

2017-07-02T17:31:51.119-0700 writing test.article to

2017-07-02T17:31:51.120-0700 writing test.mycol to

2017-07-02T17:31:51.121-0700 done dumping test.inventory (5 documents)

2017-07-02T17:31:51.122-0700 done dumping test.article (4 documents)

2017-07-02T17:31:51.122-0700 done dumping test.mycol (4 documents)

yiibai@ubuntu:~$

此时你可能想知道，上面导出的备份文件放到什么地方了？ 默认情况下，MongoDB 会在当前目录下创建一个dump目录，并把所有的数据库按数据库名称创建目录。在这个实例中，有两数据库admin和test，那么它将创建两个目录。

**怎么样知道 MongoDB 文件的位置？**

对于大部分软件，尤其是Linux平台上的软件，都有一个相关的配置文件，因此任何的设置选项都可以从这个文件中找到。配置文件的一般在/etc目录下，所以mongodb的配置文件在/etc/mongod.conf，mongod.conf配置的内容如下：

# mongod.conf

# for documentation of all options, see:

# http://docs.mongodb.org/manual/reference/configuration-options/

# Where and how to store data.

storage:

dbPath: /var/lib/mongodb

journal:

enabled: true

# engine:

# mmapv1:

# wiredTiger:

# where to write logging data.

systemLog:

destination: file

logAppend: true

path: /var/log/mongodb/mongod.log

# network interfaces

net:

port: 27017

bindIp: 127.0.0.1

#processManagement:

#security:

#operationProfiling:

#replication:

#sharding:

## Enterprise-Only Options:

#auditLog:

#snmp:

以下是可用于 mongodump 命令的可用选项的列表。

| **语法** | **描述** | **示例** |
| --- | --- | --- |
| mongodump —host HOST\_NAME —port PORT\_NUMBER | 此命令将备份指定的 mongod 实例的所有数据库。 | mongodump --host 127.0.0.1 --port 27017 |
| mongodump —out BACKUP\_DIRECTORY | 此命令将仅在指定路径上备份数据库。 | mongodump --out /home/yiibai/mongobak |
| mongodump —collection COLLECTION —db DB\_NAME | 此命令将仅备份指定数据库的指定集合。 | mongodump --collection mycol --db test |

### 恢复数据

要恢复备份数据，使用MongoDB的 mongorestore 命令。 此命令从备份目录中恢复所有数据。

**语法**

mongorestore命令的基本语法是：

> mongorestore

在恢复数据之前，先删除当前数据库的部分数据，以演示导入恢复数据后可以查询到备份时的数据。

> db.mycol.remove({})

WriteResult({ "nRemoved" : 4 })

>

> db.mycol.find({})

执行恢复命令后，重新查询数据 -

> db.mycol.find({}, {"title":1})

{ "\_id" : 101, "title" : "MongoDB Guide" }

{ "\_id" : 102, "title" : "NoSQL Database" }

{ "\_id" : 104, "title" : "Python Quick Guide" }

{ "\_id" : 100, "title" : "MongoDB Overview" }

## MongoDB部署

当准备MongoDB部署时，应该尝试了解应用程序如何在生产中的作用和位置。 开发一种一致，可重复的方法来管理部署环境是一个好办法，以便在生产过程中尽可能减少错误。

最好的方法包括原型设置，进行负载测试，监控关键指标，并使用该信息来扩展设置。 该方法的关键部分是主动监控整个系统 - 这将有助于了解生产系统在部署前如何维持，并确定需要添加容量的位置。 例如，了解内存使用中的潜在峰值时可能有助于在写入之前锁定。

为了监控部署，MongoDB提供了以下一些命令：

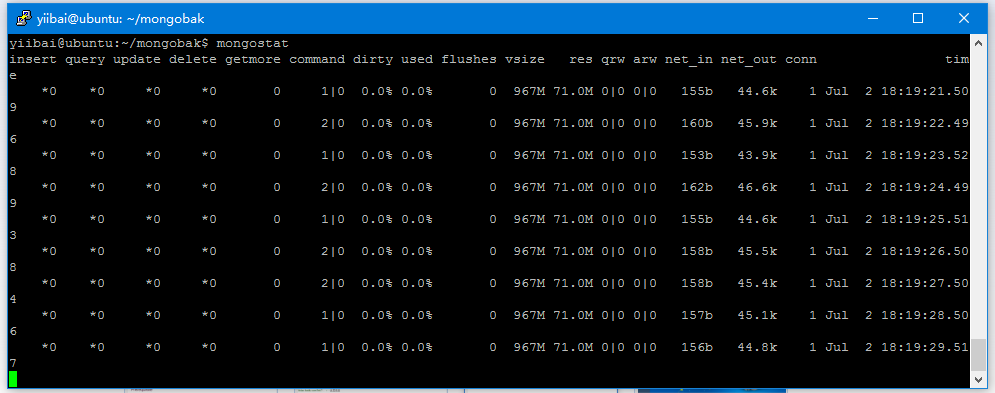
* **mongostat**

此命令检查所有运行的mongod实例的状态，并返回数据库操作的计数器。 这些计数器包括插入，查询，更新，删除和游标。 命令还显示遇到页面错误，并显示锁定百分比。这可以用来监控内存不足，写入容量或出现性能问题。

要运行命令，首先要启动mongod实例。 在另一个命令提示符下，转到 mongodb 安装的bin目录，然后键入：mongostat。

D:\Program Files\MongoDB\Server\3.4\bin> mongostat

以下是命令的输出：



* **mongotop**

此命令跟踪并报告基于集合的 MongoDB 实例的读写活动。 默认情况下，mongotop会在每秒钟内返回信息，但是可相应地更改信息。应该检查此读写活动是否符合您的应用意图，并且要一次对数据库发出太多的写入操作，从磁盘读取的频率太高，或者超出了工作集合大小。

要运行命令，请启动 mongod 实例。在另一个命令提示符下，转到mongodb安装的bin目录，然后键入：mongotop。

D:\Program Files\MongoDB\Server\3.4\bin> mongotop

以下是命令的输出 -

yiibai@ubuntu:~/mongobak$ mongotop

2017-07-02T18:27:23.485-0700 connected to: 127.0.0.1

ns total read write 2017-07-02T18:27:24-07:00

admin.system.roles 0ms 0ms 0ms

admin.system.version 0ms 0ms 0ms

local.startup\_log 0ms 0ms 0ms

local.system.replset 0ms 0ms 0ms

test.article 0ms 0ms 0ms

test.inventory 0ms 0ms 0ms

test.mycol 0ms 0ms 0ms

ns total read write 2017-07-02T18:27:25-07:00

admin.system.roles 0ms 0ms 0ms

admin.system.version 0ms 0ms 0ms

local.startup\_log 0ms 0ms 0ms

local.system.replset 0ms 0ms 0ms

test.article 0ms 0ms 0ms

test.inventory 0ms 0ms 0ms

test.mycol 0ms 0ms 0ms

要更改 mongotop 命令以更多少时间间隔返回信息，请在 mongotop 命令之后指定一个特定的数字值。假设每隔 5 秒返回一次信息：

D:\Program Files\MongoDB\Server\3.4\bin> mongotop 5

除了MongoDB工具外，10gen 还提供了一个免费的托管监控服务MongoDB管理服务(MMS)，它提供了一个仪表盘，可以让您了解整个集群的指标。

# MangoDB高级应用

## MongoDB关联关系

MongoDB中的关系表示各个文档在逻辑上的相互关联。关系可以通过嵌入式和引用方法建模。 这种关系可以是1：1，1：N，N：1或N：N。

假设有一种情况：要存储用户的地址。一个用户可以拥有多个地址，这就是1：N关系。

以下是用户(user)文档示例的文档结构：

{

"\_id":10999110,

"name": "Maxsu",

"contact": "13888990021",

"dob": "1992-10-11"

}

以下是地址(address)文档的示例文档结构：

{

"\_id":12200,

"building": "Hainan Building NO.2100",

"pincode": 571100,

"city": "Haikou",

"province": "Hainan"

}

### 嵌入式关系建模

在嵌入式方法中，我们将地址(address)文档嵌入到用户(user)文档中。

{

"\_id": 21000100,

"contact": "13800138000",

"dob": "1991-11-11",

"name": "Maxsu",

"address": [

{

"building": "Hainan Building NO.2100",

"pincode": 571100,

"city": "Haikou",

"province": "Hainan"

},

{

"building": "Sanya Building NO.2100",

"pincode": 572200,

"city": "Sanya",

"province": "Hainan"

},

]

}

该方法将所有相关数据保存在单个文档中，这使得检索和维护更容易。可以使用单个查询来在整个文档检索，例如：

> db.users.findOne({"name":"Maxsu"},{"address":1, "name":1})

请注意，在上述查询中，db和users分别是数据库和集合。缺点是如果嵌入式文档的大小如果不断增长，可能会影响读/写性能。

### 建模参考关系

这是设计规范化关系的方法。 在这种方法中，用户和地址文件将分别维护，但用户文档将包含一个将引用地址文档的id字段的字段。

{

"\_id":ObjectId("52ffc33321332111sdfaf"),

"contact": "13800138000",

"dob": "1991-11-11",

"name": "Maxsu",

"address\_ids": [

ObjectId("123123"),

ObjectId("123412")

]

}

如上所示，用户文档包含对应地址的ObjectId的数组字段address\_ids。 使用这些ObjectIds，我们可以从那里查询地址文件并获取地址详细信息。 使用这种方法，需要两个查询：首先从用户文档获取address\_ids字段，然后从地址集中获取这些地址。

>var result = db.users.findOne({"name":"Maxsu"},{"address\_ids":1})

>var addresses = db.address.find({"\_id":{"$in":result["address\_ids"]}})

## MongoDB数据库引用

如MongoDB关系的最后一章所示，要在MongoDB中实现规范化的数据库结构，我们使用引用关系的概念，也称为手动引用，将引用的文档的id手动存储在其他文档中。 但是，如果文档包含来自不同集合的引用，可以使用MongoDB DBRefs。

### DBRefs vs 手动参考

考虑一个示例场景，在这个场景中将使用DBRefs而不是手动引用，设计有一个数据库中将不同类型的地址(家庭，办公室，邮件等)存储在不同的集合(address\_home，address\_office，address\_mailing等)中。 现在，当用户集合的文档引用地址时，还需要根据地址类型指定要查找的集合。在这种情况下，文档引用了多个集合中的文档，则应该使用DBRefs。

### 使用DBRefs

DBRefs中有三个字段：

* $ref：此字段指定引用文档的集合。
* $id：此字段指定引用文档的\_id字段。
* $db：这是一个可选字段，并包含引用文档所在的数据库的名称。

假设一个具有DBRef字段address的示例用户文档，如代码片段所示：

{

"\_id":ObjectId("348362491fjaskdlf2314"),

"address": {

"$ref": "address\_home",

"$id": ObjectId("sfaafdf4137832149fssa"),

"$db": "yiibai"},

"contact": "13800138000",

"dob": "1991-12-12",

"name": "Maxsu"

}

这里的 DBRef 字段 address指定引用的地址文件位于yiibai数据库中的address\_home集合中，其ID为sfaafdf4137832149fssa。

以下代码由$ref参数(在示例中为address\_home)指定的集合中动态地查找ID为DBRef中的$id参数指定的文档。

>var user = db.users.findOne({"name":"Maxsu"})

>var dbRef = user.address

>db[dbRef.$ref].findOne({"\_id":(dbRef.$id)})

以上代码返回address\_home集合中存在的以下address文档中：

{

"\_id" : ObjectId("sfaafdf4137832149fssa"),

"building" : "Hainan Apt No.2100",

"pincode" : 571100,

"city" : "Haikou",

"province" : "Hainan"

}

## MongoDB覆盖查询

### 什么是覆盖查询？

根据官方MongoDB文档，覆盖查询是一个查询，其中：

* 查询中的所有字段都是索引的一部分。
* 查询中返回的所有字段都在同一个索引中。

由于查询中存在的所有字段都是索引的一部分，所以MongoDB查询指定条件匹配，并使用相同的索引返回结果，而不会实际查找文档。由于索引存在于内存中，与通过扫描文档获取数据相比，从索引获取数据更快。

### 使用覆盖查询

要测试覆盖的查询，请在users集合中考虑以下文档：

{

"\_id": ObjectId("234324fd852426020001111"),

"contact": "13800138000",

"dob": "1991-11-11",

"gender": "M",

"name": "Maxsu",

"user\_name": "user\_maxsu"

}

首先使用以下查询在users集合 的字段：gender 和 user\_name 上创建一个复合索引：

> db.users.ensureIndex({gender:1,user\_name:1})

现在，这个索引将覆盖以下查询：

>db.users.find({gender:"M"},{user\_name:1,\_id:0})

也就是说，对于上述查询，MongoDB不会查找数据库文档。相反，它将从索引数据中获取所需的数据。

由于索引不包括\_id字段，在上面示例中，已经将其从查询的结果集中明确排除，因为MongoDB默认情况下在每个查询中返回\_id字段。 所以下面的查询不会在上面创建的索引中被覆盖(查询有返回\_id字段)：

>db.users.find({gender:"M"},{user\_name:1})

最后，请记住，如果有下列情况，则索引无法覆盖查询：

* 任何索引的字段是一个数组。
* 任何索引的字段是一个子文档。

## MongoDB分析查询

分析查询是衡量数据库和索引设计的有效性的一个非常重要的方式。在这里我们将介绍两个经常使用的$explain和$hint查询。

### 使用 $explain 操作符

$explain操作符提供有关查询的信息，查询中使用的索引和其他统计信息。它在在分析索引优化情况时非常有用。

在最后一章中，我们已经使用以下查询在users集合的字段：gender和user\_name上创建了一个索引：

>db.users.ensureIndex({gender:1,user\_name:1})

现在将在以下查询中使用$explain：

>db.users.find({gender:"M"},{user\_name:1,\_id:0}).explain()

上述explain()查询返回以下分析结果：

{

"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

}

]

]

}

}

现在将看看这个结果集中的字段：

* indexOnly的true值表示此查询已使用索引。
* cursor字段指定使用的游标的类型。BTreeCursor类型表示使用了索引，并且还给出了使用的索引的名称。 BasicCursor表示完全扫描，而不使用任何索引的情况。
* n表示返回的文档数。
* nscannedObjects表示扫描的文档总数。
* nscanned表示扫描的文档或索引条目的总数。

### 使用 $hint

$hint操作符强制查询优化器使用指定的索引来运行查询。当要测试具有不同索引的查询的性能时，这就特别有用了。例如，以下查询指定要用于此查询的gender和user\_name字段的索引：

> db.users.find({gender:"M"},{user\_name:1,\_id:0}).hint({gender:1,user\_name:1})

要使用$explain来分析上述查询：

>db.users.find({gender:"M"},{user\_name:1,\_id:0}).hint({gender:1,user\_name:1}).explain()

## MongoDB原子操作

MongoDB不支持多文档原子事务。但是，它可以为单个文档提供了原子操作。因此，如果文档有一百个字段，则更新语句将更新或不更新所有字段的值，因此在原始级别保持原子性。

### 原子操作模型数据

维持原子性的推荐方法是将所有相关信息保存在一起，并使用嵌入式文档在一个文档中一起更新。这将确保单个文档的所有更新都是原子的。

考虑以下产品文件：

{

"\_id":1,

"product\_name": "Huawei P9",

"category": "mobiles",

"product\_total": 5,

"product\_available": 3,

"product\_bought\_by": [

{

"customer": "Kobe",

"date": "2017-07-08"

},

{

"customer": "Maxsu",

"date": "2018-07-28"

}

]

}

在上面这个文档中，已经在product\_bought\_by字段中嵌入了购买产品的客户的信息。现在，当有新客户购买产品，首先查看product\_available字段检查产品存货是否仍然够用。

如果可用，则减少product\_available字段的值，并将新客户的嵌入式文档插入到product\_bought\_by字段中。下面将使用findAndModify命令来执行此功能，因为它会以同样的方式搜索和更新文档。

>db.products.findAndModify({

query:{\_id:2,product\_available:{$gt:0}},

update:{

$inc:{product\_available:-1},

$push:{product\_bought\_by:{customer:"Curry",date:"2017-08-08"}}

}

})

嵌入式文档和使用findAndModify查询的方法确保产品购买信息仅在产品可用时才更新。 而整个这个事务在同一个查询中是原子的。

与此相反的是如果分别保留产品数量，以及谁购买产品的信息。在这种情况下，我们将首先使用第一个查询检查产品是否可用。然后在第二个查询中更新购买信息。 但是，有可能在执行这两个查询时(还未执行完)，其他一些用户已经购买了该产品，并且此产品缺货了。但是由于程序执行过程中并不知晓，第二个查询将根据第一个查询的结果更新购买信息。这会导致数据库不一致，因为产品已经没有库存，但是仍然断续销售。

## MongoDB高级索引

在这一章节中，我们来学习高级索引，假设users集合的以下文档：

{

"address": {

"city": "Haikou",

"province": "Hainan",

"pincode": "123456"

},

"tags": [

"music",

"cricket",

"blogs"

],

"name": "Maxsu"

}

上述文档包含地址子文档和标签数组。

### 索引数组字段

假设要根据用户的标签搜索用户文档。为此，我们将在集合中的tags数组上创建一个索引。

在数组上创建一个索引依次为每个字段创建单独的索引条目。所以在这个例子中，当在tags数组上创建一个索引时，将为其值music，cricket和blogs创建单独的索引。

要在tags数组上创建索引，请使用以下代码：

>db.users.ensureIndex({"tags":1})

创建索引后，可以搜索这个集合的标签字段：

>db.users.find({tags:"cricket"})

要验证是否使用正确的索引，请使用以下说明命令：

>db.users.find({tags:"cricket"}).explain()

上面的命令生成：“cursor”：“BtreeCursor tags\_1”，它确认使用了正确的索引。

### 索引子文档字段

假设要搜索基于city，province和pincode字段的文档。 由于所有这些字段都是地址子文档字段的一部分，因此将在子文档的所有字段上创建一个索引。

要在子文档的所有三个字段上创建索引，请使用以下代码：

>db.users.ensureIndex({"address.city":1,"address.state":1,"address.pincode":1})

创建索引后，就可以使用此索引来搜索任何子文档的字段了，如下所示：

>db.users.find({"address.city":"Haikou"})

请记住，查询表达式必须遵循指定的索引的顺序。 所以上面创建的索引将支持以下查询：

>db.users.find({"address.city":"Haikou","address.province":"Hainan"})

它还将支持以下查询 -

>db.users.find({"address.city":"Haikou","address.province":"Hainan",

"address.pincode":"12345"})

## MongoDB索引限制

### 索引额外开销

每个索引占用一些空间，并导致每次插入，更新和删除的开销。因此，如果很少使用集合进行读取操作(大部分是插入或更新操作)，则建议不要使用索引。

### 内存的使用

由于索引存储在RAM中，因此应确保索引的总大小不超过RAM限制。 如果总大小超过了系统内存的大小，MongoDB将开始删除一些索引，从而导致性能下降。

### 查询限制

索引不能在使用的查询中使用：

* 正则表达式或否定运算符，如$nin，$not等
* 算术运算符，如$mod等
* $where子句

因此，建议经常检查查询的索引使用情况。

### 索引键限制

从*MongoDB 2.6*版本开始，如果现有索引字段的值超过索引键限制，MongoDB将不会创建索引。

### 插入超过索引键限制的文档

如果本文档的索引字段值超过索引键限制，MongoDB将不会将任何文档插入索引集合。 mongorestore和mongoimport工具也是如此。

### 索引最大范围

* 集合不能有超过64个索引。
* 索引名称的长度不能超过125个字符。
* 复合索引最多可以编号31个字段。

## MongoDB对象ID（ObjectId）

*ObjectId* 是具有以下结构的12字节BSON类型：

* 前4个字节表示从unix时间开始的秒数
* 接下来的3个字节是机器标识符
* 接下来的2个字节由进程ID组成
* 最后3个字节是随机计数器值

MongoDB使用ObjectIds作为每个文档的\_id字段的默认值，这是在创建任何文档时生成的。 ObjectId的复杂组合使得所有\_id字段都是唯一的。

>newObjectId = ObjectId()

上述语句返回以下唯一生成的id：

ObjectId("595b99d9f6a6243715b3c312")

还可以提供一个12字节的 id 来可代替 MongoDB 生成 ObjectId：

>myObjectId = ObjectId("595b99d9f6a6243715b3c312")

### 创建文档的时间戳

由于\_id ObjectId默认存储4字节的时间戳，在大多数情况下，不需要存储任何文档的创建时间。可以使用getTimestamp方法获取文档的创建时间：

>ObjectId("595b99d9f6a6243715b3c312").getTimestamp()

这将以ISO日期格式返回此文档的创建时间：

> ObjectId("595b99d9f6a6243715b3c312").getTimestamp()

ISODate("2017-07-04T13:36:25Z")

### 将ObjectId转换为String

在某些情况下，可能需要使用字符串格式的ObjectId值。 要转换ObjectId为字符串，请使用以下代码：

>newObjectId.str

上面的代码将返回Guid的字符串格式：

595b99d9f6a6243715b3c312

## MongoDB Map Reduce

根据MongoDB文档的说明，*Map-reduce*是将大量数据合并为有用的聚合结果的数据处理范例。 MongoDB使用mapReduce命令进行map-reduce操作。MapReduce通常用于处理大型数据集。

### MapReduce命令

以下是基本 mapReduce 命令的语法：

>db.collection.mapReduce(

function() {emit(key,value);}, //map function

function(key,values) {return reduceFunction}, { //reduce function

out: collection,

query: document,

sort: document,

limit: number

}

)

map-reduce函数首先查询集合，然后将结果文档映射到发出的键值对，然后根据具有多个值的键进行减少。

在上面的语法：

* map是一个JavaScript函数，它将一个值与一个键映射并发出一个键值对；
* reduce是一个javascript功能，可以减少或分组具有相同键的所有文档；
* out指定map-reduce查询结果的位置；
* query指定选择文档的可选选择条件；
* sort指定可选的排序条件；
* limit指定可选的最大文档数；

### 使用MapReduce

请考虑存储用户帖子的以下文档结构。 该文档存储用户的user\_name和帖子的状态(status)。

{

"post\_text": "yiibai tutorials is an awesome website for tutorials",

"user\_name": "maxsu",

"status":"active"

}

现在，我们将在posts集上使用mapReduce函数来选择所有status的值为active的帖子，并根据user\_name分组，然后使用以下代码对每个用户的帖子数进行计数：

>db.posts.mapReduce(

function() { emit(this.user\_id,1); },

function(key, values) {return Array.sum(values)}, {

query:{status:"active"},

out:"post\_total"

}

)

以上mapReduce查询输出以下结果：

{

"result" : "post\_total",

"timeMillis" : 9,

"counts" : {

"input" : 4,

"emit" : 4,

"reduce" : 2,

"output" : 2

},

"ok" : 1,

}

结果表明，共有4个文档与查询(status的值为active)匹配，映射函数发出4个具有键值对的文档，最后将具有相同键的reduce函数分组的映射文档分解为2。

要查看此 mapReduce 查询的结果，请使用 find 运算符：

>db.posts.mapReduce(

function() { emit(this.user\_id,1); },

function(key, values) {return Array.sum(values)}, {

query:{status:"active"},

out:"post\_total"

}

).find()

上述查询给出以下结果，表明用户 tom 和 maxsu 有两个活动状态的帖子：

{ "\_id" : "tom", "value" : 2 }

{ "\_id" : "maxsu", "value" : 2 }

以类似的方式，MapReduce查询可用于构建大型复杂聚合查询。 使用自定义JavaScript函数可以使用MapReduce，它非常灵活和强大。

## MongoDB文本搜索

从*MongoDB 2.4*版开始，MongoDB开始支持文本索引来搜索字符串内容。文本搜索使用词法技术通过删除字符串字词，如a，an等等来查找字符串字段中的指定单词。目前，MongoDB支持大约15种语言。

### 启用文本搜索

最初，文本搜索是一个实验性功能，但从*MongoDB 2.4*版开始，默认情况下启用配置。 但是，如果使用*2.4*之前版本的MongoDB，则必须使用以下代码启用文本搜索：

>db.adminCommand({setParameter:true,textSearchEnabled:true})

### 创建文本索引

请考虑以下文档，其中包含帖子文字及其标签：

{

"post\_text": "enjoy the mongodb articles on yiibai tutorials",

"tags": [

"mongodb",

"yiibai tutorials"

]

}

我们将在post\_text字段上创建一个文本索引，以便可以在posts的文本中搜索：

>db.posts.ensureIndex({post\_text:"text"})

### 使用文本索引

现在我们已经在post\_text字段上创建了文本索引，下面在其文本中搜索所有带有“yiibai”一词的帖子。

>db.posts.find({$text:{$search:"yiibai"}})

上述命令返回以下结果文件，其文本中包含“yiibai”一词：

{

"\_id" : ObjectId("595b99e2f6a6243715b3c316"),

"post\_text" : "enjoy the mongodb articles on yiibai",

"tags" : [ "mongodb", "yiibai" ]

}

{

"\_id" : ObjectId("595b99e2f6a6243715b3c317"),

"post\_text" : "writing yiibai tutorials on mongodb",

"tags" : [ "mongodb", "tutorial" ]

}

如果您使用旧版本的MongoDB，则必须使用以下命令：

>db.posts.runCommand("text",{search:" yiibai "})

与正常搜索相比，使用文本搜索极大地提高了搜索效率。

### 删除文本索引

要删除现有的文本索引，首先使用以下查询找到索引的名称：

>db.posts.getIndexes()

从上面的查询获取索引的名称后，运行以下命令。 这里，post\_text\_text是索引的名称。

>db.posts.dropIndex("post\_text\_text")

## MongoDB正则表达式

正则表达式在所有的编程语言中经常使用，用于以搜索任何字符串中的模式或单词。 MongoDB还提供使用$regex运算符的字符串模式匹配的正则表达式的功能。 MongoDB使用PCRE(Perl兼容正则表达式)作为正则表达式语言。

与文本搜索不同，不需要使用任何配置或命令来使用正则表达式。

请考虑以下文档结构，其中包含post文本及其标签：

{

"post\_text": "enjoy the mongodb articles on yiibai tutorials",

"tags": [

"mongodb",

"yiibai"

]

}

### 使用正则表达式

以下regex查询搜索包含“yiibai”字符串的所有帖子：

>db.posts.find({post\_text:{$regex:"yiibai"}})

上面的查询也可以写成：

>db.posts.find({post\_text:/yiibai/})

### 使用不区分大小写的正则表达式

要使搜索不区分大小写，可指定$options参数的值为$i。 以下命令将寻找具有单词yiibai的字符串，不管其是小写还是大写字母：

>db.posts.find({post\_text:{$regex:"yiibai", $options:"$i"}})

此查询返回的结果之一是以下文档，其中包含大小写字母的“yiibai”单词：

{

"\_id" : ObjectId("595b99e1f6a6243715b3c316"),

"post\_text" : "hey! this is my post on Yiibai Yiibai",

"tags" : [ "yiibai" ]

},

{

"\_id" : ObjectId("595b99e1f6a6243715b3c317"),

"post\_text" : "hey! this is my post on yiiBai Yiibai",

"tags" : [ "yiibai" ]

}

### 对数组元素使用正则表达式

我们也可以在数组字段上使用正则表达式。当实现标签的功能时，这尤其重要。 所以，如果要从单词：tutorial(tutorial，tutorials , tutorialpoint 或 tutorialphp 或 tutorialpython 或 tutorialphp)开始搜索所有具有标签的帖子，可以使用以下代码：

> db.posts.find({tags:{$regex:"tutorial"}})

### 化正则表达式查询

* 如果文档字段创建了索引，则查询将使用索引值来匹配正则表达式。 与正常表达式扫描整个集合相比，使用索引值使得搜索非常快。
* 如果正则表达式是前缀表达式，则所有匹配都是以某个字符串字符开始。 例如，如果正则表达式是^tut，则查询必须仅搜索以tut开头的那些字符串。

## MongoDB GridFS

GridFS是用于存储和检索大型文件(如图像、音频文件、视频文件等)的MongoDB规范。它是一种用于存储文件的文件系统，但其数据存储在MongoDB集合中。GridFS存储文件可超过文件大小限制为16MB的功能。

GridFS将文件划分成块，并将每个数据块存储在单独的文档中，每个文件的最大大小为255k。

默认情况下，GridFS使用两个集合fs.files和fs.chunks来存储文件的元数据和块。每个块由其唯一的ObjectId字段\_id标识。fs.files作为父文档进行切换。fs.chunks文档中的files\_id字段将该块链接到其父块。

以下是fs.files集合的示例文档：

{

"filename": "test.txt",

"chunkSize": NumberInt(261120),

"uploadDate": ISODate("2018-04-13T11:32:33.557Z"),

"md5": "7b762939321e146569b07f72c62cca4f",

"length": NumberInt(646)

}

文档指定文件名，块大小，上传的日期和长度。

以下是fs.chunks文档的示例文档：

{

"files\_id": ObjectId("595b99d9f6a6243715b3c312"),

"n": NumberInt(0),

"data": "Mongo Binary Data"

}

### 将文件添加到GridFS

现在，使用put命令存储一个使用GridFS的mp3文件。 为此，我们将使用MongoDB安装文件夹的bin文件夹中出现的mongofiles.exe工具程序。

打开命令提示符，导航到MongoDB安装文件夹的bin文件夹中的mongofiles.exe，然后键入以下代码：

D:\Program Files\MongoDB\Server\3.4\bin> mongofiles.exe -d gridfs put song.mp3

这里，gridfs是要存储文件的数据库的名称。如果数据库不存在，MongoDB将自动创建一个新的文档。song.mp3是上传文件的名称。要在数据库中查看文件的文档，可以使用查询查询：

> db.fs.files.find()

上述命令返回以下文档：

{

\_id: ObjectId('595b99d9f6a6243715b3c316'),

filename: "song.mp3",

chunkSize: 381120,

uploadDate: new Date(1398391643474), md5: "e4f53379c909f7bed2e9d631e15c1c41",

length: 10401959

}

还可以使用以前查询中返回的文档ID，查看与存储文件相关的fs.chunks集合中存在的所有块，并使用以下代码：

>db.fs.chunks.find({files\_id:ObjectId('595b99d9f6a6243715b3c316')})

在这个示例中，查询返回50多个文件，表示整个mp3文件被分成50个数据块。

## MongoDB固定循环集合

MongoDB固定集合(**Capped collections**)是固定大小的循环集合，遵循插入顺序以支持创建，读取和删除操作的高性能。 通过循环，当分配给集合的固定大小被耗尽时，它将开始删除集合中最旧的文档，而不用提供任何明确的命令。

如果更新导致增加的文档大小，固定集合会限制文档的更新。 由于上限集合按照磁盘存储的顺序存储文档，因此可确保文档大小不会增加磁盘上分配的大小。固定集合最适用于存储日志信息，缓存数据或任何其他高容量数据。

### 创建固定集合

要创建一个固定集合，可使用 createCollection 命令，但是capped选项的值为true，并指定以字节为单位的最大集合大小。

>db.createCollection("cappedLogCollection",{capped:true,size:99999})

除了指定集合大小，还可以使用max参数限制集合中的文档数量：

>db.createCollection("cappedLogCollection",{capped:true,size:99999,max:1000})

如果要查看集合是否固定，请使用以下isCapped命令：

>db.cappedLogCollection.isCapped()

如果想要将一个集合转换为上限的集合，则可以使用以下代码进行操作：

>db.runCommand({"convertToCapped":"posts",size:99999})

此代码将现有的post转换为固定集合。

### 查询固定集合

默认情况下，在固定集合上查询将以插入顺序显示结果。但是，如果要以相反的顺序检索文档，请使用sort命令，如以下代码所示：

> db.cappedLogCollection.find().sort({$natural:-1})

关于固定集合值得注意的几个要点：

* 无法从固定集合中删除文档。
* 固定集合中没有默认索引，甚至不在\_id字段上。
* 在插入新文档时，MongoDB不需要在磁盘上实际寻找一个容纳新文档的位置。它可以随便地将新文档插入集合的尾部。这样就使得在固定集合中的插入操作非常快。
* 类似地，在读取文档的同时，MongoDB按照磁盘上存储的顺序返回文档。这样使读取操作非常快。

## MongoDB自动递增序列

MongoDB中没有类似SQL数据库中那么拿来即用的自动增量功能。默认情况下，它使用\_id字段的12字节ObjectId作为唯一标识文档的主键。但是，可能存在我们可能希望\_id字段拥有除ObjectId之外的一些自动递增值的情况。

由于MongoDB不提供默认自动增长功能，所以我们通过使用MongoDB文档中所建议的计数器集合以编程的方式来实现此功能。

### 使用计数器集合

考虑以下product文档。 我们希望\_id字段是从:1,2,3,4到n开始的自动递增整数序列。

{

"\_id":1,

"product\_name": "Huawei P9",

"category": "mobiles"

}

为此，创建一个计数器集合，它将跟踪所有序列字段的最后一个序列值。

>db.createCollection("counters")

现在，将把以下文档插入计数器集合中，以productid作为键：

{

"\_id":"productid",

"sequence\_value": 0

}

字段sequence\_value跟踪序列的最后一个值。使用以下代码将此序列文档插入计数器集合中：

>db.counters.insert({\_id:"productid",sequence\_value:0})

### 使用Javascript函数

现在将在创建一个新文档时使用函数getNextSequenceValue，并将返回的序列值分配为文档的\_id字段的值。

使用以下代码插入两个示例文档：

>db.products.insert({

"\_id":getNextSequenceValue("productid"),

"product\_name":"HuaWei P9",

"category":"mobiles"

})

>db.products.insert({

"\_id":getNextSequenceValue("productid"),

"product\_name":"OPPO X6s",

"category":"mobiles"

})

如上所示，使用getNextSequenceValue函数来设置\_id字段的值。

>db.products.find()

上述查询返回了具有自动递增的\_id字段的值，如下文档所示：

{ "\_id" : 1, "product\_name" : "HuaWei P9", "category" : "mobiles"}

{ "\_id" : 2, "product\_name" : "OPPO X6s", "category" : "mobiles" }

# 用户及安全

## MongoDB安全检查表

### 启用访问控制和强制验证

启用访问控制并指定认证机制。可以使用默认的MongoDB身份验证机制或现有的外部框架。 认证要求所有客户端和服务器在连接到系统之前提供有效的凭据。 在群集部署中，为每个MongoDB服务器启用身份验证。

### 配置基于角色的访问控制

先创建用户管理员，然后创建其他用户。 为访问系统的每个人员和应用程序创建一个唯一的MongoDB用户。

创建定义一组用户需要的确切访问权限的角色。 遵循最低权限的原则。 然后创建用户并分配他们只需要执行其操作所需的角色。用户可以是个人或客户端应用程序。

### 加密通信

配置MongoDB为所有传入和传出连接使用*TLS/SSL*。 使用*TLS/SSL*加密MongoDB部署的mongod和mongos组件之间以及所有应用程序和MongoDB之间的通信。

### 加密和保护数据

从MongoDB Enterprise 3.2开始，WiredTiger存储引擎的本机加密在Rest中可以配置为加密存储层中的数据。

如果您没有使用WiredTiger的加密功能，则应使用文件系统，设备或物理加密在每台主机上对MongoDB数据进行加密。 使用文件系统权限保护MongoDB数据。MongoDB数据包括数据文件，配置文件，审核日志和密钥文件。

### 限制网络曝光

确保MongoDB在受信任的网络环境中运行，并限制MongoDB实例监听传入连接的接口。 只允许受信任的客户端访问MongoDB实例可用的网络接口和端口。

### 审计系统活动

跟踪数据库配置和数据的访问和更改。 MongoDB Enterprise包括一个系统审核工具，可以在MongoDB实例上记录系统事件(例如用户操作，连接事件)。 这些审核记录允许进行法证分析，并允许管理员验证正确的控制。

### 使用专用用户运行MongoDB

使用专用的操作系统用户帐户运行MongoDB进程。 确保帐户具有访问数据但没有不必要权限的权限而导致的安全问题。

### 使用安全配置选项运行MongoDB

MongoDB支持执行某些服务器端操作的JavaScript代码：mapReduce，group和$where。 如果不使用这些操作，请使用命令行上的--noscripting选项禁用服务器端脚本。

在生产部署中仅使用MongoDB线路协议。不启用以下功能，所有这些都启用了Web服务器接口：net.http.enabled，net.http.JSONPEnabled和net.http.RESTInterfaceEnabled。保持这些禁用，除非向后兼容性要求。

*注：自 MongoDB3.2 版本以来已弃用：MongoDB的HTTP接口*

保持输入验证有效。 MongoDB默认通过wireObjectCheck设置启用输入验证。 这确保了mongod实例存储的所有文档都是有效的BSON。

### 请求安全技术实施指南(如适用)

“安全技术实施指南”(STIG)载有美国国防部内部署的安全准则。 MongoDB Inc.根据要求提供其STIG，用于需要的情况。 请[索取副本](http://www.mongodb.com/lp/contact/stig-requests?_ga=2.177803685.1051370967.1499006039-1870842856.1498599569)以获取更多信息。

## MongoDB认证

验证是验证客户端身份的过程。 当启用访问控制(即授权)时，MongoDB要求所有客户端进行身份验证，以确定其访问。

虽然认证和授权密切相关，但认证与授权不同。认证是验证用户的身份; 授权确定验证的用户对哪些资源和操作的访问。

### 认证方法

要验证用户，可参考MongoDB提供的[db.auth()](http://docs.mongodb.com/manual/reference/method/db.auth/#db.auth)方法。

对于mongo shell和MongoDB工具，还可以通过从命令行传递用户身份验证信息来验证用户。

### 认证机制

MongoDB支持许多身份验证机制，客户端可以使用它们来验证身份。 这些机制允许MongoDB集成到现有的身份验证系统中。

MongoDB支持多种认证机制：

* [SCRAM-SHA-1](http://docs.mongodb.com/manual/core/security-scram-sha-1/#authentication-scram-sha-1)
* [MongoDB质询和响应(MongoDB-CR)](http://docs.mongodb.com/manual/core/security-mongodb-cr/#authentication-mongodb-cr)。MongoDB 3.0版本更改：3.0中创建的新质询响应用户将使用SCRAM-SHA-1。 如果使用MongoDB 2.6的用户数据，MongoDB 3.0将继续使用MONGODB-CR。
* [x.509证书认证](http://docs.mongodb.com/manual/core/security-x.509/#security-auth-x509)。

MongoDB Enterprise除了支持上述机制外，还支持以下机制：

* [LDAP代理身份验证](http://docs.mongodb.com/manual/core/authentication-mechanisms-enterprise/#security-auth-ldap)
* [Kerberos身份验证](http://docs.mongodb.com/manual/core/authentication-mechanisms-enterprise/#security-auth-kerberos)

### 内部认证

除了验证客户端的身份之外，MongoDB还可以要求副本集和分片集群的成员对其各自的副本集或分片集群进行身份验证。 有关详细信息，请参阅[内部认证](http://docs.mongodb.com/manual/core/security-internal-authentication/#inter-process-auth)。

### 分片集群的认证

在分片集群中，客户端通常直接向mongos实例进行身份验证。 但是，某些维护操作可能需要直接对特定的分片进行身份验证。 有关认证和分片集群的更多信息，请参阅[分片集群用户](http://docs.mongodb.com/manual/core/security-users/#sharding-security)。

## MongoDB用户

要在MongoDB中验证客户端，必须先在MongoDB添加相应的用户。

### 用户管理接口

要添加用户，可使用MongoDB提供的[db.createUser()](http://docs.mongodb.com/manual/reference/method/db.createUser/#db.createUser)方法。 添加用户时，可以为用户分配角色以授予权限。

注意：在数据库中创建的第一个用户应该是具有管理其他用户的权限的用户管理员。

还可以更新现有用户，例如更改密码并授予或撤销角色。有关用户管理方法的完整列表，请参阅[用户管理](http://www.yiibai.com/mongodb/enable-authentication.html)。

### 验证数据库

添加用户时，可以在特定数据库中创建用户。该数据库是用户的认证的数据库。

用户可以跨不同数据库拥有权限; 即用户的权限不限于认证数据库。 通过分配给其他数据库中的用户角色，在一个数据库中创建的用户可以拥有对其他数据库的操作权限。 有关角色的更多信息，请参阅[基于角色的访问控制](http://www.yiibai.com/mongodb/authorization.html)。

用户名和认证数据库作为该用户的唯一标识符。 也就是说，如果两个用户具有相同的名称，但是在不同的数据库中创建，则它们是两个不同的用户。 如果您打算拥有具有多个数据库权限的单个用户，请在适用的数据库中创建具有角色的单个用户，而不是在不同数据库中多次创建用户。

### 认证用户

要验证用户，也可以：

* 在连接到 mongod 或 mongos 实例时使用命令行身份验证选项(例如：-u，-p，--authenticationDatabase)
* 先连接到 mongod 或 mongos 实例，然后针对身份验证数据库运行 authenticate 命令或[db.auth()](http://docs.mongodb.com/manual/reference/method/db.auth/#db.auth)方法。

要进行身份验证，客户端必须对用户的身份验证数据库进行身份验证。

例如，如果使用 mongo shell作为客户端，则可以使用--authenticationDatabase选项为用户指定身份验证数据库。

### 分片集群用户

要为分片集群创建用户，请连接到 mongos 实例并添加用户。 客户端然后通过 mongos 实例验证这些用户。

*Mongo 2.6* 版本中更改：MongoDB将这些分片的群集用户数据存储在配置服务器的管理数据库中。 以前，用于对数据库的主分片上驻留在分片集群上的数据库进行身份验证的凭据。

**分片本地用户**

某些维护操作(如：cleanupOrphaned，compact，rs.reconfig())需要直接连接到分片集群中的特定分片。要执行这些操作，必须直接连接到分片，并作为分片本地管理用户进行身份验证。

要创建分片本地管理用户，请直接连接到分片并创建用户。 MongoDB将分片本地用户存储在分片本身的admin数据库中。

这些分片本地用户完全独立于通过mongos添加到分片集群的用户。 分片本地用户是分片本地的，并且是无法访问的。

与分片的直接连接只能用于特定于分片的维护和配置。 一般来说，客户端应通过mongos连接到分片的集群。

## MongoDB添加用户

MongoDB采用基于角色的访问控制(RBAC)来确定用户的访问。 授予用户一个或多个角色，确定用户对MongoDB资源的访问权限和用户可以执行哪些操作。 用户应该只有最小权限集才能确保最小权限的系统。

MongoDB系统的每个应用程序和用户都应该映射到不同的用户。 这种访问隔离便于访问撤销和持续的用户维护。

### 前提条件

如果启用了部署的访问控制，则可以使用[localhost异常](http://docs.mongodb.com/manual/core/security-users/#localhost-exception)来创建系统中的第一个用户。 此第一个用户必须具有创建其他用户的权限。 对于MongoDB 3.0，使用localhost异常只能在admin数据库上创建用户。 创建第一个用户后，必须使用该用户进行身份验证以添加后续用户。 [启用Auth](http://docs.mongodb.com/manual/tutorial/enable-authentication/)提供有关在启用部署访问控制时添加用户的更多详细信息。

对于常规用户创建，必须拥有以下权限：

* 要在数据库中创建新用户，必须在该数据库资源上具有[createUser](http://docs.mongodb.com/manual/reference/privilege-actions/#createUser)操作。
* 要向用户授予角色，必须对角色的数据库执行grantRole操作。

[userAdmin](http://docs.mongodb.com/manual/reference/built-in-roles/#userAdmin)和[userAdminAnyDatabase](http://docs.mongodb.com/manual/reference/built-in-roles/#userAdminAnyDatabase)内置角色在其各自的资源上提供createUser和grantRole操作。

### 例子

要在MongoDB部署中创建用户，请连接到部署，然后使用db.createUser()方法或createUser命令添加用户。

MongoDB是一个nosql数据库服务器。 默认安装提供使用mongo命令通过命令行访问数据库而不进行身份验证。下面我们来学习如何在具有适当身份验证的Mongodb服务器中创建用户。

#### 创建管理员用户

可以使用以下命令在MongoDB服务器中创建具有管理员权限的用户。

$ mongo

> use admin

> db.createUser(

{

user:"maxsu",

pwd:"pwd123",

roles:[{role:"root",db:"admin"}]

}

)

> exit

现在尝试通过命令行使用上述用户凭据登录：

D:\Program Files\MongoDB\Server\3.4\bin>mongo -u maxsu -p --authenticationDatabase admin

MongoDB shell version v3.4.5

Enter password:

connecting to: mongodb://127.0.0.1:27017

MongoDB server version: 3.4.5

Server has startup warnings:

2017-07-05T21:16:55.901+0800 I CONTROL [initandlisten]

2017-07-05T21:16:55.902+0800 I CONTROL [initandlisten] \*\* WARNING: Access control is not enabled for the database.

2017-07-05T21:16:55.905+0800 I CONTROL [initandlisten] \*\* Read and write access to data and configuration is unrestricted.

2017-07-05T21:16:55.908+0800 I CONTROL [initandlisten]

#### 添加数据库用户

您还可以创建指定数据库的用户，该用户只能访问该数据库。也可以为此数据库上的用户指定访问级别。 例如，创建一个在mydb数据库上具有读写访问权限的用户帐户。

> use mydb

> db.createUser(

{

user: "user01",

pwd: "pwd123",

roles: ["readWrite"]

}

)

> exit

验证身份验证使用以下命令。 返回结果为1，表示认证成功。

> db.auth('user01','pwd123')

1

要列出数据库的所有用户，请使用以下命令。

> db.getUsers()

上面语句，返回结果如下所示：

> db.getUsers()

[

{

"\_id" : "mydb.user01",

"user" : "user01",

"db" : "mydb",

"roles" : [

{

"role" : "readWrite",

"db" : "mydb"

}

]

},

{

"\_id" : "mydb.user02",

"user" : "user02",

"db" : "mydb",

"roles" : [

{

"role" : "readWrite",

"db" : "mydb"

}

]

}

]

#### 删除数据库用户

也可以使用以下命令从数据库中删除用户。

> use mydb

> db.dropUser('user02')

#### 创建带角色的用户

以下操作在test数据库中创建用户：mynewuser，并向用户提供readWrite和dbAdmin角色。

use test

db.createUser(

{

user: "mynewuser",

pwd: "myuser123",

roles: [ "readWrite", "dbAdmin" ]

}

);

#### 创建没有角色的用户

以下操作在test数据库中创建一个名为mynewuser1的用户，但尚未分配角色：

use test

db.createUser(

{

user: "mynewuser1",

pwd: "myuser1123",

roles: [ ]

}

);

#### 创建具有角色的管理用户

以下操作在管理数据库中创建一个名为 myadmin1 的用户，并给予用户对config数据库的 readWrite 访问权限，这样可以让用户更改分片分区的某些设置，例如平衡器设置。

use admin

db.createUser(

{

user: "myadmin1",

pwd: "myadmin123",

roles:

[

{ role: "readWrite", db: "config" },

"clusterAdmin"

]

}

);

## MongoDB启用身份验证

对MongoDB部署启用访问控制会强制执行用户身份验证，要求在登录MongoDB系统用户识别自己。 当访问启用了访问控制的MongoDB部署时，用户只能执行由其角色确定的操作。

对于认证，MongoDB支持各种认证机制。

以下教程启用独立 mongod 实例的访问控制，并使用默认身份验证机制。

### 用户管理员

启用访问控制后，请确保在admin数据库中拥有userAdmin或userAdminAnyDatabase角色的用户。该用户可以管理用户和角色，例如：创建用户，授予或撤销用户角色，以及创建或修改定义角色。

可以在启用访问控制之前或之后创建用户。如果在创建任何用户之前启用访问控制，MongoDB将提供本地主机异常，允许在管理数据库中创建用户管理员。创建后，必须作为用户管理员进行身份验证，以根据需要创建其他用户。

### 过程

以下过程首先将用户管理员添加到运行无访问控制的 MongoDB 实例，然后启用访问控制。

#### 第一步：启动MongoDB无需访问控制

例如，以下启动不具有访问控制的独立 mongod 实例。

mongod --port 27017 --dbpath /data/db1

#### 第二步：连接到实例

例如，使用mongo shell连接到实例。

mongo --port 27017

根据需要指定其他命令行选项以将mongo shell连接到部署，例如--host。

#### 第三步：创建用户管理员

在管理数据库中，添加具有userAdminAnyDatabase角色的用户。 例如，以下在admin数据库中创建用户myUserAdmin：

注意：创建用户的数据库(在此示例中为admin)是用户的身份验证数据库。用户将对该数据库进行身份验证，但用户可以在其他数据库中担任角色; 即用户的认证数据库不限制用户的权限。

use admin

db.createUser(

{

user: "myUserAdmin",

pwd: "abc123",

roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]

}

)

执行上面命令后，断开mongo shell。

#### 第四步：重新启动具有访问控制的MongoDB实例

使用--auth命令行选项重新启动mongod实例，或者如果使用配置文件，则执行security.authorization设置。

mongod --auth --port 27017 --dbpath /data/db1

连接到此实例的客户端现在必须以MongoDB用户身份进行身份验证。客户只能执行由其分配的角色确定的操作。

#### 第五步：以用户管理员身份进行连接和验证

使用mongo shell，可以：

* 传递用户凭据。
* 连接第一个withouth身份验证，然后发出db.auth()方法进行身份验证。

**在连接期间进行身份验证**

使用-u <username>，-p <password>和--authenticationDatabase <database>命令行选项启动一个mongo shell：

$ mongo --port 27017 -u "myUserAdmin" -p "abc123" --authenticationDatabase "admin"

**连接后验证**

将 *mongo shell* 连接到 mongodb，也就是先连接，后验证用户身份：

mongo --port 27017

切换到身份验证数据库(在这种情况下为admin)，并使用db.auth(<username>，<pwd>)方法进行身份验证：

use admin

db.auth("myUserAdmin", "abc123" )

#### 第六步：根据需要创建其他用户

当管理员用户进行身份验证通过之后，可使用db.createUser()创建其他用户。可以为用户分配任何内置角色或用户定义的角色。

myUserAdmin用户只具有管理用户和角色的权限。如果使用myUserAdmin尝试执行任何其他操作，例如从test数据库中的foo集合读取数据，MongoDB将返回错误。

以下操作将用户 myTester 添加到在test数据库中并给予test数据库的readWrite角色以及在reporting数据库中读取角色。

注意：创建用户的数据库(在本示例中为test)是该用户的身份验证数据库。虽然用户将对该数据库进行身份验证，但用户可以在其他数据库中担任角色; 即用户的认证数据库不会限制用户的权限。

use test

db.createUser(

{

user: "myTester",

pwd: "xyz123",

roles: [ { role: "readWrite", db: "test" },

{ role: "read", db: "reporting" } ]

}

)

#### 第七步：连接并验证为myTester

**在连接期间进行身份验证**

使用-u <username>，-p <password>和--authenticationDatabase <database>命令行选项启动一个*mongo shell*：

$ mongo --port 27017 -u "myTester" -p "xyz123" --authenticationDatabase "test"

**连接后验证**

将*mongo shell*连接到 mongodb：

$ mongo --port 27017

切换到身份验证数据库(在这里为test)，并使用db.auth(<username>，<pwd>)方法进行身份验证：

> use test

> db.auth("myTester", "xyz123" )

使用用户 myTester 插入一个集合

使用用户 myTester，此用户有权在test数据库中执行读写操作(以及在reporting数据库中执行读操作)。 例如，在test数据库中执行以下插入操作：

> db.foo.insert( { x: 1, y: 1 } )

最后，使用用户 myTester，在reporting数据库中执行插入操作看看返回结果：

> use reporting

db.auth("myTester", "xyz123" )

db.product.insert( { x: 1, y: 1 } )

## MongoDB管理用户和角色

### 创建用户定义的角色

角色授权用户访问MongoDB资源。 MongoDB提供了许多内置的角色，管理员可以使用它们来控制对MongoDB系统的访问。 但是，如果这些角色无法描述所需的权限集，则可以在特定数据库中创建新角色。

除了在管理数据库中创建的角色外，角色只能包含适用于其数据库的权限，并且只能继承其数据库中的其他角色。

在管理数据库中创建的角色可以包括适用于管理数据库，其他数据库或群集资源的权限，并且可以从其他数据库中的角色以及管理数据库继承。

要创建新角色，可使用db.createRole()方法，指定permissions数组中的权限和roles数组中的继承角色。

MongoDB使用数据库名称和角色名称的组合来唯一定义角色。 每个角色的范围限定在创建角色的数据库中，但MongoDB将所有角色信息存储在admin数据库的admin.system.roles集合中。

**先决条件**

要在数据库中创建角色，您必须具有：

* 对该数据库资源的createRole操作。
* 对该数据库的grantRole操作指定新角色的权限以及指定要继承的角色。

内置角色userAdmin和userAdminAnyDatabase在其各自的资源上提供createRole和grantRole操作。

### 创建角色来管理当前操作

以下示例创建一个名为manageOpRole的角色，该角色仅提供运行db.currentOp()和db.killOp()的权限。

**第一步：使用相应的权限连接到MongoDB**

使用“**先决条件**”部分指定的权限连接到mongod或mongos。

以下过程使用在“启用认证”中创建的用户：myUserAdmin。

$ mongo --port 27017 -u "myUserAdmin" -p "abc123" --authenticationDatabase "admin"

myUserAdmin具有在管理员以及其他数据库中创建角色的权限。

**第二步：创建一个新角色来管理当前操作**

manageOpRole具有对多个数据库以及群集资源的权限。 因此，您必须在管理数据库中创建该角色。

use admin

db.createRole(

{

role: "manageOpRole",

privileges: [

{ resource: { cluster: true }, actions: [ "killop", "inprog" ] },

{ resource: { db: "", collection: "" }, actions: [ "killCursors" ] }

],

roles: []

}

)

新角色授予杀死/终止任何操作的权限。

警告: 终止运行操作需要非常小心。只能使用db.killOp()方法或killOp命令终止客户端发起的操作，并且不会终止内部数据库操作。

### 创建角色用来运行 mongostat

以下示例创建一个名为mongostatRole的角色，该角色仅提供运行mongostat的权限。

**第一步：使用相应的权限连接到MongoDB**

使用“**先决条件**”部分指定的权限连接到mongod或mongos。

以下过程使用在[启用认证](http://www.yiibai.com/mongodb/authentication.html)中创建的用户：myUserAdmin。

$ mongo --port 27017 -u "myUserAdmin" -p "abc123" --authenticationDatabase "admin"

myUserAdmin具有在管理员以及其他数据库中创建角色的权限。

**第二步：创建一个新角色来管理当前的操作**

mongostatRole具有作用于群集资源的权限。 因此，您必须在管理数据库中创建该角色。

use admin

db.createRole(

{

role: "mongostatRole",

privileges: [

{ resource: { cluster: true }, actions: [ "serverStatus" ] }

],

roles: []

}

)

### 修改现有用户的访问权限

**先决条件**

* 必须对数据库具有grantRole操作才能在该数据库上授予角色。
* 必须在数据库上具有revokeRole操作以撤销该数据库上的角色。
* 要查看角色的信息，必须明确授予该角色，或必须对该角色的数据库具有viewRole操作。

**执行步骤**

**第一步：使用相应的权限连接到MongoDB**

以具有先决条件部分中指定的权限的用户身份连接到mongod或mongos。

以下过程使用在[启用认证](http://www.yiibai.com/mongodb/authentication.html)中创建的用户：myUserAdmin。

$ mongo --port 27017 -u "myUserAdmin" -p "abc123" --authenticationDatabase "admin"

**第二步：识别用户的角色和权限**

要显示要修改的用户的角色和权限，请使用db.getUser()和db.getRole()方法。

例如，要查看在示例中创建的reportsUser的角色，执行以下命令：

use reporting

db.getUser("reportsUser")

要显示在“accounts”数据库上由readWrite角色授予用户的权限，请执行以下操作：

use accounts

db.getRole( "readWrite", { showPrivileges: true } )

**第三步：确定授予或撤销的权限**

如果用户需要额外的权限，则向用户授予具有所需权限集的角色或角色。 如果此类角色不存在，请使用适当的权限集创建新角色。

撤销由现有角色提供的特权子集：撤销原始角色并授予仅包含所需权限的角色。如果角色不存在，您需要创建新角色。

**第四步：修改用户的访问权限**

（1）撤销角色

使用db.revokeRolesFromUser()方法撤销角色。以下示例操作从account数据库上删除用户reportsUser的readWrite角色：

use reporting

db.revokeRolesFromUser(

"reportsUser",

[

{ role: "readWrite", db: "accounts" }

]

)

（2）授予角色

使用db.grantRolesToUser()方法授予角色。 例如，以下操作授予reportsUser用户account数据库上的读取角色：

use reporting

db.grantRolesToUser(

"reportsUser",

[

{ role: "read", db: "accounts" }

]

)

对于分片集群，用户的更改将在命令运行的mongos上即时生效。但是，对于群集中的其他mongos实例，用户缓存可能会等待10分钟才能刷新。请参阅[userCacheInvalidationIntervalSecs](http://docs.mongodb.com/manual/reference/parameters/#param.userCacheInvalidationIntervalSecs)。

### 修改现有用户的密码

**先决条件**

要修改数据库上另一个用户的密码，您必须对该数据库具有changeAnyPassword操作。

**操作步骤：**

**第一步：使用相应的权限连接到MongoDB**

以具有先决条件部分中指定的权限的用户身份连接到mongod或mongos。

以下过程使用在[启用认证](http://www.yiibai.com/mongodb/authentication.html)中创建的用户：myUserAdmin。

$ mongo --port 27017 -u "myUserAdmin" -p "abc123" --authenticationDatabase "admin"

**第二步：更改密码**

将用户的用户名和新密码传递给db.changeUserPassword()方法。

以下操作将reporting用户的密码更改为：SOh3TbYhxuLiW8ypJPxmt1oOfL。

db.changeUserPassword("reporting", "SOh3TbYhxuLiW8ypJPxmt1oOfL")

### 查看用户的角色

**先决条件**

要查看其他用户的信息，您必须对其他用户的数据库具有viewUser操作。

用户可以查看自己的信息。

**第一步：使用相应的权限连接到MongoDB**

以具有先决条件部分中指定的权限的用户身份连接到mongod或mongos。

以下过程使用在[启用认证](http://www.yiibai.com/mongodb/authentication.html)中创建的用户：myUserAdmin。

$ mongo --port 27017 -u "myUserAdmin" -p "abc123" --authenticationDatabase "admin"

**第二步：查看用户的角色**

使用usersInfo命令或db.getUser()方法显示用户信息。

例如，要查看在示例中创建的reportsUser的角色，请发出：

use reporting

db.getUser("reportsUser")

在返回的文档中，roles字段显示reportsUser的所有角色：

...

"roles" : [

{ "role" : "readWrite", "db" : "accounts" },

{ "role" : "read", "db" : "reporting" },

{ "role" : "read", "db" : "products" },

{ "role" : "read", "db" : "sales" }

]

### 查看角色的权限

**先决条件**

要查看角色的信息，您必须明确授予该角色，或必须对该角色的数据库具有viewRole操作。

**第一步：使用相应的权限连接到MongoDB**

以具有先决条件部分中指定的权限的用户身份连接到mongod或mongos。

以下过程使用在[启用认证](http://www.yiibai.com/mongodb/authentication.html)中创建的用户：myUserAdmin。

$ mongo --port 27017 -u "myUserAdmin" -p "abc123" --authenticationDatabase "admin"

**第二步：查看角色授予的权限**

对于给定的角色，请使用db.getRole()方法或rolesInfo命令与showPrivileges选项一起执行：

例如，要查看在product数据库上由读取角色授予的权限，请使用以下操作，问题如下：

use products

db.getRole( "read", { showPrivileges: true } )

在返回的文档中，有两个数组：privileges和inheritedPrivileges。权限列出了角色指定的权限，并排除了从其他角色继承的权限。inheritedPrivileges列出了由此角色授予的所有权限，这两个角色都是直接指定的并被继承。如果该角色不能从其他角色继承，则两个字段是相同的。

...

"privileges" : [

{

"resource": { "db" : "products", "collection" : "" },

"actions": [ "collStats","dbHash","dbStats","find","killCursors","planCacheRead" ]

},

{

"resource" : { "db" : "products", "collection" : "system.js" },

"actions": [ "collStats","dbHash","dbStats","find","killCursors","planCacheRead" ]

}

],

"inheritedPrivileges" : [

{

"resource": { "db" : "products", "collection" : "" },

"actions": [ "collStats","dbHash","dbStats","find","killCursors","planCacheRead" ]

},

{

"resource" : { "db" : "products", "collection" : "system.js" },

"actions": [ "collStats","dbHash","dbStats","find","killCursors","planCacheRead" ]

}

]

## MongoDB更改用户密码和自定义数据

具有适当权限的用户可以更改自己的密码和自定义数据。 自定义数据存储可选的用户信息。

### 注意事项

在此过程要生成一个强大的密码，您可以使用openssl实用程序的rand命令。 例如，使用以下选项发出openssl rand，以创建48个伪随机字节的base64编码字符串：

openssl rand -base64 48

### 先决条件

要修改自己的密码和自定义数据，您必须具有在用户数据库上分别授予changeOwnPassword和changeOwnCustomData操作的权限。

**第一步：使用相应的权限连接到MongoDB**

使用“**先决条件**”部分指定的权限连接到mongod或mongos。

以下过程使用在“启用认证”中创建的用户：myUserAdmin。

$ mongo --port 27017 -u "myUserAdmin" -p "abc123" --authenticationDatabase "admin"

**第二步：使用适当的权限创建角色**

在管理数据库中，使用changeOwnPassword和changeOwnCustomData创建一个新角色。

use admin

db.createRole(

{ role: "changeOwnPasswordCustomDataRole",

privileges: [

{

resource: { db: "", collection: ""},

actions: [ "changeOwnPassword", "changeOwnCustomData" ]

}

],

roles: []

}

)

**第三步：添加具有此角色的用户**

在test数据库中，使用创建的“changeOwnPasswordCustomDataRole”角色创建一个新用户。 例如，以下操作将创建具有内置角色readWrite和用户创建的“changeOwnPasswordCustomDataRole”的用户。

use test

db.createUser(

{

user:"user123",

pwd:"12345678",

roles:[ "readWrite", { role:"changeOwnPasswordCustomDataRole", db:"admin" } ]

}

)

要向现有用户授予新角色，请使用db.grantRolesToUser()。

### 执行过程

**第一步：使用相应的权限连接到MongoDB**

使用“**先决条件**”部分指定的权限连接到mongod或mongos。

以下过程使用在“启用认证”中创建的用户：myUserAdmin。

$ mongo --port 27017 -u "myUserAdmin" -p "abc123" --authenticationDatabase "admin"

要检查您是否具有**先决条件**部分中指定的权限以及查看用户信息，请使用带有--showPrivileges选项的usersInfo命令。

**第二步：更改您的密码和自定义数据**

使用db.updateUser()方法来更新密码和自定义数据。

例如，以下操作将用户的密码更改为：KNlZmiaNUp0B，并将自定义数据更改为{title：“Senior Manager”}：

use test

db.updateUser(

"user123",

{

pwd: "KNlZmiaNUp0B",

customData: { title: "Senior Manager" }

}

)