

1. 数据库概述

本文主要包含以下知识点：

- 数据存储概述
- SQL 介绍

数据存储概述

在计算机出现以前，人们都是以纸质文件的形式来保存数据的。但是这种方式保存数据，有很多缺点。

例如：

1. 纸质文件容易丢失和损坏

因为使用的是纸质文件来保存数据，所以很容易发生文件的丢失和损坏。甚至可能会出现因为时间过长，纸质文件产生发潮、字迹模糊等状况。

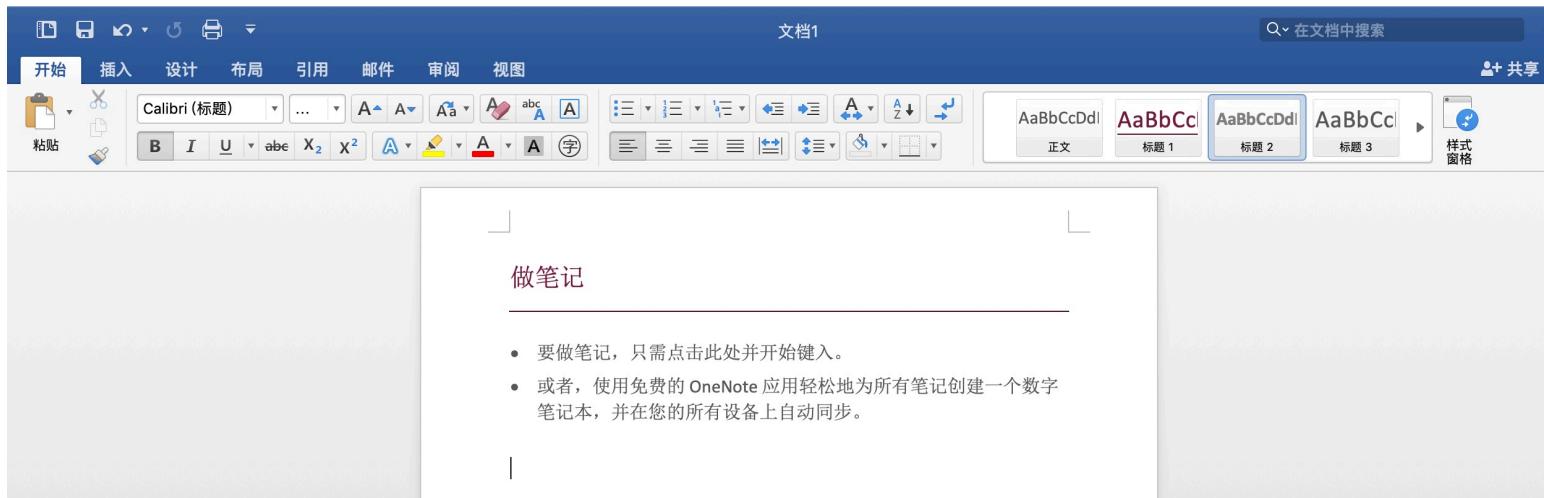
2. 占用空间大

如果要存储海量的信息，那么就需要大量的纸质文件。而这些纸质文件又需要大量的文件柜来存放，所以占用空间大。需要有足够大的场地来放置这些文件柜。

3. 查找数据困难

面对大量使用纸质文件来保存的数据，如果我要查找很多年前的一个人的信息的话，是非常困难的。

后年，计算机出现了，人们选择使用计算机来保存数据，最常见的就是使用微软公司的办公软件 Word 或者 Excel 来存储数据，如下图：



(图为使用 Word 软件来保存数据)

管理我的财务1										
开始 插入 页面布局 公式 数据 审阅 视图										
粘贴	剪切	插入	合并后居中	常规	条件格式	插入	Σ	A	Z	共享
F7	X	✓	f7	C	D	E	F	G	H	I
17										
18	每月支出									
19	项目	金额								
20	租金/抵押贷款	¥8,000.00								
21	电费	¥1,200.00								
22	加油费	¥500.00								
23	移动电话	¥450.00								
24	日用杂货	¥5,000.00								
25	车贷	¥2,730.00								
26	信用卡	¥1,200.00								
27	车险	¥500.00								
28	杂项	¥1,000.00								
29										

(图为使用 Excel 软件来保存数据)

但是，使用这种方式来存储数据仍然存在以下缺点：

1. 查询数据不便

在数据的查询方面，虽然这种方式比起使用纸质文件要便利很多，但是仍然会存在各种各样的不方便。并且，如果存储的数据量小倒也还好，但是一旦数据量巨大，文件大小上了 10 个 G 的话，别说查询了，连打开文件都特别困难。

2. 并发量差

如果是多人一起访问数据的情况下，文件非常容易崩溃甚至损坏。

最终，为了解决上述的问题，数据库出现了。

那么，什么是数据库？

数据库，英语为 Database，简称 DB。是按照数据结构组织，存储和管理数据的仓库。其本身可看作电子化的文件柜，用户可以对文件中的数据进行增加、删除、修改和查找等操作。

注：这里所说的数据不仅仅包括普通意义的数字，还包括文字、图象、声音等。也就是说，凡是在计算机中用来描述事物的记录都可以称作数据。

数据库具有如下的特点：

1. 数据结构化

所谓结构化，就是数据拒绝零散，按照结构排列。

2. 实现数据共享

因为数据是面向整体的，所以数据可以被多个用户、多个应用程序共享使用。这样可以大幅度的减少数据的冗余，节约存储空间，避免数据之间的不相容性与不一致性。

3. 数据独立性高

数据的独立性包括逻辑独立性和物理独立性。

逻辑独立性：是指数据库中数据的逻辑结构和应用程序相互独立。

物理独立性：是指物理结构的变化不影响数据的逻辑结构。

4. 数据统一管理和控制

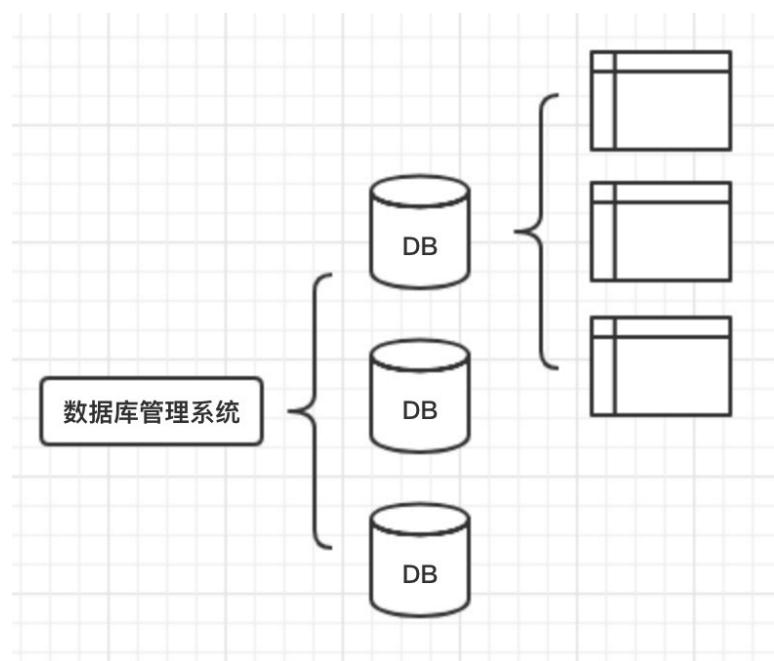
数据的统一控制包含安全控制、完整控制和并发控制。

简单来说，就是防止数据丢失，确保数据的正确有效，并且在同一时间，允许用户对数据进行多路存取。

数据库的分类

数据库的分类是根据数据在数据库中的存储结构来进行划分的。一般来讲，可以分为**关系型数据库**以及**非关系型数据库**。

在关系型数据库，数据是存储在一张一张的表格里面的，而每一张表格彼此之间又产生了相应的关系，如下图：



常见的关系型数据库有：*Oracle*、*SQL Server*、*MySQL* 等。

什么是数据库系统？

大多初学者认为，数据库就是数据库系统。但实际上，数据库系统的范围要比数据库大很多。

数据库系统是由**硬件**和**软件**组成的。其中硬件主要用于存储数据库中的数据，包括计算机、存储设备等，软件主要包括操作系统以及应用程序等。

这里我们主要来看一下软件的部分。一般来讲，软件系统可以分为 3 个部分，分别是：

1. 数据库

英语全称 Database，简称 DB，翻译成中文就是数据库。数据库提供了一个存储空间用来存储各种数据。可以将数据库视为一个存储数据的容器。

2. 数据库管理系统

英语全称 Database Management System，简称 DBMS，翻译成中文就是数据库管理系统。所谓数据库管理系统，是指专门用于创建和管理数据库的一套软件，介于应用程序和操作系统之间。

下面介绍几种常见的数据库管理系统：

Oracle 数据库管理系统

Oracle 数据库管理系统是由甲骨文公司开发的，在数据库领域一直处于领先地位。目前，Oracle 数据库覆盖了大、中、小型计算机等几十种计算机型，成为世界上使用最广泛的关系型数据管理系统。



SQL Server 数据库管理系统

SQL Server 数据库管理系统是由微软公司开发的一种关系型数据库，它已经广泛用于电子商务，银行，保险，电力等行业。

该数据库管理系统界面友好，易于操作，深受广大用户喜爱，但是只能在 Windows 平台上运行。



DB 2 数据库管理系统

DB 2 数据库管理系统是由IBM公司研制的一种关系型数据库，该数据库管理系统提供了高层次的数据利用性，完整性，完全性和可恢复性。

该数据库管理系统适合海量数据的存储，但相对于其他数据库管理系统而言，DB 2 的操作比较复杂。



PostgreSQL 数据库管理系统

PostgreSQL 是以加州大学伯克利分校计算机系开发的 POSTGRES，现在已经更名为 PostgreSQL，版本 4.2 为基础的对象关系型数据库管理系统（ORDBMS）。

这个数据库管理系统是以教学为目的而被开发出来的，所以在性能和标准的取舍上，该数据库以追求功能实现完美为首要目标。



MySQL 数据库管理系统

由瑞典 MySQL AB 公司开发，目前属于 Oracle 旗下的产品。MySQL 是开源数据的，任何人都可以获得该

数据库的源代码并修正 MySQL 的缺陷。

MySQL 具有跨平台的特性，它不仅可以在 Windows 平台使用，还可以在 UNIX, Linux 上使用。相对其他数据库而言，MySQL 的使用更加方便，快捷，而且 MySQL 是免费的，运营成本低。



MongoDB 数据库管理系统

由 10ten 公司开发的一个介于关系数据库和非关系数据库之间的产品，被誉为是非关系数据库中功能最丰富，最像关系数据库的管理系统。

并且 MongoDB 是一个开源数据库，具有高性能、易部署、易使用、存储数据非常方便等特点。对于大数据量，高并发等互联网应用尤其能够显示优势。



3. 数据库应用程序

虽然已经有了数据库管理系统，但是无法满足用户对数据库的管理，纯命令式的数据管理方式也给用户带来了诸多的不便。

此时就需要数据库应用程序来与数据库管理系统进行通信，访问和管理数据。一般来讲，数据库应用程序都提供了图形化的界面，这样使得用户在管理数据上面更加的便捷。

常见的数据库应用程序有：SQLyog、toad for mysql 等。

SQL 介绍

很长一段时间，世界范围内都是流行使用关系型数据库。并且使用 SQL 作为关系型数据库的标准查询语言。

SQL，英语全称 Structured Query Language，翻译成中文就是结构化查询语言。这是一门专门用于管理数据库中的数据，例如存储数据、查询数据、更新数据等操作的程序设计语言。

SQL 于 1975 – 1979 年之间被 IBM 开发出来，在 20 世纪 80 年代，SQL 被美国国家标准学会（ANSI）和国家标准化组织（ISO）定义为关系型数据库语言的标准。

SQL 从结构上进行划分，大致又可以分为 4 大类，分别为：

1. 数据定义语言

数据库定义语言主要用于定义数据库、表等操作。其中包括 `create` 语句、`alter` 语句、`drop` 语句等都属于数据定义语言。

2. 数据操作语言

数据操作语言主要用于对数据库进行添加、修改和删除操作。其中包括 `insert` 语句、`update` 语句、`delete` 语句等都属于数据操作语言。

3. 数据查询语言

数据库查询语言主要用于查询数据，主要就是 `select` 语句。使用该语句可以查询数据库中一条或者多条数据。

4. 数据控制语言

数据库控制语言主要用于控制用户的访问权限，其中包括 `commit` 语句、`rollback` 语句等都属于数据控制语言。

下图展示了一张存储在 MySQL 数据管理系统中的数据表，如下：

MySQL> select * from t_stu;								
id stuName stuAge stuGender stuPhone stuScore stuScore2 classNo birthday								
1	谢杰	18	男	12345	99	74	1	1990-03-23 00:00:00
4	大妹子	22	女	2147483647	65	70	2	1991-05-14 00:00:00
5	奥特曼	5	男	12345678	60	80	3	2018-03-16 00:00:00
6	张学友	20	男	2147483647	89	90	3	2017-03-06 00:00:00
7	刘德华	23	男	12345	80	80	1	1960-07-12 00:00:00
8	林志玲	25	女	54321	99	80	4	1972-04-12 00:00:00
9	雅静	20	女	NULL	90	60	2	1988-07-14 00:00:00
10	希之	1	男	12345	95	96	2	2017-03-06 00:00:00
12	佐佐木希	35	女	54321	99	100	4	1988-03-23 00:00:00
13	小东	20	男	1234	99	97	1	2013-05-05 00:00:00
14	小西	20	女	1234	99	97	2	2012-04-06 00:00:00
15	小南	20	女	1234	99	97	3	2011-03-07 00:00:00
16	小北	20	男	1234	99	97	4	2010-01-08 00:00:00

在这张数据表中我们存储了 16 个学生信息。之后如果要使用到某一个学生的信息，直接从这张数据表中取出即可。

在上图中的 `select * from t_stu` 就是一条 SQL 语句。

2. NoSQL 简介

本文主要包含以下知识点：

- NoSQL 概述
- NoSQL 数据库类型
- NoSQL 现状以及挑战

NoSQL 概述

随着互联网 Web 2.0 网站的兴起，传统的 SQL 数据库（关系型数据库）在应付 Web 2.0 网站，特别是超大规模和高并发的 SNS 类型的 Web 2.0 纯动态网站时，已经显得力不从心，暴露了很多难以克服的问题。这些问题，大多都是设计上的一些弊端。例如：

1. 不够灵活、扩展困难

在 SQL 数据库中，都是通过表来对数据进行存储。一张信息表一般拥有固定的字段，这样使得 Web 应用在进行扩充的时候显得不够灵活，横向扩展比较困难。

2. 过于笨重，性能低下

与一些特性不足的 NoSQL 数据库（非关系型数据库）相比，SQL 数据库的很多特性反而显得没有用武之地。比如，在很多场景下，及时存取并不是必要的，也没有特别多的事务需求，而这些额外的特性消耗着 SQL 数据库的性能。

NoSQL，英语全称 Not Only SQL，翻译成中文就是“不仅仅是 SQL”。



但实际上，绝大多数 NoSQL 数据库都放弃了对 SQL 语言的支持。并且与 SQL 数据库相比，NoSQL 数据库大多放弃了一些特性。例如，放弃了实时一致性、对事务的完整支持以及多表查询等。

这听起来 NoSQL 数据库的缺点很多，但收益也很明显。NoSQL 数据库简单便捷、方便扩展，并且拥有更好的性能。关于 NoSQL 的具体优缺点总结如下：

NoSQL 的优缺点

优点:

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

缺点:

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

NoSQL 数据库类型

常见的 NoSQL 类型数据库有 4 种，下面将分别进行介绍。

键值（Key-Value）存储数据库

这一类数据库主要会使用到一个哈希表，这个表中有一个特定的键和一个指针指向特定的数据。Key-Value 模型相对于 IT 系统的优势在于简单，易部署。但是如果 DBA（数据库操作员）只对部分值进行查询或更新时，Key-Value 就显得效率低下了。

该类型数据库有 Tokyo Cabinet/Tyrant、Redis、Voldemort 和 OracleBDB。



(图为 Redis 数据库 Logo)

列存储数据库

这部分数据库通常是用来应对分布式存储的海量数据。键仍然存在，但是它们的特点是指向了多个列，这些列是由列家族来安排的。

该类型数据库有 Cassandra、HBase 和 Riak。



(图为 Cassandra 数据库 Logo)

文档型数据库

文档型数据库的灵感来自于 LotusNotes 办公软件，而且它同第一种键值存储相类似。该类型的数据模型是版本化的文档，半结构化的文档以特定的格式存储，比如 JSON。文档型数据库可以看作是键值数据库的升级版，允许之间嵌套键值。而且文档型数据库比键值数据库的查询效率更高。

该类型数据库有 CouchDB 和 MongoDb，国内也有文档型数据库 SequoiaDB 已经开源。



(图为 CouchDB 数据库 Logo)

图形 (Graph) 数据库

图形结构的数据库同其他行列及刚性结构的 SQL 数据库不同，它使用灵活的图形模型，并且能够扩展到多个服务器上。NoSQL 数据库没有标准的查询语言 (SQL)，因此进行数据库查询需要制定数据模型。许多 NoSQL 数据库都有 REST 式的数据接口或者查询 API。

该类型数据库有 Neo4J、Infogrid 和 InfiniteGraph。



(图为 InfiniteGraph 数据库 Logo)

适用场景

虽然 NoSQL 数据库拥有 4 种不同的类型，但是总体来讲，NoSQL 数据库在以下的这几种情况下比较适用：

- 数据模型比较简单
- 需要灵活性更强的 IT 系统
- 对数据库性能要求较高
- 不需要高度的数据一致性
- 对于给定 key，比较容易映射复杂值的环境。

NoSQL 现状以及挑战

NoSQL 发展现状

计算机体系结构在数据存储方面要求具备庞大的水平扩展性，而 NoSQL 致力于改变这一现状。的确，NoSQL 对于大型企业来说还不是主流，但是，一两年之后很可能就会变个样子。目前已经有 Google 的 BigTable 和 Amazon 的 Dynamo 等项目使用的就是 NoSQL 型数据库。

水平扩展性 (*horizontal scalability*) 是指能够连接多个软硬件的特性，这样可以将多个服务器从逻辑上看成一个实体。

目前市面上的 NoSQL 数据库种类繁多，单从 NoSQL 项目的名字上可能看不出有什么相同之处，但是，它们通常在某些方面具有相似性，特别是在可以处理超大量数据这一方面，可以看作是一场革命。

不过，这场革命仍然需要等待。

在 NoSQL 运动的最新一次聚会中，来自世界各地的 150 人挤满在 CBS Interactive 的一间会议室里，分享着他们如何推翻缓慢而昂贵的关系数据库的暴政的经验，以及怎样使用更有效和更便宜的方法来管理数据。

"关系型数据库给你强加了太多东西。它们要你强行修改对象数据，以满足 RDBMS (relational database management system, 关系型数据库管理系统) 的需要。"在 NoSQL 拥护者们看来，基于 NoSQL 的替代方案只是给你所需要的。

NoSQL 面临的挑战

尽管大多数 NoSQL 数据存储系统都已被部署于实际应用中，但归纳其研究现状，还有许多挑战性问题。

1. 已有 Key-Value 数据库产品大多是面向特定应用自治构建的，缺乏通用性。
2. 已有产品支持的功能有限（不支持事务特性），导致其应用具有一定的局限性。
3. 目前已存在一些研究成果和改进的 NoSQL 数据存储系统，但它们都是针对不同应用需求而提出的相应解决方案，如支持组内事务特性、弹性事务等，很少从全局考虑系统的通用性，也没有形成系列化的研究成果。
4. 缺乏类似关系数据库所具有的强有力的理论（如 armstrong 公理系统）、技术（如成熟的基于启发式的优化策略、两段封锁协议等）、以及标准规范（如 SQL 语言）的支持。
5. 目前，HBase 数据库是安全特性最完善的 NoSQL 数据库产品之一，而其他的 NoSQL 数据库多数没有提供内建的安全机制，但随着 NoSQL 的发展，越来越多的人开始意识到安全的重要，部分 NoSQL 产品逐渐开始提供一些安全方面的支持。

3. MongoDB 简介

本文主要包含以下知识点：

- MongoDB 概述
- MongoDB 安装
- MongoDB 配置和启动

MongoDB 概述

在前面我们曾经提到过，目前的 NoSQL 大致分为 4 大阵营，分别是：

- 键值存储类型
- 列存储类型
- 文档存储类型
- 图形存储类型

而这里我们要为大家介绍的，就是文档存储类型的 NoSQL 中的佼佼者，MongoDB。

MongoDB 是一个开源的 NoSQL 数据库，在国内被称为芒果数据库。Linux、Apache、MySQL 和 PHP 组成了非常有名的 LAMP 架构。现在，有人提议将 LAMP 中的代表 M 的 MySQL 替换为 MongoDB。

NoSQL 数据库有很多，为什么要选择 MongoDB 呢？

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

MongoDB 使用集合（collection）和文档（document）来描述和存储数据，集合（collection）就相当于表，文档（document）相当于行，字段相当于列，不像 MySQL 之类的关系型数据库，表结构是固定的，比如某一行由若干列组成，行行都一样，而 MongoDB 不同，一个集合里的多个文档可以有不同的结构，更灵活一些。

MongoDB 有自己很鲜明的特色，总结起来有以下 4 条：

1. 没有表结构的限制

传统 SQL 数据库中，对每张表都需要定义表结构。如果有新的存储需求，往往需要添加新的字段，更改表结构。在一些场景下，会显得很不方便。

而对于 MongoDB，这不再是问题。因为它没有表结构这个概念，在使用一张表之前，不需要对这张表进行任何初始化操作。MongoDB 的这种特性对快捷开发和多变的业务需求是很合适的。

2. 完全的索引支持

有些 NoSQL 数据库，比如 Redis，它是内存数据库，速度很快。但是，做为键值数据库，只支持一种按键查询的方式。灵活性、使用范围和易用性都受到影响。再比如 HBase，写入速度很快。但是，同样查询受限，它只支持单索引，二级索引需要自己实现。

而 MongoDB 支持单键索引、多键索引、全文索引和地理位置索引。所以 MongoDB 是功能非常完善的 NoSQL 数据库，也被称为最接近关系数据库的非关系数据库。

3. 良好的数据安全性和方便的规模扩展

MongoDB 使用复制集做多副本存储，以保证数据的安全性。

同时，MongoDB 内置的分片技术可以很方便地进行数据规模的扩展。分片技术是很新颖的一个特性，它包含了自动数据接口，动态扩容和缩容等一系列在其他数据库中需要大量人工操作的工作，同时提供了对数据库的统一访问入口，不需要在应用层再进行分发，显著减少了人工成本。

4. 完善的文档支持和驱动支持

MongoDB 安装

Mac 系统安装 MongoDB

首先访问 MongoDB 的官方网站：<https://www.mongodb.com/download-center/community>

在下图的地方可以下载 MongoDB 的压缩包

Select the server you would like to run:

MongoDB Community Server
FEATURE RICH. DEVELOPER READY.

MongoDB Enterprise Server
ADVANCED FEATURES. PERFORMANCE GRADE.

Version: 4.0.4 (current release)

OS: macOS 64-bit x64

Package: TGZ

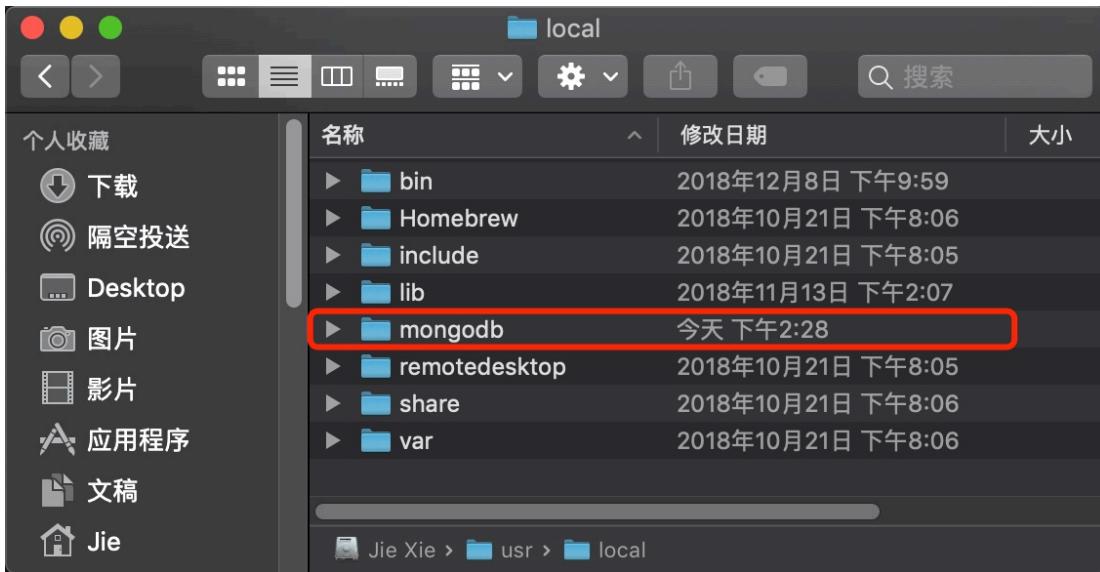
Download

https://fastdl.mongodb.org/osx/mongodb-osx-ssl-x86_64-4.0.4.tgz

- Release notes
- Changelog
- All version binaries
- Installation instructions
- Download source (tgz)
- Download source (zip)

(图为 MongoDB 官网部分截图)

下载好压缩包以后，对压缩包进行解压，将解压后得到的目录放置到 `/usr/local` 目录下，如下：



(这里我将解压后的目录改名为 mongodb)

也可以打开 Mac 的终端，全程使用命令行来进行下载和安装

进入 /usr/local 目录: `cd /usr/local`

下载压缩包: `sudo curl -O https://fastdl.mongodb.org/osx/mongodb-osx-x86_64-3.4.2.tgz`

进行解压: `sudo tar -zxvf mongodb-osx-x86_64-3.4.2.tgz`

目录重命名为 mongodb: `sudo mv mongodb-osx-x86_64-3.4.2 mongodb`

放置完毕后，我们的 MongoDB 就算是安装完毕了。但是每次调用 MongoDB 的相应命令需要切换到所在目录下的 bin 目录下面，这样显得非常的麻烦，我们可以为我们的 Mac 配置全局环境变量。命令如下：

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

注：此命令是临时的，每次重启终端后都需要重新输入一次。

配置完全局环境变量后，我们就不再需要进入 mongodb 目录下的 bin 目录里面来执行命令，直接在任何位置都可以执行命令。例如我这里输入测试命令 `mongod --version`

```
Last login: Thu Dec 13 14:43:45 on ttys000
[Jie-Xie:~ Jie$ export PATH=/usr/local/mongodb/bin:$PATH
[Jie-Xie:~ Jie$ mongod --version
db version v4.0.4
git version: f288a3bdf201007f3693c58e140056adf8b04839
allocator: system
modules: none
build environment:
    distarch: x86_64
    target_arch: x86_64
```

Windows 系统安装 MongoDB

上面介绍了如何在 Mac 系统下安装 MongoDB。相比 Mac 系统，Windows 操作系统下提供了安装 MongoDB 的可执行文件，具体操作这里不再进行示例，详细可以参见下面的博文：

<http://www.runoob.com/mongodb/mongodb-window-install.html>

MongoDB 配置和启动

使用 MongoDB 之前，需要配置数据库所在的位置，配置的命令如下：

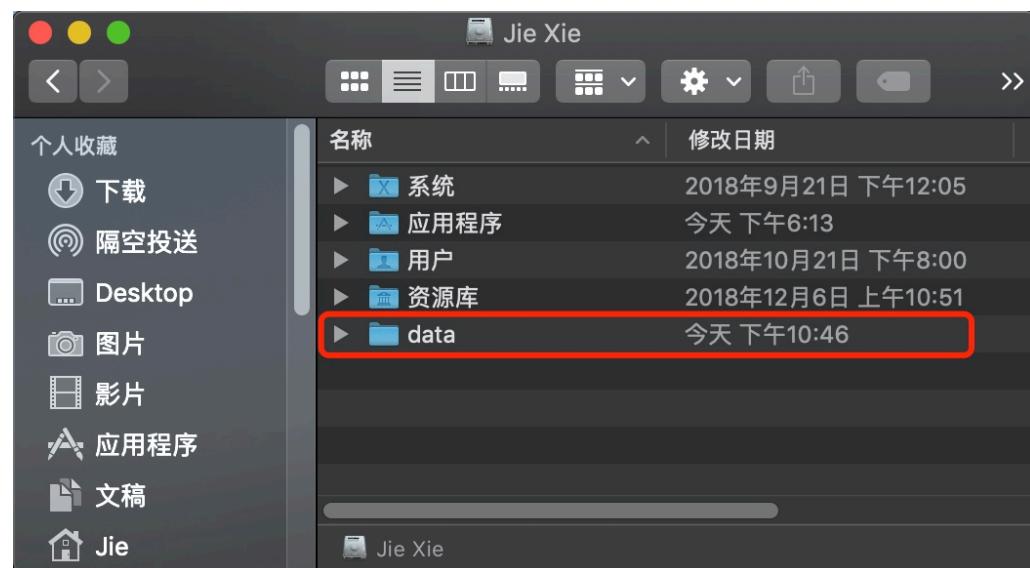
```
sudo mkdir -p 路径
```

示例如下：

```
Last login: Thu Dec 13 22:39:58 on ttys001
[Jie-Xie:~ Jie$ sudo mkdir -p /data/db
[Password:
Jie-Xie:~ Jie$
```

注：Mac 中将目录或者文件拖入到终端，可以快速获取到当前目录或文件的路径。

这里我们就配置好了数据库的位置，数据库位于根目录下：



位置配置好以后，可以使用终端来启动 MongoDB，命令为：

```
sudo mongod
```

如果没有创建全局路径，则需要进入到以下目录：

```
cd /usr/local/mongodb/bin  
sudo ./mongod
```

具体操作如下图所示：

```
Jie-Xie:~ Jie$ cd /usr/local/mongodb/bin  
[Jie-Xie:bin Jie$ sudo ./mongod  
Password:  
2018-12-13T22:52:32.881+0800 I CONTROL [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'  
2018-12-13T22:52:32.892+0800 I CONTROL [initandlisten] MongoDB starting : pid=981 port=27017 dbpath=/data/db 64-bit host=Jie-Xie.lan  
2018-12-13T22:52:32.892+0800 I CONTROL [initandlisten] db version v4.0.4  
2018-12-13T22:52:32.892+0800 I CONTROL [initandlisten] git version: f288a3bdf201007f3693c58e140056adf8b04839  
2018-12-13T22:52:32.892+0800 I CONTROL [initandlisten] allocator: system  
2018-12-13T22:52:32.892+0800 I CONTROL [initandlisten] modules: none  
2018-12-13T22:52:32.892+0800 I CONTROL [initandlisten] build environment:  
2018-12-13T22:52:32.892+0800 I CONTROL [initandlisten] distarch: x86_64  
2018-12-13T22:52:32.892+0800 I CONTROL [initandlisten] target_arch: x86_64  
2018-12-13T22:52:32.892+0800 I CONTROL [initandlisten] options: {}  
2018-12-13T22:52:32.893+0800 I STORAGE [initandlisten] wiretiger_open config: create,cache_size=3584M,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),statistics_log=(wait=0),verbose=(recovery_progress),  
2018-12-13T22:52:33.584+0800 I STORAGE [initandlisten] WiredTiger message [1544712753:584105]{[981:0x1122975c0]}, txn-recover: Set global recovery timestamp: 0  
2018-12-13T22:52:33.664+0800 I RECOVERY [initandlisten] WiredTiger recoveryTimestamp. Ts: Timestamp(0, 0)  
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten]  
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.  
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.  
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** WARNING: You are running this process as the root user, which is not recommended.  
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten]  
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** WARNING: This server is bound to localhost.  
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** Remote systems will be unable to connect to this server.  
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** Start the server with --bind_ip <address> to specify which IP  
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** addresses it should serve responses from, or with --bind_ip_all to  
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** bind to all interfaces. If this behavior is desired, start the  
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** server with --bind_ip 127.0.0.1 to disable this warning.  
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten]  
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** WARNING: soft rlimits too low. Number of files is 256, should be at least 1000  
2018-12-13T22:52:33.773+0800 I STORAGE [initandlisten] createCollection: admin.system.version with provided UUID: 86cf6884-de6c-4e0a-855c-68b671578216  
2018-12-13T22:52:33.868+0800 I COMMAND [initandlisten] setting featureCompatibilityVersion to 4.0  
2018-12-13T22:52:33.872+0800 I STORAGE [initandlisten] createCollection: local.startup_log with generated UUID: 4d887b6e-2ffb-443d-a3fb-c4b7008c811f  
2018-12-13T22:52:33.962+0800 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory '/data/db/diagnostic.data'  
2018-12-13T22:52:33.966+0800 I NETWORK [initandlisten] waiting for connections on port 27017  
2018-12-13T22:52:33.966+0800 I STORAGE [LogicalSessionCacheRefresh] createCollection: config.system.sessions with generated UUID: d8ee2feb-8f34-4e39-b34b-0478060590e4  
2018-12-13T22:52:34.104+0800 I INDEX [LogicalSessionCacheRefresh] build index on: config.system.sessions properties: { v: 2, key: { lastUse: 1 }, name: "lsidTTLIndex", ns: "con  
fig.system.sessions", expireAfterSeconds: 1800 }  
2018-12-13T22:52:34.104+0800 I INDEX [LogicalSessionCacheRefresh] building index using bulk method; build may temporarily use up to 500 megabytes of RAM  
2018-12-13T22:52:34.124+0800 I INDEX [LogicalSessionCacheRefresh] build index done. scanned 0 total records. 0 secs  
2018-12-13T22:52:34.124+0800 I COMMAND [LogicalSessionCacheRefresh] command config.$cmd command: createIndexes { createIndexes: "system.sessions", indexes: [ { key: { lastUse: 1  
}, name: "lsidTTLIndex", expireAfterSeconds: 1800 } ], $db: "config" } numYields:0 reslen:114 locks: { Global: { acquireCount: { w: 2, w: 2 } }, Database: { acquireCount: { w: 2, W  
: 1 } }, Collection: { acquireCount: { w: 2 } } } protocol:op_msg 158ms  
2018-12-13T22:53:07.399+0800 I NETWORK [listener] connection accepted from 127.0.0.1:49839 #1 (1 connection now open)  
2018-12-13T22:53:07.399+0800 I NETWORK [conn1] received client metadata from 127.0.0.1:49839 conn1: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Cl  
ient", version: "4.0.4" }, os: { type: "Darwin", name: "Mac OS X", architecture: "x86_64", version: "18.0.0" } }
```

至此，我们就已经启动了 MongoDB 的服务端。

接下来，新开启一个终端，输入 `./mongo` 命令来连接服务器。

如果没有配置全局路径，则需要进入 `/usr/local/mongodb/bin` 目录下。

具体操作如下图所示：

```
Last login: Thu Dec 13 22:48:39 on ttys001
[Jie-Xie:~ Jie$ cd /usr/local/mongodb/bin
[Jie-Xie:bin Jie$ ./mongo
MongoDB shell version v4.0.4
connecting to: mongodb://127.0.0.1:27017
Implicit session: session { "id" : UUID("94211058-e4d8-4e20-a58a-3dde686a5929") }
MongoDB server version: 4.0.4
Server has startup warnings:
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten]
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** WARNING: You are running this process as the root user, which is not recommended.
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten]
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** WARNING: This server is bound to localhost.
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** Remote systems will be unable to connect to this server.
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** Start the server with --bind_ip <address> to specify which IP
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** addresses it should serve responses from, or with --bind_ip_all to
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** bind to all interfaces. If this behavior is desired, start the
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** server with --bind_ip 127.0.0.1 to disable this warning.
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten]
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** WARNING: soft rlimits too low. Number of files is 256, should be at least 1000
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

[> 1+1
2
> ]
```

至此，我们已经成功的启动和连接上了 MongoDB，我们可以使用 `db` 命令来查看当前使用的是哪个数据库，默认连接的是 `test` 数据库如下：

```
Last login: Thu Dec 13 22:48:39 on ttys001
[Jie-Xie:~ Jie$ cd /usr/local/mongodb/bin
[Jie-Xie:bin Jie$ ./mongo
MongoDB shell version v4.0.4
connecting to: mongodb://127.0.0.1:27017
Implicit session: session { "id" : UUID("94211058-e4d8-4e20-a58a-3dde686a5929") }
MongoDB server version: 4.0.4
Server has startup warnings:
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten]
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** WARNING: You are running this process as the root user, which is not recommended.
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten]
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** WARNING: This server is bound to localhost.
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** Remote systems will be unable to connect to this server.
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** Start the server with --bind_ip <address> to specify which IP
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** addresses it should serve responses from, or with --bind_ip_all to
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** bind to all interfaces. If this behavior is desired, start the
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** server with --bind_ip 127.0.0.1 to disable this warning.
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten]
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** WARNING: soft rlimits too low. Number of files is 256, should be at least 1000
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

[> 1+1
2
> db
test
> ]
```

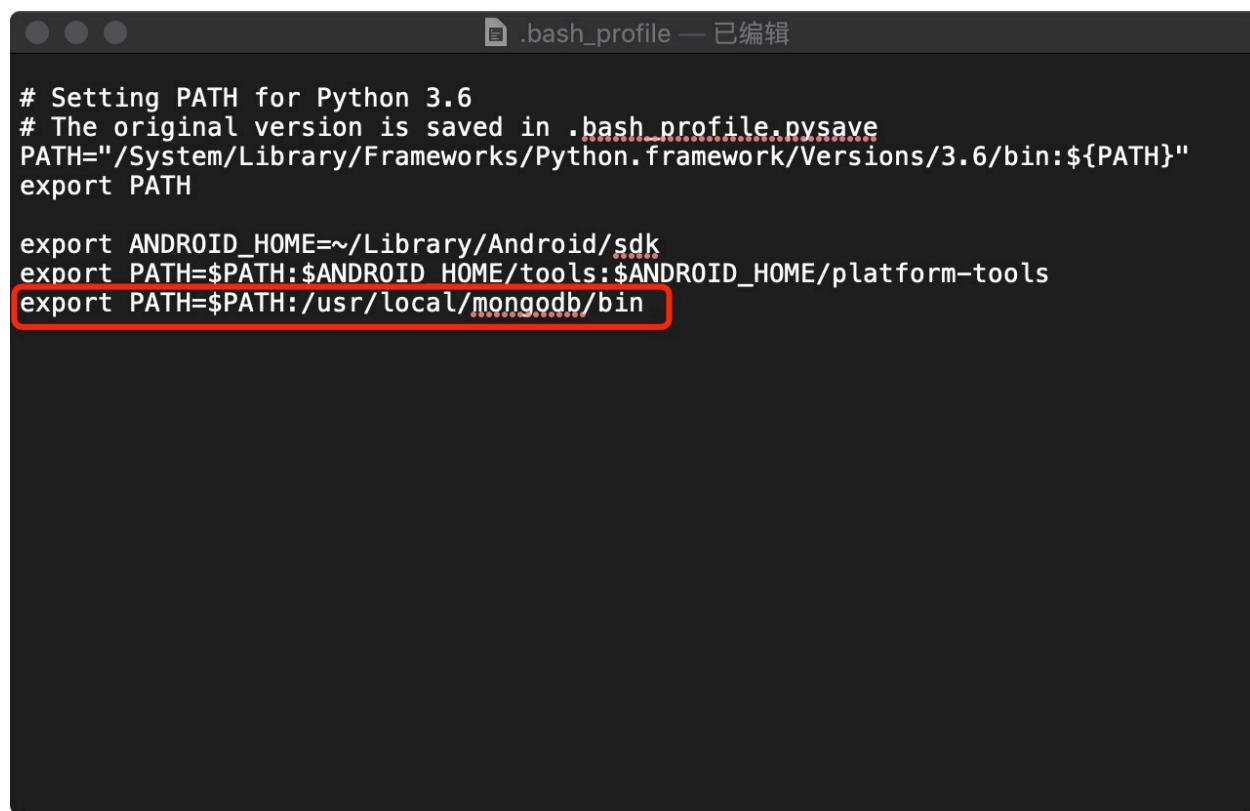
配置环境变量

每次启动 MongoDB 都要切换到对应的 bin 目录，然后执行 mongod 命令才能启动，非常的麻烦，这里我们可以为 mongodb 配置一个环境变量。

以 Mac 系统为例：（Windows 系统自带了 exe 启动文件）

- (1) 打开终端，输入 `open -e .bash_profile`，这时会自动打开环境变量的配置文件。
- (2) 输入 `export PATH=$PATH:MongoDB 所在路径`。例如我电脑上的 MongoDB 所在路径为 `/usr/local/mongodb/bin`，所以我应该书写 `export PATH=$PATH:/usr/local/mongodb/bin`

如下图所示：



```
# Setting PATH for Python 3.6
# The original version is saved in .bash_profile.pysave
PATH="/System/Library/Frameworks/Python.framework/Versions/3.6/bin:${PATH}"
export PATH

export ANDROID_HOME=~/Library/Android/sdk
export PATH=$PATH:$ANDROID_HOME/tools:$ANDROID_HOME/platform-tools
export PATH=$PATH:/usr/local/mongodb/bin
```

- (3) 输入 `source .bash_profile`，应用新设置的环境变量

至此，我们的环境变量就已设置完毕，之后输入 `sudo mongod` 即可开启 MongoDB 服务器端，如下：

Jie-Xie:~ Jie\$ █

█

输入 `mongo` 也可以直接连接上 MongoDB 服务器，如下：

```
Last login: Mon May 27 23:07:31 on ttys002
[Jie-Xie:~ Jie$ mongo
MongoDB shell version v4.0.4
connecting to: mongodb://127.0.0.1:27017
Implicit session: session { "id" : UUID("eeefb5f8-bab1-4996-b3dc-4dfe9e7bd87e") }
MongoDB server version: 4.0.4
Server has startup warnings:
2019-05-27T23:08:15.076+0800 I CONTROL [initandlisten]
2019-05-27T23:08:15.076+0800 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2019-05-27T23:08:15.076+0800 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.
2019-05-27T23:08:15.076+0800 I CONTROL [initandlisten] ** WARNING: You are running this process as the root user, which is not recommended.
2019-05-27T23:08:15.076+0800 I CONTROL [initandlisten]
2019-05-27T23:08:15.076+0800 I CONTROL [initandlisten] ** WARNING: This server is bound to localhost.
2019-05-27T23:08:15.076+0800 I CONTROL [initandlisten] ** Remote systems will be unable to connect to this server.
2019-05-27T23:08:15.077+0800 I CONTROL [initandlisten] ** Start the server with --bind_ip <address> to specify which IP
2019-05-27T23:08:15.077+0800 I CONTROL [initandlisten] ** addresses it should serve responses from, or with --bind_ip_all to
2019-05-27T23:08:15.077+0800 I CONTROL [initandlisten] ** bind to all interfaces. If this behavior is desired, start the
2019-05-27T23:08:15.077+0800 I CONTROL [initandlisten] ** server with --bind_ip 127.0.0.1 to disable this warning.
2019-05-27T23:08:15.077+0800 I CONTROL [initandlisten]
2019-05-27T23:08:15.077+0800 I CONTROL [initandlisten]
2019-05-27T23:08:15.077+0800 I CONTROL [initandlisten] ** WARNING: soft rlimits too low. Number of files is 256, should be at least 1000
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

> 1+1
2
> |
```

退出 MongoDB

如果要结束 MongoDB 的使用，一定要正常退出，不然下一次再连接数据库时可能会遇到问题。退出 MongoDB 的命令如下：

```
use admin;
db.shutdownServer();
```

具体操作如下图所示：

```
Last login: Thu Dec 13 22:48:39 on ttys001
[Jie-Xie:~ Jie$ cd /usr/local/mongodb/bin
[Jie-Xie:bin Jie$ ./mongo
MongoDB shell version v4.0.4
connecting to: mongodb://127.0.0.1:27017
Implicit session: session { "id" : UUID("94211058-e4d8-4e20-a58a-3dde686a5929") }
MongoDB server version: 4.0.4
Server has startup warnings:
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten]
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** WARNING: You are running this process as the root user, which is not recommended.
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten]
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** WARNING: This server is bound to localhost.
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** Remote systems will be unable to connect to this server.
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** Start the server with --bind_ip <address> to specify which IP
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** addresses it should serve responses from, or with --bind_ip_all to
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** bind to all interfaces. If this behavior is desired, start the
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** server with --bind_ip 127.0.0.1 to disable this warning.
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten]
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten]
2018-12-13T22:52:33.773+0800 I CONTROL [initandlisten] ** WARNING: soft rlimits too low. Number of files is 256, should be at least 1000
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

[> 1+1
2
[> db
test
[> use admin;
switched to db admin
[> use db;
switched to db db
[> db.shutdownServer();
shutdown command only works with the admin database; try 'use admin'
[> use admin;
switched to db admin
[> db.shutdownServer();
server should be down...
2018-12-13T23:03:17.180+0800 I NETWORK [js] trying reconnect to 127.0.0.1:27017 failed
2018-12-13T23:03:17.180+0800 I NETWORK [js] reconnect 127.0.0.1:27017 failed failed
> ]
```

注：退出 MongoDB 服务器必须先切换到 admin 数据库下。

常见问题以及解决方案

1. 创建 lock file 失败

有些时候我们在退出了 MongoDB 之后再次启动时，可能会遇到如下的问题：

```
MongoDB: exception in initAndListen: 20 Attempted to create a lock file on a read-only directory: /data/db, terminal
```

这是由于 db 目录里面的 lock file 文件出了问题。系统需要创建一个 lock file 文件，但是目录是只读的，输入如下命令即可解决：

```
sudo chown -R `id -u` /data/db
sudo chmod -R go+w /data/db
```

or

```
sudo chown -R $USER /data/db  
sudo chmod -R go+w /data/db
```

2. 端口被占用

有些时候可能还会遇到端口被占用的问题，如下：

```
listen(): bind() failed Address already in use for socket: 0.0.0.0:27017
```

这是因为我们之前所启动的 MongoDB 没有正常退出，端口还被占用着的。解决方案如下：

首先通过 `ps aux|grep mongod` 来查看已经开启的 MongoDB 进程，然后使用 `kill -9 进程参数` 命令来关闭掉 mongod 所对应的进程。

根据 Mac 系统的版本可能会涉及到要关闭保护模式的操作，具体可以参见博文：<https://blog.csdn.net/u012377393/article/details/71601240>

具体操作示例如下：

```
[Jie-Xie:bin Jie$ ps aux | grep mongod  
Jie          2525  0.3  0.3  4421980  27536  ??  S      4:58下午  0:03.78 ./mongod  
Jie          2622  0.0  0.0  4267752     880 s000  S+    5:07下午  0:00.00 grep mongod  
[Jie-Xie:bin Jie$ kill -9 2622  
[-bash: kill: (2622) - No such process  
[Jie-Xie:bin Jie$ cd ..  
[Jie-Xie:mongodb Jie$ kill -9 2622  
[-bash: kill: (2622) - No such process  
[Jie-Xie:mongodb Jie$ cd ../../..  
[Jie-Xie:usr Jie$ kill -9 2622  
[-bash: kill: (2622) - No such process  
[Jie-Xie:usr Jie$ kill -9 2525  
[Jie-Xie:usr Jie$  
[Jie-Xie:usr Jie$  
[Jie-Xie:usr Jie$ cd /usr/local/mongodb/bin           重启即可  
[Jie-Xie:bin Jie$ ./mongod
```

如果认为操作复杂，也可以通过重启操作系统的方式来解决端口被占用的问题。

4. 数据库与集合相关操作

本文主要包含以下知识点：

- 概念解析
- 数据库相关操作
- 集合相关操作

概念解析

MongoDB 一直被誉为"最像关系型数据库的非关系型数据库"。那么它们之间有哪些相似之处呢？

首先，MongoDB 中是有数据库这个概念的。一个数据库中包含若干个集合，每个集合中包含若干个文档。

文档是 MongoDB 中存储数据最基础单位，所有存储在 MongoDB 中的数据，都应该以文档的形式来组织。在每个集合中的若干文档，都是有一定格式的，该格式类似于 JSON。

具体对比如下：

RDBMS	MongoDB
Database	Database
Table (表格)	Collection (集合)
Row (行)	Document (文档)
Column (列)	Field (字段)
PrimaryKey (主键)	PrimaryKey (自动提供主键)

数据库相关操作

查看数据库

查看数据库的命令为 `show dbs`，示例如下：

```
> show dbs
admin  0.000GB
config  0.000GB
local   0.000GB
```

创建数据库

创建数据库的命令为 `use 数据库名`。系统会自动检测该数据库是否存在，如果存在，就切换到此数据库下，如果不存在，就会创建一个新的数据库。示例如下：

```
> use stuInfo;
switched to db stuInfo
> show dbs;
admin    0.000GB
config   0.000GB
local    0.000GB
```

这里我们就创建了一个名为 `stuInfo` 的数据库。

但是当我们使用 `show dbs` 命令来查看时，却发现看不到该数据库。这是因为该数据库里面还没有任何的数据。我们向该数据库里面添加一条测试数据，然后再次进行查看，如下：

```
> db.class1.insert({"name":"xiejie"})
WriteResult({ "nInserted" : 1 })
> show dbs
admin    0.000GB
config   0.000GB
local    0.000GB
stuInfo  0.000GB
```

我们向 `stuInfo` 数据库中添加了一个名为 `class1` 的集合，然后向该集合里面添加了一个文档。之后再使用 `show dbs` 进行查看时，可以看到这一次显示出了 `stuInfo` 数据库。

删除数据库

删除数据库的命令为 `db.dropDatabase()`。调用该方法后会删除当前所使用的数据库，如下：

```
> db.dropDatabase()
{ "dropped" : "stuInfo", "ok" : 1 }
> show dbs;
admin    0.000GB
config   0.000GB
local    0.000GB
```

集合相关操作

一个集合，类似于关系型数据库中的一张表。下面介绍集合的常见操作。

创建集合

创建集合的命令如下：

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

参数说明：

name：要创建的集合名称。

options：可选参数，指定有关内存大小以及索引的选项。其中 options 可以是如下的参数

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

(该图来源自：<http://www.runoob.com/mongodb/mongodb-create-collection.html>)

接下来我们来看一下具体的示例：

```
> use test
switched to db test
> db.createCollection("class1")
{ "ok" : 1 }
```

这里我们就创建了一个名为 class1 的集合。

下面的例子演示了创建具有 options 参数的集合，如下：

```
> use test
switched to db test
> db.createCollection("mycol", { capped : true, autoIndexId : true, size : 6142800, max : 10000 } )
{
  "note" : "the autoIndexId option is deprecated and will be removed in a future release",
  "ok" : 1
```

```
}
```

这里，我们就创建了固定集合 mycol，整个集合空间大小 6142800 KB，文档最大个数为 10000 个。

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

值得一提的是，在 MongoDB 中，其实不需要我们显式的来创建集合。当我们插入一些文档时，MongoDB 会自动创建集合。如下：

```
> show collections
class1
col
mycol
> db.mycol2.insert({ "name": "xiejie" })
WriteResult({ "nInserted" : 1 })
> show collections
class1
col
mycol
mycol2
```

在上面的例子中，我们并没有创建 mycol2 集合，但是当我们直接向 mycol2 集合插入文档时，MongoDB 会自动创建 mycol2 集合。

查看集合

查看集合的命令为 `show collections`，具体的操作示例如下：

```
> show collections
class1
col
mycol
mycol2
```

注：该命令有 1 个别名为 `show tables`

删除集合

删除集合的命令为 `db.集合名.drop()`，具体的操作示例如下：

```
> db.mycol.drop()
true
```

```
> show collections
class1
col
mycol2
```

5. 文档基本操作

本文主要包含以下知识点：

- 插入文档
- 更新文档
- 删除文档

插入文档

向 MongoDB 中插入的文档数据结构和 JSON 基本一样。所有存储在集合中的数据都是 BSON 格式。

注：BSON 是一种类似于 JSON 的二进制存储格式，全称为 *Binary JSON*。

在 MongoDB 中，使用 `insert()` 或者 `save()` 方法来向集合中插入文档。示例如下：

```
[> use test
switched to db test
[> db.family.insert({"name": "xiejie", "age": 18, "gender": "male", "score": 100})
WriteResult({ "nInserted" : 1 })
[> db.family.insert({"name": "yajing", "age": 20, "gender": "female", "score": 90})
WriteResult({ "nInserted" : 1 })
[> db.family.find()
{ "_id" : ObjectId("5c1342fa78b54b274a3c9fec"), "name" : "xiejie", "age" : 18, "gender" : "male", "score" : 100 }
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 90 }
[>
```

也可以将数据存储在一个变量里面，然后直接插入此变量，具体的示例如下：

```
[> someone = {"name": "xizhi", "age": 2, "gender": "male", "score": 10}
{ "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
[> db.family.insert(someone)
WriteResult({ "nInserted" : 1 })
[> db.family.find()
{ "_id" : ObjectId("5c1342fa78b54b274a3c9fec"), "name" : "xiejie", "age" : 18, "gender" : "male", "score" : 100 }
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 90 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
[>
```

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

MongoDB 3.2 版本后还新增了以下几种语法来插入文档：

`db.collection.insertOne()` 方法向指定集合中插入一条文档数据。

`db.collection.insertMany()` 方法向指定集合中插入多条文档数据。

下面演示一下使用 `db.collection.insertMany()` 方法来插入多条记录，如下：

```
[> use test
switched to db test
[> db.family.insertMany([{"name": "佐佐木希", "age": 30, "gender": "female"}, {"name": "孙悟空", "age": 100, "gender": "male"}])
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("5c1344fc78b54b274a3c9fef"),
        ObjectId("5c1344fc78b54b274a3c9ff0")
    ]
}
[> db.family.find()
{ "_id" : ObjectId("5c1342fa78b54b274a3c9fec"), "name" : "xiejie", "age" : 18, "gender" : "male", "score" : 100 }
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 90 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木希", "age" : 30, "gender" : "female" }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9ff0"), "name" : "孙悟空", "age" : 100, "gender" : "male" }
[>
```

更新文档

MongoDB 中可以使用 `update()` 和 `save()` 方法来更新集合中的文档。

`update()` 方法具体的语法如下：

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

`update()` 方法中接收 3 个参数，对应的说明如下：

- `query`: `update`的查询条件，类似sql `update`查询内`where`后面的。
- `update`: `update`的对象和一些更新的操作符（如,`inc...`）等，也可以理解为sql `update`查询内`set`后面的。
- `upsert`: 可选，这个参数的意思是，如果不存在`update`的记录，是否插入`objNew`,`true`为插入，默认是`false`，不插入。
- `multi`: 可选，mongodb 默认是`false`,只更新找到的第一条记录，如果这个参数为`true`,就把按条件查出来多条记录全部更新。
- `writeConcern`: 可选，抛出异常的级别。

下面是 `update()` 方法的一个具体示例：

首先我们向 `family` 集合中插入 2 条相同的数据。

```
|> db.family.insertMany([{"name": "孙悟空", "age": 200, "gender": "male"}, {"name": "孙悟空", "age": 300, "gender": "female"}])
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("5c1355d5bd36aa3e32644aab"),
        ObjectId("5c1355d5bd36aa3e32644aac")
    ]
}
|> db.family.find()
{ "_id" : ObjectId("5c1342fa78b54b274a3c9fec"), "name" : "xiejie", "age" : 18, "gender" : "male", "score" : 100 }
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 90 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木希", "age" : 30, "gender" : "female" }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9ff0"), "name" : "孙悟空", "age" : 100, "gender" : "male" }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aab"), "name" : "孙悟空", "age" : 200, "gender" : "male" }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aac"), "name" : "孙悟空", "age" : 300, "gender" : "female" }
|>
|>
```

接下来我们将 `name` 为“孙悟空”的人的 `age` 属性修改为 1000，如下：

```
|> db.family.find()
{ "_id" : ObjectId("5c1342fa78b54b274a3c9fec"), "name" : "xiejie", "age" : 18, "gender" : "male", "score" : 100 }
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 90 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木希", "age" : 30, "gender" : "female" }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9ff0"), "name" : "孙悟空", "age" : 100, "gender" : "male" }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aab"), "name" : "孙悟空", "age" : 200, "gender" : "male" }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aac"), "name" : "孙悟空", "age" : 300, "gender" : "female" }
|>
|> db.family.update({ 'name': '孙悟空' }, { '$set': { 'age': 1000 } })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
|> db.family.find()
{ "_id" : ObjectId("5c1342fa78b54b274a3c9fec"), "name" : "xiejie", "age" : 18, "gender" : "male", "score" : 100 }
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 90 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木希", "age" : 30, "gender" : "female" }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9ff0"), "name" : "孙悟空", "age" : 1000, "gender" : "male" }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aab"), "name" : "孙悟空", "age" : 200, "gender" : "male" }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aac"), "name" : "孙悟空", "age" : 300, "gender" : "female" }
|>
|>
```

如果要修改多条相同的文档，则需要设置 `multi` 参数为 `true`。如下：

```
|> db.family.find()
{ "_id" : ObjectId("5c1342fa78b54b274a3c9fec"), "name" : "xiejie", "age" : 18, "gender" : "male", "score" : 100 }
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 90 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木希", "age" : 30, "gender" : "female" }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9ff0"), "name" : "孙悟空", "age" : 1000, "gender" : "male" }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aab"), "name" : "孙悟空", "age" : 200, "gender" : "male" }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aac"), "name" : "孙悟空", "age" : 300, "gender" : "female" }
|>
|> db.family.update({'name':'孙悟空'},{$set:{'age':1000}},{'multi:true'})
WriteResult({ "nMatched" : 3, "nUpserted" : 0, "nModified" : 2 })
|> db.family.find()
{ "_id" : ObjectId("5c1342fa78b54b274a3c9fec"), "name" : "xiejie", "age" : 18, "gender" : "male", "score" : 100 }
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 90 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木希", "age" : 30, "gender" : "female" }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9ff0"), "name" : "孙悟空", "age" : 1000, "gender" : "male"  }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aab"), "name" : "孙悟空", "age" : 1000, "gender" : "male"  }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aac"), "name" : "孙悟空", "age" : 1000, "gender" : "female"  }
|>
```

`save()` 方法的特点为通过传入的文档来替换已有文档。语法格式如下：

```
db.collection.save(
  <document>,
  {
    writeConcern: <document>
  }
)
```

参数说明：

- `document`: 文档数据。
- `writeConcern`: 可选，抛出异常的级别。

下面是一个使用 `save()` 方法来更新文档的具体示例，如下：

```

|> db.family.find()
{ "_id" : ObjectId("5c1342fa78b54b274a3c9fec"), "name" : "xiejie", "age" : 18, "gender" : "male", "score" : 100 }
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 90 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木希", "age" : 30, "gender" : "female" }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9ff0"), "name" : "孙悟空", "age" : 1000, "gender" : "male" }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aab"), "name" : "孙悟空", "age" : 1000, "gender" : "male" }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aac"), "name" : "孙悟空", "age" : 1000, "gender" : "female" }
>
>
|> db.family.save({"_id":ObjectId("5c1344fc78b54b274a3c9fef")}, {"name":"佐佐木", "age":22, "gender":"male"})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
|> db.family.find()
{ "_id" : ObjectId("5c1342fa78b54b274a3c9fec"), "name" : "xiejie", "age" : 18, "gender" : "male", "score" : 100 }
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 90 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male" } // Updated document
{ "_id" : ObjectId("5c1344fc78b54b274a3c9ff0"), "name" : "孙悟空", "age" : 1000, "gender" : "male" }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aab"), "name" : "孙悟空", "age" : 1000, "gender" : "male" }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aac"), "name" : "孙悟空", "age" : 1000, "gender" : "female" }
>

```

同样，从 MongoDB 3.2 版本开始，MongoDB 提供以下更新集合文档的方法：

`db.collection.updateOne()` 方法向指定集合更新单个文档。

`db.collection.updateMany()` 方法向指定集合更新多个文档。

这里我们演示一下使用 `db.collection.updateMany()` 方法来更新多个文档，如下：

```

|> db.family.find()
{ "_id" : ObjectId("5c1342fa78b54b274a3c9fec"), "name" : "xiejie", "age" : 18, "gender" : "male", "score" : 100 }
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 90 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male" }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9ff0"), "name" : "孙悟空", "age" : 1000, "gender" : "male" }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aab"), "name" : "孙悟空", "age" : 1000, "gender" : "male" }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aac"), "name" : "孙悟空", "age" : 1000, "gender" : "female" }
>
|> db.family.updateMany({"age":{$gt:18}}, {$set:{'score':60}})
{ "acknowledged" : true, "matchedCount" : 5, "modifiedCount" : 5 }
|> db.family.find()
{ "_id" : ObjectId("5c1342fa78b54b274a3c9fec"), "name" : "xiejie", "age" : 18, "gender" : "male", "score" : 100 }
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9ff0"), "name" : "孙悟空", "age" : 1000, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aab"), "name" : "孙悟空", "age" : 1000, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aac"), "name" : "孙悟空", "age" : 1000, "gender" : "female", "score" : 60 }
>

```

本示例中，我们将 age 大于 18 的文档的 score 字段更新为 60，并且不管符合要求的文档之前有没有 score 字段，更新后都统一拥有 score 字段，值为 60。

删除文档

MongoDB 中使用 `remove()` 方法来删除文档。语法如下：

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

具体的参数说明：

- query：（可选）删除的文档的条件。
- justOne：（可选）如果设为 true 或 1，则只删除一个文档，如果不设置该参数，或使用默认值 false，则删除所有匹配条件的文档。
- writeConcern：（可选）抛出异常的级别。

下面是使用 `remove()` 方法的一个具体示例，如下：

```
|> db.family.find()  
{ "_id" : ObjectId("5c1342fa78b54b274a3c9fec"), "name" : "xiejie", "age" : 18, "gender" : "male", "score" : 100 }  
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }  
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }  
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }  
{ "_id" : ObjectId("5c1344fc78b54b274a3c9ff0"), "name" : "孙悟空", "age" : 1000, "gender" : "male", "score" : 60 }  
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aab"), "name" : "孙悟空", "age" : 1000, "gender" : "male", "score" : 60 }  
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aac"), "name" : "孙悟空", "age" : 1000, "gender" : "female", "score" : 60 }  
{ "_id" : ObjectId("5c135c32bd36aa3e32644aad"), "name" : "xiejie", "age" : 18 }  
>  
> db.family.remove({ "name": "xiejie" })  
WriteResult({ "nRemoved" : 2 })  
>  
> db.family.find()  
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }  
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }  
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }  
{ "_id" : ObjectId("5c1344fc78b54b274a3c9ff0"), "name" : "孙悟空", "age" : 1000, "gender" : "male", "score" : 60 }  
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aab"), "name" : "孙悟空", "age" : 1000, "gender" : "male", "score" : 60 }  
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aac"), "name" : "孙悟空", "age" : 1000, "gender" : "female", "score" : 60 }  
>
```

使用 `remove()` 方法后，所有 name 值为 xiejie 的文档就全部被删除了。

如果只想删除第一条找到的记录，可以设置 justOne 为 1，如下所示：

```
> db.family.find()
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9ff0"), "name" : "孙悟空", "age" : 1000, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aab"), "name" : "孙悟空", "age" : 1000, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aac"), "name" : "孙悟空", "age" : 1000, "gender" : "female", "score" : 60 }
>
> db.family.remove({name:"孙悟空"},1)
WriteResult({ "nRemoved" : 1 })
>
> db.family.find()
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aab"), "name" : "孙悟空", "age" : 1000, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aac"), "name" : "孙悟空", "age" : 1000, "gender" : "female", "score" : 60 }
>
```

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

```
db.集合名.remove({})
```

值得一提的是，使用 `remove()` 方法来进行文档的删除时，并不会真正释放空间。所以需要继续执行 `db.repairDatabase()` 来回收磁盘空间。

```
> db.repairDatabase()
或者
> db.runCommand({ repairDatabase: 1 })
```

正因为 `remove()` 方法有此缺点，所以现在官方已经不再推荐使用 `remove()` 方法来删除文档，而是推荐使用 `deleteOne()` 和 `deleteMany()` 方法。

删除集合下全部文档：

```
db.集合名.deleteMany({})
```

删除 `status` 等于 A 的全部文档：

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

删除 `status` 等于 D 的一个文档：

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

下面的示例演示了使用 `deleteMany()` 方法来删除所有 `name` 值为“孙悟空”文档。

```
[> db.family.find()
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aab"), "name" : "孙悟空", "age" : 1000, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1355d5bd36aa3e32644aac"), "name" : "孙悟空", "age" : 1000, "gender" : "female", "score" : 60 }
[>
[> db.family.deleteMany({name:"孙悟空"})
{ "acknowledged" : true, "deletedCount" : 2 }
[>
[> db.family.find()
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
[>
```

6. 文档查询

本文主要包含以下知识点：

- 基本查询
- 条件查询
- 查询结果排序
- 查询结果限制
- 模糊查询

基本查询

MongoDB 中查询文档使用 `find()` 方法。`find()` 方法会以非结构化的方式来显示所有文档。语法如下：

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

参数说明：

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

具体的示例如下：

```
|> db.family.find()
{| "id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{| "id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{| "id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
|>
|>
```

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

```
db.集合名.find().pretty()
```

具体的示例如下：

```

[> db.family.find().pretty()
{
    "_id" : ObjectId("5c13431b78b54b274a3c9fed"),
    "name" : "yajing",
    "age" : 20,
    "gender" : "female",
    "score" : 60
}
{
    "_id" : ObjectId("5c1343b378b54b274a3c9fee"),
    "name" : "xizhi",
    "age" : 2,
    "gender" : "male",
    "score" : 10
}
{
    "_id" : ObjectId("5c1344fc78b54b274a3c9fef"),
    "name" : "佐佐木",
    "age" : 22,
    "gender" : "male",
    "score" : 60
}
>

```

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

条件查询

MongoDB 中的条件查询语句与 RDBMS 是非常类似的。下表列出了 MongoDB 与 RDBMS 里面 WHERE 语句的具体比较，如下：

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

(该图来源自：<http://www.runoob.com/mongodb/mongodb-query.html>)

下图我们选择一个条件操作符进行示例，如下：

```
[> db.family.find()
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1369b2bd36aa3e32644aae"), "name" : "孙悟空", "age" : 20, "gender" : "male" }
{ "_id" : ObjectId("5c137505bd36aa3e32644aaf"), "name" : "孙悟空", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137505bd36aa3e32644ab0"), "name" : "布尔玛", "age" : 18, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab1"), "name" : "佐佐木希", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab2"), "name" : "孙悟空", "age" : 300, "gender" : "female" }
>
> db.family.find({"age" : {$gte : 20}})
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1369b2bd36aa3e32644aae"), "name" : "孙悟空", "age" : 20, "gender" : "male" }
{ "_id" : ObjectId("5c137505bd36aa3e32644aaf"), "name" : "孙悟空", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab1"), "name" : "佐佐木希", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab2"), "name" : "孙悟空", "age" : 300, "gender" : "female" }
>
> ]
```

在本例中，我们查询出了 age 字段值大于等于 20 的文档有哪些。其他条件操作符的用法也可以以此类推，这里不再依次做演示。

AND 条件

MongoDB 的 `find()` 方法可以传入多个键 (key)，每个键 (key) 以逗号隔开，即常规 SQL 的 AND 条件。语法格式如下：

```
db.集合名.find({key1:value1, key2:value2})
```

具体示例如下：

```
[> db.family.find()
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1369b2bd36aa3e32644aae"), "name" : "孙悟空", "age" : 20, "gender" : "male" }
[>
[> db.family.find({"age":20})
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1369b2bd36aa3e32644aae"), "name" : "孙悟空", "age" : 20, "gender" : "male" }
[>
[> db.family.find({"age":20,"gender":"male"})
{ "_id" : ObjectId("5c1369b2bd36aa3e32644aae"), "name" : "孙悟空", "age" : 20, "gender" : "male" }
[>
[> ]
```

在本例中，如果只查询 age 字段为 20 的文档，则存在 2 条记录。如果查询 age 字段为 20 并且 gender 字段的值为 male，则只有 1 条记录。

注：类似于 SQL 语句中 WHERE 语句：`WHERE age=20 AND gender='male'`

OR 条件

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

```
db.集合名.find()
{
  $or: [
    {key1: value1}, {key2:value2}
  ]
}
)
```

具体的操作示例如下：

```
[> db.family.find()
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1369b2bd36aa3e32644aae"), "name" : "孙悟空", "age" : 20, "gender" : "male" }
[>
[> db.family.find({"age":20})
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1369b2bd36aa3e32644aae"), "name" : "孙悟空", "age" : 20, "gender" : "male" }
[>
[> db.family.find({$or : [{"age":20}, {"score":10}]}))
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1369b2bd36aa3e32644aae"), "name" : "孙悟空", "age" : 20, "gender" : "male" }
[>
[>
```

在本例中，如果只查询 age 字段为 20 的文档，则存在 2 条记录。如果查询 age 字段为 20 或者 name 字段的值为 xizhi，则有 3 条记录。

注：类似于 SQL 语句中 WHERE 语句：`WHERE age=20 OR name='xizhi'`

AND 和 OR 联合使用

以下实例演示了 AND 和 OR 联合使用：

```

|> db.family.find()
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1369b2bd36aa3e32644aae"), "name" : "孙悟空", "age" : 20, "gender" : "male" }
{ "_id" : ObjectId("5c137505bd36aa3e32644aaf"), "name" : "孙悟空", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137505bd36aa3e32644ab0"), "name" : "布尔玛", "age" : 18, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab1"), "name" : "佐佐木希", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab2"), "name" : "孙悟空", "age" : 300, "gender" : "female" }
|>
|> db.family.find({"gender": "female", $or: [{"age": 20}, {"name": "孙悟空"}]})
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c137505bd36aa3e32644aaf"), "name" : "孙悟空", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab1"), "name" : "佐佐木希", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab2"), "name" : "孙悟空", "age" : 300, "gender" : "female" }
|>
|>

```

注：类似常规 SQL 语句：*WHERE gender='female' AND (age = 20 OR name = '孙悟空')*

查询结果排序

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

基本语法如下所示：

```
db.集合名.find().sort({KEY:1})
```

这里我们按照 age 字段的大小来进行排序，具体的示例如下：

```

|> db.family.find()
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1369b2bd36aa3e32644aae"), "name" : "孙悟空", "age" : 20, "gender" : "male" }
{ "_id" : ObjectId("5c137505bd36aa3e32644aaf"), "name" : "孙悟空", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137505bd36aa3e32644ab0"), "name" : "布尔玛", "age" : 18, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab1"), "name" : "佐佐木希", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab2"), "name" : "孙悟空", "age" : 300, "gender" : "female" }
{ "_id" : ObjectId("5c13798bbd36aa3e32644ab3"), "name" : "中佐", "age" : 28, "gender" : "male" }
|>
|> db.family.find().sort({"age":1})
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c137505bd36aa3e32644ab0"), "name" : "布尔玛", "age" : 18, "gender" : "female" }
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1369b2bd36aa3e32644aae"), "name" : "孙悟空", "age" : 20, "gender" : "male" }
{ "_id" : ObjectId("5c137505bd36aa3e32644aaf"), "name" : "孙悟空", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab1"), "name" : "佐佐木希", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c13798bbd36aa3e32644ab3"), "name" : "中佐", "age" : 28, "gender" : "male" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab2"), "name" : "孙悟空", "age" : 300, "gender" : "female" }
|>
|>

```

下面的示例中演示了如果指定的第一个排序字段相同，那么可以指定第二个排序字段的情况，如下：

```
|>
|> db.family.find().sort({"age":1,"gender":-1})
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c137505bd36aa3e32644ab0"), "name" : "布尔玛", "age" : 18, "gender" : "female" }
{ "_id" : ObjectId("5c1369b2bd36aa3e32644aae"), "name" : "孙悟空", "age" : 20, "gender" : "male" }
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c137505bd36aa3e32644aaaf"), "name" : "孙悟空", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab1"), "name" : "佐佐木希", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c13798bbd36aa3e32644ab3"), "name" : "中佐", "age" : 28, "gender" : "male" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab2"), "name" : "孙悟空", "age" : 300, "gender" : "female" }
|>
|>
```

在本例中，我们首先指定按照 age 字段来进行排序，如果 age 字段的值相同，则按照 gender 字段来进行排序。

查询结果限制

有些时候，我们不需要一次性查询出所有符合条件的文档，而是需要对符合要求的文档进行分批显示，这个时候就需要对查询结果进行一个限制。

可以使用 MongoDB 的 `limit()` 方法，该方法接受一个数字参数，指定从 MongoDB 中读取的记录条数。

对应的语法如下：

```
db.集合名.find().limit(条数)
```

具体的使用示例如下：

```
|> db.family.find()
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1369b2bd36aa3e32644aae"), "name" : "孙悟空", "age" : 20, "gender" : "male" }
{ "_id" : ObjectId("5c137505bd36aa3e32644aaaf"), "name" : "孙悟空", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137505bd36aa3e32644ab0"), "name" : "布尔玛", "age" : 18, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab1"), "name" : "佐佐木希", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab2"), "name" : "孙悟空", "age" : 300, "gender" : "female" }
{ "_id" : ObjectId("5c13798bbd36aa3e32644ab3"), "name" : "中佐", "age" : 28, "gender" : "male" }
|>
|> db.family.find().limit(3)
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
|>
|>
```

在本例中，我们使用 `limit()` 方法来限制了我们的查询结果，仅显示前面 3 个文档。

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

```
db.集合名.find().limit(条数).skip(条数)
```

具体的使用示例如下：

```
[> db.family.find()
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1369b2bd36aa3e32644aae"), "name" : "孙悟空", "age" : 20, "gender" : "male" }
{ "_id" : ObjectId("5c137505bd36aa3e32644aaf"), "name" : "孙悟空", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137505bd36aa3e32644ab0"), "name" : "布尔玛", "age" : 18, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab1"), "name" : "佐佐木希", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab2"), "name" : "孙悟空", "age" : 300, "gender" : "female" }
{ "_id" : ObjectId("5c13798bbd36aa3e32644ab3"), "name" : "中佐", "age" : 28, "gender" : "male" }
[>
[> db.family.find().skip(5)
{ "_id" : ObjectId("5c137505bd36aa3e32644ab0"), "name" : "布尔玛", "age" : 18, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab1"), "name" : "佐佐木希", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab2"), "name" : "孙悟空", "age" : 300, "gender" : "female" }
{ "_id" : ObjectId("5c13798bbd36aa3e32644ab3"), "name" : "中佐", "age" : 28, "gender" : "male" }
[>
[> db.family.find().limit(5).skip(2)
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1369b2bd36aa3e32644aae"), "name" : "孙悟空", "age" : 20, "gender" : "male" }
{ "_id" : ObjectId("5c137505bd36aa3e32644aaf"), "name" : "孙悟空", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137505bd36aa3e32644ab0"), "name" : "布尔玛", "age" : 18, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab1"), "name" : "佐佐木希", "age" : 20, "gender" : "female" }
[>
[>
```

在本例中，第一次查询时使用 `skip()` 方法来跳过前面 5 个文档，显示出了第 5 个文档之后的所有文档。第二次查询时限制查询结果为 5 个，并且跳过了前面 2 个，所以显示出的文档为 3 – 7。

模糊查询

MongoDB 中同样还支持模糊查询。

例如：查询 `name` 字段以“佐”开头的文档，只需要传入一个正则表达式即可

```
[> db.family.find()
{ "_id" : ObjectId("5c13431b78b54b274a3c9fed"), "name" : "yajing", "age" : 20, "gender" : "female", "score" : 60 }
{ "_id" : ObjectId("5c1343b378b54b274a3c9fee"), "name" : "xizhi", "age" : 2, "gender" : "male", "score" : 10 }
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c1369b2bd36aa3e32644aae"), "name" : "孙悟空", "age" : 20, "gender" : "male" }
{ "_id" : ObjectId("5c137505bd36aa3e32644aaf"), "name" : "孙悟空", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137505bd36aa3e32644ab0"), "name" : "布尔玛", "age" : 18, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab1"), "name" : "佐佐木希", "age" : 20, "gender" : "female" }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab2"), "name" : "孙悟空", "age" : 300, "gender" : "female" }
{ "_id" : ObjectId("5c13798bbd36aa3e32644ab3"), "name" : "中佐", "age" : 28, "gender" : "male" }
[>
[> db.family.find({"name":/^佐/})
{ "_id" : ObjectId("5c1344fc78b54b274a3c9fef"), "name" : "佐佐木", "age" : 22, "gender" : "male", "score" : 60 }
{ "_id" : ObjectId("5c137568bd36aa3e32644ab1"), "name" : "佐佐木希", "age" : 20, "gender" : "female" }
[>
```

7. 可视化工具

工欲善其事，必先利其器。我们在使用数据库时，通常需要各种工具的支持来提高效率。

作为一个数据库软件，和 MySQL 数据库一样，其本身的使用包括查询和基本操作，都需要使用相关的命令，而在命令行中使用命令进行操作对于开发者而言是一件不方便的事情。尤其是新手开发者在没有熟练使用命令行时，这样的操作不亚于新学一门技术。

就像在 SQL Server 中的 SQL 查询一样，一个拥有界面和良好交互的 GUI 工具将极大地帮助 MongoDB 新用户，并为那些经常以多种语言查询的人节省宝贵的时间，让他们将更多的精力放在代码开发上。

目前 MongoDB 的可视化工具众多，其中 Studio 3T 比较热门，但是该工具是收费的。不过，该工具也有对应的免费版 Robo 3T。

Robo 3T 基本使用

我们可以在 Robo 3T 的官网上下载到该软件，官网地址为：<https://robomongo.org/>

The screenshot shows two side-by-side web pages. On the left is the Studio 3T website, featuring a large green rocket icon at the top. Below it, the text "Studio 3T" is prominently displayed. A subtext asks, "Are you serious about MongoDB? Choose Studio 3T - our fully featured IDE for MongoDB professionals." A bulleted list of features follows: "• Fully featured IDE with embedded shell", "• Visual Query Builder", "• In-Place editing", "• IntelliShell with Auto-Completion", "• Query MongoDB with SQL", "• Export to / import from SQL DB", "• Aggregation Pipeline Editor", and "• And so much more...". A green button labeled "Download Studio 3T" is at the bottom. A large green cartoon robot head is positioned below the text. On the right is the Robo 3T website, also featuring a large green rocket icon at the top. Below it, the text "Robo 3T" is prominently displayed. A subtext states, "Robo 3T (formerly Robomongo) is the free lightweight GUI for MongoDB enthusiasts." A bulleted list of features follows: "• MongoDB GUI with embedded shell". A green button labeled "Download Robo 3T" is at the bottom. A large green cartoon robot head is positioned below the text.

(图为 Robo 3T 官网部分截图)

下载安装完成后，打开页面如下：

Robo 3T - 1.2

Welcome

Release the full power of the mongo shell with Studio 3T

Write MongoDB queries faster with IntelliShell, Studio 3T's built-in mongo shell.

IntelliShell autocompletes:

- JavaScript standard library functions
- shell-specific types and methods
- operators
- collection names
- field names
- shell helper commands... and more

In addition to robust autocompletion, enjoy other features like code reformatting.

Logs

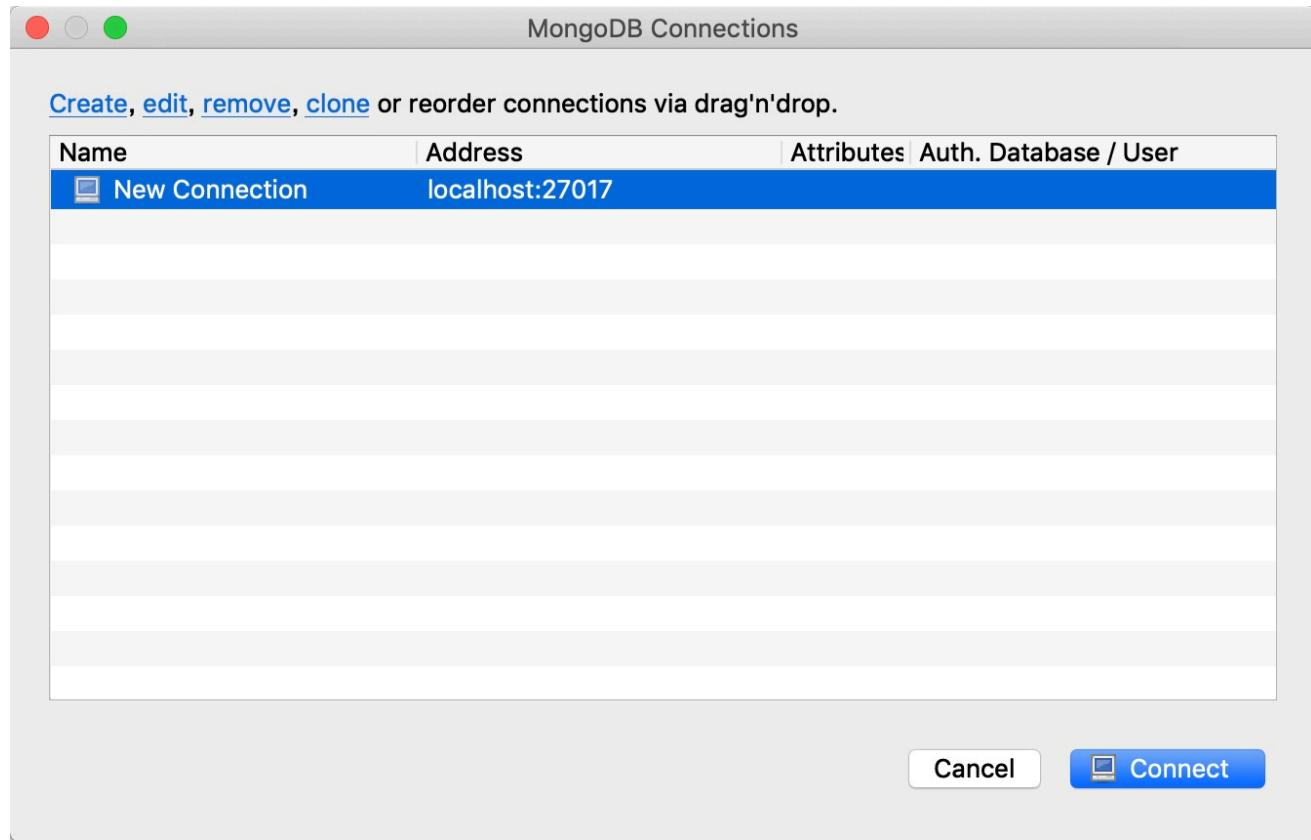
Blog Posts

- Robo 3T 1.2 is released
- Mon, 19 Feb 2018
- Robomongo is now Robomongo 3.4 support
- Wed, 14 Jun 2017
- Robomongo 1.0 — Off
- Thu, 20 Apr 2017
- Robomongo has been
- Tue, 14 Mar 2017
- Changing the name of
- Mon, 13 Feb 2017
- Robomongo 1.0 RC1 b
- Replica Set Clusters
- Thu, 02 Feb 2017
- Robomongo 0.9.0 Final
- Thu, 06 Oct 2016
- Robomongo RC10 brings
- Fri, 19 Aug 2016
- Robomongo RC9
- Wed, 01 Jun 2016
- Robomongo RC8

打开终端，启动 MongoDB 服务，如下：

```
Last login: Tue Jan 15 09:09:13 on console
[jie-xie:~ Jie$ cd /usr/local/mongodb/bin
[jie-xie:bin Jie$ sudo ./mongod
Password:
2019-01-15T15:41:42.542+0800 I CONTROL [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'
2019-01-15T15:41:42.582+0800 I CONTROL [initandlisten] MongoDB starting : pid=1017 port=27017 dbpath=/data/db 64-bit host=jie-xie
2019-01-15T15:41:42.582+0800 I CONTROL [initandlisten] db version v4.0.4
2019-01-15T15:41:42.582+0800 I CONTROL [initandlisten] git version: f288a3bdf201007f3693c58e140056adf8b04839
2019-01-15T15:41:42.582+0800 I CONTROL [initandlisten] allocator: system
2019-01-15T15:41:42.582+0800 I CONTROL [initandlisten] modules: none
2019-01-15T15:41:42.582+0800 I CONTROL [initandlisten] build environment:
2019-01-15T15:41:42.583+0800 I CONTROL [initandlisten] distarch: x86_64
2019-01-15T15:41:42.583+0800 I CONTROL [initandlisten] target_arch: x86_64
2019-01-15T15:41:42.583+0800 I CONTROL [initandlisten] options: {}
2019-01-15T15:41:42.586+0800 I STORAGE [initandlisten] Detected data files in /data/db created by the 'wiredTiger' storage engine, so setting the active storage engine to 'wiredTiger'.
2019-01-15T15:41:42.586+0800 I STORAGE [initandlisten] wiredtiger_open config: create,cache_size=3584M,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=10000),statistics_log=(wait=0),verbose=(recovery_progress),
2019-01-15T15:41:43.492+0800 I STORAGE [initandlisten] WiredTiger message [1547538103:388638][1017:0x1117295c0], txn-recover: Main recovery loop: starting at 12/27520 to 13/256
2019-01-15T15:41:43.492+0800 I STORAGE [initandlisten] WiredTiger message [1547538103:492147][1017:0x1117295c0], txn-recover: Recovering log 12 through 13
2019-01-15T15:41:43.564+0800 I STORAGE [initandlisten] WiredTiger message [1547538103:564763][1017:0x1117295c0], txn-recover: Recovering log 13 through 13
2019-01-15T15:41:43.629+0800 I STORAGE [initandlisten] WiredTiger message [1547538103:629822][1017:0x1117295c0], txn-recover: Set global recovery timestamp: 0
2019-01-15T15:41:43.775+0800 I RECOVERY [initandlisten] WiredTiger recoveryTimestamp: Ts: Timestamp(0, 0)
2019-01-15T15:41:43.986+0800 I CONTROL [initandlisten]
2019-01-15T15:41:43.986+0800 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2019-01-15T15:41:43.986+0800 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.
2019-01-15T15:41:43.986+0800 I CONTROL [initandlisten] ** WARNING: You are running this process as the root user, which is not recommended.
2019-01-15T15:41:43.986+0800 I CONTROL [initandlisten]
2019-01-15T15:41:43.986+0800 I CONTROL [initandlisten] ** WARNING: This server is bound to localhost.
2019-01-15T15:41:43.986+0800 I CONTROL [initandlisten] ** Remote systems will be unable to connect to this server.
2019-01-15T15:41:43.986+0800 I CONTROL [initandlisten] ** Start the server with --bind_ip <address> to specify which IP
2019-01-15T15:41:43.986+0800 I CONTROL [initandlisten] ** addresses it should serve responses from, or with --bind_ip_all to
2019-01-15T15:41:43.986+0800 I CONTROL [initandlisten] ** bind to all interfaces. If this behavior is desired, start the
2019-01-15T15:41:43.986+0800 I CONTROL [initandlisten] ** server with --bind_ip 127.0.0.1 to disable this warning.
2019-01-15T15:41:43.986+0800 I CONTROL [initandlisten]
2019-01-15T15:41:43.986+0800 I CONTROL [initandlisten] ** WARNING: soft rlimits too low. Number of files is 256, should be at least 1000
2019-01-15T15:41:44.222+0800 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory '/data/db/diagnostic.data'
2019-01-15T15:41:44.250+0800 I NETWORK [initandlisten] waiting for connections on port 27017
```

接下来点击左上角的第 1 个图标，选择 Create 来创建一个新的连接，如下：



连接上以后，本地 MongoDB 里面所有的数据库就会显示出来，如下：

Robo 3T - 1.2

New Connection (6)

System config stlinfo test xiejie

Welcome

Release the full power of the mongo shell with Studio 3T

Write MongoDB queries faster with IntelliShell, Studio 3T's built-in mongo shell.

IntelliShell autocompletes:

- JavaScript standard library functions
- shell-specific types and methods
- operators
- collection names
- field names
- shell helper commands... and more

In addition to robust autocompletion, enjoy other features like code reformatting.

Blog Posts

- Robo 3T 1.2 is released Mon, 19 Feb 2018
- Robomongo is now RC MongoDB 3.4 support Wed, 14 Jun 2017
- Robomongo 1.0 — Official Thu, 20 Apr 2017
- Robomongo has been Tue, 14 Mar 2017
- Changing the name of Mon, 13 Feb 2017
- Robomongo 1.0 RC1 b Replica Set Clusters Thu, 02 Feb 2017
- Robomongo 0.9.0 Final Thu, 06 Oct 2016
- Robomongo RC10 brings Fri, 19 Aug 2016
- Robomongo RC9 Wed, 01 Jun 2016
- Robomongo RC8

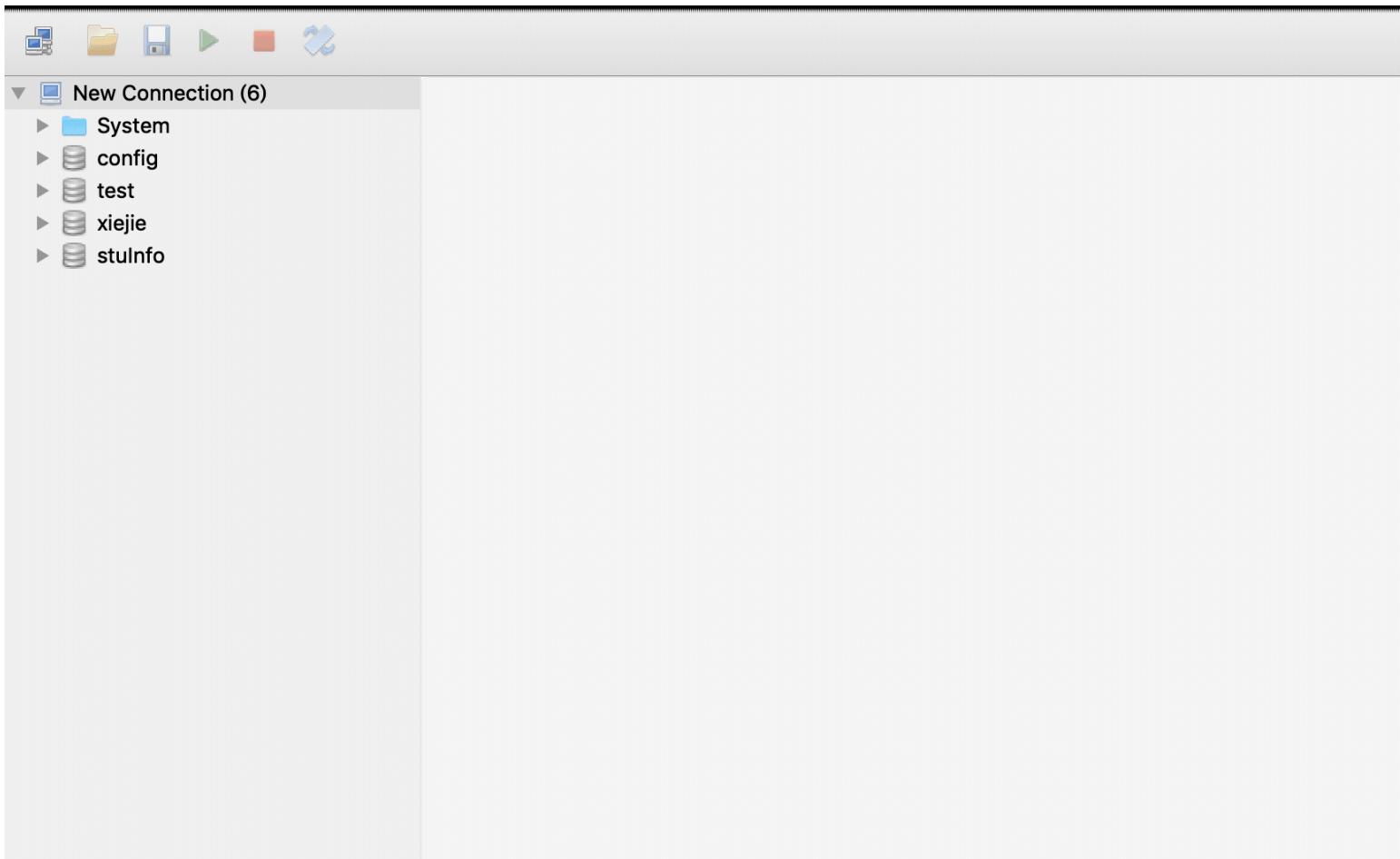
Logs

展开数据库，里面会有该数据库中的集合，选择集合以后右边会出现具体的信息，如下：

The screenshot shows the Robo 3T interface. On the left, the sidebar displays a tree view of databases and collections. Under the 'xiejie' database, there is a single collection named 'stu'. The main pane shows the results of the query `db.getCollection('stu').find({})`. The results are displayed in a table with the following data:

_id	name	age	gender	score
1	希之	1.5	男	15.0
2	李四	19	女	93
3	谢杰	18	男	100
4	亚静	20	女	98
5	王五	21	男	77
6	张三	25	男	92
7	韩梅梅	19	女	98

可以通过鼠标右键来创建数据库或者集合，通过 insert document 来插入新的文档，如下：



8. 数据的导入与备份

本文主要包含以下知识点：

- 数据的导入
- 数据的导出
- 数据库的备份
- 数据库的恢复

数据的导出

MongoDB 中的 mongoexport 工具可以把一个 collection 导出成 JSON 格式或 CSV 格式的文件。可以通过参数指定导出的数据项，也可以根据指定的条件导出数据。

具体的语法如下：

```
mongoexport -d dbname -c collectionname -o file --type json/csv -f field
```

参数说明如下：

```
-d : 数据库名  
-c : collection 名  
-o : 输出的文件名  
--type : 输出的格式, 默认为 json  
-f : 输出的字段
```

下面来看一个具体的示例：

我要导出的数据如下图所示，数据库名为 stuInfo，集合名为 students

_id	id	name	age	birth	gender	email	score	mobile	addr
1	ObjectId(...)	谢杰	18	1990-03...	男	7450078...	100	1598227...	龙泉驿区...
2	ObjectId(...)	宋亚静	20	1988-07...	女	5645766...	99	1820012...	龙泉驿区...
3	ObjectId(...)	谢希之	1	2017-03...	男	1252969...	90	1598227...	龙泉驿区...
4	ObjectId(...)	秦毅	20	2019-01...	男	2@2.com	90	14112341...	龙泉驿区...
5	ObjectId(...)	袁进	35	2019-01...	男	3@3.com	12	1311234...	成都朗沃...
6	ObjectId(...)	谢思奇	21	2019-01...	女	4@4.com	99	15112341...	曹家巷...
7	ObjectId(...)	张宏磊	21	2019-01...	男	5@5.com	100	18112341...	大丰镇...
8	ObjectId(...)								

进行导出：

```
Last login: Mon May 27 23:22:20 on ttys003
[Jie-Xie:~ Jie$ mongoexport -d stuInfo -c students -o /Users/Jie/Desktop/stu.json --type json -f "_id,id,name,age,birth,gender,email,score,mobile,addr"
2019-05-27T23:43:45.236+0800      connected to: localhost
2019-05-27T23:43:45.240+0800      exported 7 records
Jie-Xie:~ Jie$
```

注：需要配置了环境变量才能够直接使用 `mongoexport` 命令，否则需要切换到 `mongodb` 所在的 `bin` 目录下。

导出成功后，在我指定的路径下（这里我指定的路径为桌面）会生成了一个名为 `stu.json` 的文件。

数据的导入

数据导入所使用的命令为 `mongoimport`。具体的语法如下：

```
mongoimport -d dbname -c collectionname --file filename --headerline --type json/csv -f field
```

参数说明如下：

- d : 数据库名
- c : collection名
- type : 导入的格式默认json
- f : 导入的字段名
- headerline : 如果导入的格式是csv，则可以使用第一行的标题作为导入的字段
- file : 要导入的文件

来看一个具体的例子，我来将刚才导出的 `stu.json` 文件重新导入到数据库，创建一个新的数据库为

stuInfo2, 集合名为 students2

```
[Jie-Xie:~ Jie$ sudo mongoimport -d stuInfo2 -c students2 --file /Users/Jie/Desktop/stu.json --type json  
Password:  
2019-05-28T00:02:19.227+0800      connected to: localhost  
2019-05-28T00:02:19.332+0800      imported 7 documents  
Jie-Xie:~ Jie$
```

效果：可以看到，我们的 stu.json 数据已经成功导入到了 stuInfo2 数据库中，并且 stuInfo2 数据库原本是不存在的，这里已经自动为我们创建了此数据库。

_id	id	name	age	birth	gender	email	score	mobile	addr
1	ObjectId(...)	宋亚静	20	1988-07...	女	5645766...	99	1820012...	龙泉驿区...
2	ObjectId(...)	秦毅	20	2019-01...	男	2@2.com	90	14112341...	龙泉驿区...
3	ObjectId(...)	谢希之	1	2017-03...	男	1252969...	90	1598227...	龙泉驿区...
4	ObjectId(...)	谢杰	18	1990-03...	男	7450078...	100	1598227...	龙泉驿区...
5	ObjectId(...)	袁进	35	2019-01...	男	3@3.com	12	1311234...	成都朗沃
6	ObjectId(...)	谢思奇	21	2019-01...	女	4@4.com	99	15112341...	曹家巷
7	ObjectId(...)	张宏磊	21	2019-01...	男	5@5.com	100	18112341...	大丰镇

数据库的备份

数据库的备份也是经常会涉及到的操作。数据库备份的语法如下：

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

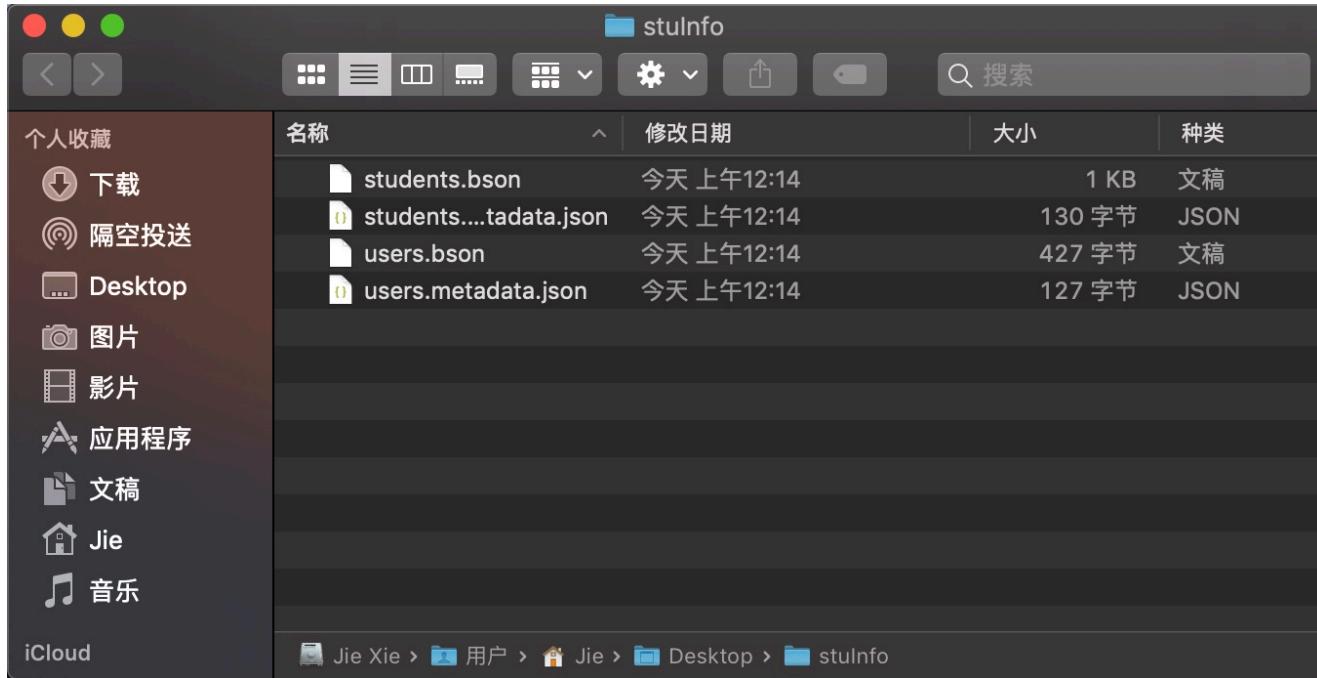
参数如下：

```
-h: MongoDB 所在的服务器地址, 例如: 127.0.0.1, 当然也可以指定端口号: 127.0.0.1:27017  
-d: 需要备份的数据库实例, 例如: test  
-o: 备份的数据存放位置, 例如: /home/mongodump/, 当然该目录需要提前建立, 这个目录里面存放该数据库实例的备份数据。
```

下面我们来看一个具体的示例：

```
[Jie-Xie:~ Jie$ mongodump -h localhost:27017 -d stuInfo -o /Users/Jie/Desktop/
2019-05-28T00:14:50.165+0800      writing stuInfo.students to
2019-05-28T00:14:50.165+0800      writing stuInfo.users to
2019-05-28T00:14:50.167+0800    done dumping stuInfo.students (7 documents)
2019-05-28T00:14:50.168+0800    done dumping stuInfo.users (7 documents)
Jie-Xie:~ Jie$ ]
```

在上面的示例中，我将 stuInfo 数据库备份至我的桌面，执行之后，我的桌面上就会生成一个数据库备份的目录，如下：



数据库的恢复

数据库恢复是指将备份的数据库重新还原至 MongoDB 数据库中。语法如下：

```
mongorestore -h dbhost -d dbname --dir dbdirectory
```

参数说明如下：

- h: MongoDB 所在服务器地址
- d: 需要恢复的数据库实例，例如：test，当然这个名称也可以和备份时候的不一样，比如test2
- dir: 备份数据所在位置，例如：/home/mongodump/itcast/
- drop: 恢复的时候，先删除当前数据，然后恢复备份的数据。就是说，恢复后，备份后添加修改的数据都会被删除，慎用！

```

Jie-Xie:~ Jie$ mongorestore -h localhost:27017 -d stuInfo3 --dir /Users/Jie/Desktop/stuInfo
2019-05-28T00:23:15.107+0800      the --db and --collection args should only be used when restoring from a BSON file. Other uses
are deprecated and will not exist in the future; use --nsInclude instead
2019-05-28T00:23:15.107+0800      building a list of collections to restore from /Users/Jie/Desktop/stuInfo dir
2019-05-28T00:23:15.109+0800      reading metadata for stuInfo3.students from /Users/Jie/Desktop/stuInfo/students.metadata.json
2019-05-28T00:23:15.109+0800      reading metadata for stuInfo3.users from /Users/Jie/Desktop/stuInfo/users.metadata.json
2019-05-28T00:23:15.248+0800      restoring stuInfo3.users from /Users/Jie/Desktop/stuInfo/users.bson
2019-05-28T00:23:15.358+0800      restoring stuInfo3.students from /Users/Jie/Desktop/stuInfo/students.bson
2019-05-28T00:23:15.361+0800      no indexes to restore
2019-05-28T00:23:15.361+0800      finished restoring stuInfo3.users (7 documents)
2019-05-28T00:23:15.361+0800      no indexes to restore
2019-05-28T00:23:15.361+0800      finished restoring stuInfo3.students (7 documents)
2019-05-28T00:23:15.361+0800      done
Jie-Xie:~ Jie$ 

```

效果：数据库备份已经成功恢复至 MongoDB 中，并且命名为 stuInfo3

The screenshot shows the Robo 3T MongoDB interface. On the left, the sidebar displays a tree view of databases and collections. A red box highlights the 'stuInfo3' database entry, which contains two collections: 'students' and 'users'. The 'students' collection is currently selected, and its data is shown in the main pane as a table with columns: _id, id, name, age, birth, gender, email, score, mobile, and addr. There are 7 rows of data, each representing a student record.

	_id	id	name	age	birth	gender	email	score	mobile	addr
1	ObjectId(...)	1	谢杰	18	1990-03...	男	7450078...	100	1598227...	龙泉驿区...
2	ObjectId(...)	2	宋亚静	20	1988-07...	女	5645766...	99	1820012...	龙泉驿区...
3	ObjectId(...)	3	谢希之	1	2017-03...	男	1252969...	90	1598227...	龙泉驿区...
4	ObjectId(...)	5	秦毅	20	2019-01...	男	2@2.com	90	14112341...	龙泉驿区...
5	ObjectId(...)	6	袁进	35	2019-01...	男	3@3.com	12	1311234...	成都朗沃...
6	ObjectId(...)	7	谢思奇	21	2019-01...	女	4@4.com	99	15112341...	曹家巷...
7	ObjectId(...)	8	张宏磊	21	2019-01...	男	5@5.com	100	18112341...	大丰镇...

9. Mongoose

mongoose 是 node.js 提供的一个便捷操作 mongodb 的库。除了 mongoose 以外还有 mongoskin、mongodb（mongodb 官方出品）等。本文将对 mongoose 进行简单介绍。

本文主要包含以下内容：

- mongoose 安装
- 连接 mongodb
- Schema
- model
- 常用命令

mongoose 安装

使用 mongoose 首先需要保证已经安装好 node.js 和 mongodb。然后在项目根目录中通过以下命令安装 mongoose：

```
npm install mongoose
```

连接 mongodb

安装好 mongoose 后，就可以在项目中通过 mongoose 提供的方法将项目与 mongodb 连接起来。代码如下：

```
const mongoose = require('mongoose');
const dbURI = 'mongodb://localhost/test'; // 其中 test 为连接的数据库名称
mongoose.connect(dbURI, {useNewUrlParser: true});
mongoose.connection.on('connected', function() {
  console.log('Mongoose connected to ' + dbURI);
});
```

然后启动 mongodb，运行以上代码。命令行出现以下字符串表示连接成功。

```
'Mongoose connected to mongodb://localhost/test'
```

具体示例如下：

db.js

The screenshot shows a code editor interface. On the left, there's a sidebar with a tree view of files:

- 打开的编辑器
- APP
 - node_modules
 - JS db.js
 - package-lock.json
 - package.json

The main area shows a file named "db.js" with the following content:

```
1 const mongoose = require('mongoose');
2 const dbURI = 'mongodb://localhost/schooldb'; // 其中 schooldb 为连接的数据库名称
3 mongoose.connect(dbURI, {useNewUrlParser: true});
4 mongoose.connection.on('connected', function() {
5     console.log('Mongoose connected to ' + dbURI);
6});
```

运行结果：

```
[Jie-Xie:app Jie$ node db.js
Mongoose connected to mongodb://localhost/schooldb
```

Schema

Schema 是 mongoose 提供的一个方法，用来定义 mongodb 中对应的数据集合的结构。例如：

```
const usersSchema = new mongoose.Schema({
  username: String,    // 用户名
  password: String    // 密码
})
// 或
const usersSchema = new mongoose.Schema({
  username: { type: String },
  password: { type: String }
})
```

Schema Types 内置类型如下：

- String
- Number
- Boolean | Bool
- Array
- Buffer
- Date
- ObjectId | Oid
- Mixed

只有 Schema 中所对应数据类型的数据会被查询出来。例如 Schema 中定义的 age 字段为

Number, 但是在数据库中存储的类型为 *String*, 则查询结果中不会有 *age* 字段的数据。

model

model 是由 Schema 生成的数据模型，可以对 mongodb 中的数据进行操作。

结构定好之后创建数据模型，mongoose 对象还提供了一个方法 `model()` 用来创建数据模型：

```
mongoose.model("usersModel", userSchema, "users");
```

- "usersModel": 数据模型名称
- userSchema: 数据结构
- "users": 数据模型对应的数据集合名称

具体的示例如下：

studentModel.js

```
资源管理器 JS studentModel.js ×
1  require('./db');
2
3  const mongoose = require('mongoose');
4
5  // 定义 Schema
6  const studentsSchema = new mongoose.Schema({
7      name: String,
8      gender: String,
9      age: String,
10     address: String,
11 })
12
13 // 定义 model
14 // studentsModel 为数据模型的名字
15 // studentsSchema 为上面所定义的 Schema
16 // students 为数据库中对应的集合名称
17 mongoose.model("studentsModel", studentsSchema, "students");
18
19 // 导出定义好的 model
20 module.exports.studentsModel = mongoose.model("studentsModel");
21
```

常用命令

`find()` 查询

查询所有

```
mongoose.model("usersModel").find((err, data) => {})
```

具体示例如下：

studentDao.js

The screenshot shows a code editor interface. On the left, there is a sidebar titled "资源管理器" (Resource Manager) showing the project structure:

- 打开的编辑器 (Open Editors): JS studentDao.js
- APP:
 - node_modules
 - JS db.js
 - npm package-lock.json
 - npm package.json
 - JS studentDao.js (highlighted)
 - JS studentModel.js

The main area is titled "JS studentDao.js" and contains the following code:

```
1 const { studentsModel } = require('../studentModel'); // 导入上面所定义好的数据模型
2
3 studentsModel.find((err, data) => {
4   console.log(data)
5});
```

运行结果部分截图：

```
[Jie-Xie:app Jie$ node studentDao
Mongoose connected to mongodb://localhost/schooldb
[ { _id: '5cb5a7255f5d040fcfb5dca6',
  name: '阎磊',
  gender: '男',
  age: '21',
  address: '安徽省 池州市',
  phone: '13798067741',
  class: 5c2c834e43a8342ca8ba7165,
  course: 5c2c4ef92f420645040ac723 },
{ _id: '5cb5a7255f5d040fcfb5dca5',
  name: '宋军',
  gender: '女',
  age: '21',
  address: '青海省 海西蒙古族藏族自治州',
  phone: '14245901262',
  class: 5c2c834e43a8342ca8ba7165,
  course: 5c2c4ef92f420645040ac723 },
```

分页查询

在进行查询操作时，一个常见的操作就是分页查询。其核心思想就是根据总数据条数和每页的数据条数计算出总的页数。

在 Mongoose 中有一个 countDocuments 方法可以快速获取到数据的总条数。

下面是一个根据当前页码和每页显示的条数来进行分页查询的代码片段：

```
module.exports.findSomeStuDao = async function(page){  
    let pageObj = {  
        currentPage: Number(page.currentPage), // 当前页码  
        eachPage: Number(page.eachPage) // 每页显示的条数  
    }  
    pageObj.count = await stuModel.countDocuments(); // 数据总条数  
    pageObj.totalPage = Math.ceil(pageObj.count / pageObj.eachPage); // 总页数  
    pageObj.data = await stuModel // 查询到的数据结果  
        .find()  
        .skip((pageObj.currentPage - 1) * pageObj.eachPage) // 设置跳过的数据条数  
        .limit(pageObj.eachPage); // 设置查询条数  
    return pageObj;  
}
```

条件查询

```
mongoose.model("usersModel").find({username:'zhangsan', password: '123'}, (err, data) =>  
{});
```

具体示例如下：

studentDao.js

The screenshot shows a code editor interface with a sidebar on the left containing a file tree. The tree includes '打开的编辑器' (Open Editors) with 'studentModel.js' and 'studentDao.js' (marked with a red 'X'), and 'APP' with 'node_modules', 'db.js', 'package-lock.json', 'package.json', 'studentDao.js' (highlighted in blue), and 'studentModel.js'. The main panel shows two tabs: 'JS studentModel.js' and 'JS studentDao.js' (marked with a red 'X'). The 'studentDao.js' tab contains the following code:

```
1 const { studentsModel } = require('../studentModel'); // 导入上面所定义好的数据模型  
2  
3 studentsModel.find({name:'乔敏'},(err, data) => {  
4     console.log(data)  
5});
```

运行结果：

```
[Jie-Xie:app Jie$ node studentDao
Mongoose connected to mongodb://localhost/schooldb
[ { _id: '5cb5a7255f5d040fcfb5dcb7',
  name: '乔敏',
  gender: '女',
  age: '20',
  address: '安徽省 蚌埠市',
  phone: '16353195759',
  class: 5c2c834e43a8342ca8ba7165,
  course: 5c2c4ef92f420645040ac723 } ]
```

模糊查询

- `$or` : 表示在数组中的条件满足其中一个即可。

```
mongoose.model("usersModel").find({
  $or: [
    {name: 'zhang'},
    {age: '1'},
    {gender: '男'}
  ]
})
```

- `$regex/$options` : 表示一个正则表达式，匹配后面的值，同时加入 `$options:'$i'` 表示忽略大小写。

```
mongoose.model("usersModel").find({
  name: { $regex: 'zhang', $options: '$i' }
})
```

create() 新增

```
mongoose.model("usersModel").create({username:'zhangsan', password: '123'}, (err, data) =>
{})
```

具体示例如下：

studentDao.js

资源管理器

打开的编辑器

- JS** studentDao.js
- APP
- node_modules
- JS** db.js
- JSON** package-lock.json
- JSON** package.json
- JS** studentDao.js
- JS** studentModel.js

JS studentDao.js

```

1 const { studentsModel } = require('../studentModel'); // 导入上面所定义好的数据模型
2
3 studentsModel.create({
4   name: 'zhangsan',
5   age: '1230',
6   gender: 'male',
7   address: 'haijiaoshi',
8 }, (err, data) => {
9   if(err) throw err;
10   console.log(data);
11 });
12

```

运行效果：

```
[Jie-Xie:app Jie$ node studentDao
Mongoose connected to mongodb://localhost/schooldb
{ _id: 5cf2a1fc99e05704c9005b02,
  name: 'zhangsan',
  age: '1230',
  gender: 'male',
  address: 'haijiaoshi',
  __v: 0 }
```

students 0.002 sec.									
	_id	name	class	course	gender	phone	age	address	__v
1	ObjectId("5cf2a1fc99e05704c9005b02")	谢杰	ObjectId("5cf2a1fc99e05704c9005b02")	ObjectId("5cf2a1fc99e05704c9005b02")	男	13112341...	18	12	
2	123	zhangsan			male	1230	haijiaoshi	0	
3	asd	zhangsan			male	1230	haijiaoshi	0	
4	ObjectId("5cf2a1fc99e05704c9005b02")	zhangsan			male	1230	haijiaoshi	0	

update() 修改

```
mongoose.model("usersModel").update({_id:1}, {username: 'zhangsan', password: '123'}, (err, data) => {});
```

具体示例如下：

studentDao.js

资源管理器

打开了的编辑器

- JS studentDao.js

APP

- node_modules
- JS db.js
- package-lock.json
- package.json
- JS studentDao.js
- JS studentModel.js

JS studentDao.js

```
1 const { studentsModel } = require('../studentModel'); // 导入上面所定义好的数据模型
2
3 studentsModel.update({name:"zhangsan"},{name:'张三', age: '17'}, (err, data) => {
4     if(err)throw err;
5     console.log(data);
6 });
7
```

运行效果：

```
Jie-Xie:app Jie$ node studentDao
Mongoose connected to mongodb://localhost/schooldb
(node:1253) DeprecationWarning: collection.update is deprecated. Use updateOne, updateMany, or bulkWrite instead.
{ n: 1, nModified: 1, ok: 1 }
```

students 0.002 sec.								
	_id	name	class	course	gender	phone	age	address
1	ObjectId(...)	谢杰	ObjectId(...)	ObjectId(...)	男	13112341...	18	12
2	123	张三			male		17	haijiaoshi 0
3	asd	zhangsan			male		1230	haijiaoshi 0
4	ObjectId(...)	zhangsan			male		1230	haijiaoshi 0

注：如果要修改多条，可以使用 `updateMany` 方法。

`delete()` 删除

```
mongoose.model("usersModel").deleteOne({_id:1}, (err, data) => {});
```

具体示例如下：

资源管理器

JS studentDao.js ×

```
1 const { studentsModel } = require('./studentModel'); // 导入上面所定义好的数据模型
2
3 studentsModel.deleteOne({name: '张三'}, (err, data) => {
4     if(err) throw err;
5     console.log(data);
6 });
7
```

▲ 打开的编辑器

✖ JS studentDao.js

▲ APP

▷ node_modules

JS db.js

npm package-lock.json

npm package.json

JS studentDao.js

JS studentModel.js

运行结果：数据成功被删除

```
[Jie-Xie:app Jie$ node studentDao
Mongoose connected to mongodb://localhost/schooldb
{ n: 1, ok: 1, deletedCount: 1 }
```

10. 连表查询

本文主要包含以下知识点：

- 为集合设置外键
- 多表联合查询

为集合设置外键

有些时候，集合与集合之间需要彼此拥有联系。

例如：一张学生表里面存储有学生的信息，一张班级表中存储有班级的信息。如果学生表中需要记录每个学生是哪个班级，这时候就需要学生表和班级表之间要产生联系。

如下图：

stu_id	stu_name	stu_age	stu_gender	stu_score	class_id
ObjectId("5c433804765f0b11dd6fd327")	xiejie	18	male	100	ObjectId("5cefeb6ad03233337da93c31")
ObjectId("5c43382a765f0b11dd6fd328")	yajing	20	female	99	ObjectId("5cefeb6ad03233337da93c31")

class_id	class_name
ObjectId("5cefeb6ad03233337da93c31")	Web 前端

在学生表中，有一个字段存储的是班级表中的班级 id，这样班级表就和学生表之间产生了联系。当我们要新添加学生时，需要选择学生所在的班级，如下：

学生管理系统

学生管理

[学生列表](#)

[新增学生](#)

班级管理

[班级列表](#)

[新增班级](#)

新增学生

学生姓名

学生年龄

学生性别

联系方式

学生班级

未选择任何文件

此时所选的班级信息应该来自于数据库中的班级表，代码片段如下：

```
// 从后台获取班级表的信息
getClasses() {
    $.ajax({
        url: '/classes/getClasses',
        type: 'get',
        dataType: 'json',
        success(msg) {
            let optionsHtml = msg.map(item => {
                return `<option value="${item._id}">${item.className}</option>`;
            }).join("");
            $("#className").html(optionsHtml);
        }
    })
}
```

多表联合查询

当我们查询学生数据时，班级字段不可能显示一个班级 id，而是需要显示班级名称。而班级名称存储在班级表里面的，所以这个时候就需要关联查询。

在 mongoose 中使用 population 来进行关联查询。

下面我们来看一个具体的示例，首先准备两张表，表结构与数据如下：

学生表：

_id		name	age	gender	phone	classId
1	ObjectId("5cefedd83aadfa343d80a96e")	biebie	18	男	15982274150	ObjectId("5cefedba3aadfa343d80a96c")
2	ObjectId("5cefefdf53aadfa343d80a96f")	123	123	男	123	ObjectId("5cefedba3aadfa343d80a96c")
3	ObjectId("5cfb1b4a5b127b03e4739eb4")	biebie	18	男	15982274150	ObjectId("5cefefdc03aadfa343d80a96d")

班级表：

_id		className	_v
1	ObjectId("5cefedba3aadfa343d80a96c")	f59	0
2	ObjectId("5cefefdc03aadfa343d80a96d")	f60	0
3	ObjectId("5cfb1b5b5b127b03e4739eb5")	f61	0

接下来配置 studentModel，代码如下：

```
// studentModel.js

require('./db.js');

const mongoose = require('mongoose');

const studentsSchema = new mongoose.Schema({
  name: String,
  age: String,
  gender: String,
  phone: String,
  classId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "classesModel"
  }
})

mongoose.model("studentsModel", studentsSchema, "students");

module.exports.studentsModel = mongoose.model("studentsModel");
```

在 studentModel.js 中，classId.ref 表示关联 classesModel，代码如下：

```
// classesModel.js

const mongoose = require('mongoose');

const classesSchema = new mongoose.Schema({
```

```
    className: String,
  })
}

mongoose.model("classesModel", classesSchema, "classes");

module.exports.classesModel = mongoose.model("classesModel");
```

最后在 studentDao.js 中使用 population 进行关联查询，代码如下：

```
// studentDao.js

const { studentsModel } = require('./studentModel');
const { classesModel } = require('./classesModel');

studentsModel.find({}, 'name age gender phone').
  populate('classId').
  exec(function (err, data) {
  if(err) throw err;
  console.log(data);
});
```

运行结果如下：

```
Jie-Xie:app Jie$ node studentDao
Mongoose connected to mongodb://localhost/f52
[ { _id: 5cefedd83aadfa343d80a96e,
  name: 'xiejie',
  age: '18',
  gender: '男',
  phone: '15982274150',
  classId: { _id: 5cefedba3aadfa343d80a96c, className: 'f59', __v: 0 } },
{ _id: 5cefef53aadfa343d80a96f,
  name: '123',
  age: '123',
  gender: '男',
  phone: '123',
  classId: { _id: 5cefedba3aadfa343d80a96c, className: 'f59', __v: 0 } },
{ _id: 5cfb1b4a5b127b03e4739eb4,
  name: 'xiejie',
  age: '18',
  gender: '男',
  phone: '15982274150',
  classId: { _id: 5cefecd03aadfa343d80a96d, className: 'f60', __v: 0 } },
{ _id: 5cffc0212bd70114e99bedf0,
  name: 'xiejie',
  age: '18',
  gender: '男',
  phone: '15982274150',
  classId: { _id: 5cefedba3aadfa343d80a96c, className: 'f59', __v: 0 } } ]
```

可以看到，这里我们就成功的进行了两张表的关联查询。