第3章

文档数据库与MongoDB

1989年，Lotus提出文档数据库(Document Database)的概念。文档数据库区别于传统的关系数据库，它是用于管理文档的。在传统的关系数据库中，信息被分割成离散的数据段，而在文档数据库中，文档是处理信息的基本单位。一个文档可以很长、很复杂，甚至可以无结构。

文档数据库也可以说是键值对数据库[2](NoSQL数据库的另一个概念)的一个子类。在键值对数据库中，数据对数据库来说是不透明的。但面向文档的数据库系统依赖于文档的内部结构，它会获取元数据用于数据库的深层优化。

MongoDB是一种面向集合且模式自由的文档数据库。它在简化开发的同时为Web应用提供可扩展的高性能数据存储解决方案。

本章通过对文档数据库基本概念的阐述和对MongoDB安装配置和基本操作的介绍展现了文档数据库的适用范围和其使用方法。

3.1 MongoDB简介

MongoDB[3]数据库是基于分布式文件存储的开源数据库系统之一，用C++编程语言编写。它是一种面向集合且模式自由的文档数据库。

面向集合是指数据备份组存在于数据集(集合)中。MongoDB的集合类似于关系数据库的表，但要操作一个集合并不需要创建它。在存入数据时，如果集合不存在，则自动创建。

文档数据库的特点是将数据存储为文档。MongoDB数据库的一个文档类似于关系数据库的一条记录。它支持的数据结构非常松散，是类似JSON[4]格式的BSON(Binary Serialized Document Format，Binary JSON)格式，因此可以存储比较复杂的数据类型。

模式自由是对存储在 MongoDB 数据库中的数据而言的。MongoDB 数据库的文档存储的数据结构是键值对(Key-Value),键(Key)是字符串,值(Value)可以是数据类型集合中的任意类型,包括数组和文档。

MongoDB 支持的查询语言也非常强大,其语法有点类似于面向对象的查询语言,几乎可以实现类似关系数据库单表查询的绝大部分功能,而且还支持对数据建立索引。

MongoDB 数据库数据查询和存储与关系数据库的相似性使它成为非关系数据库当中功能最丰富,最像关系数据库的数据库。

3.2　基本概念

3.2.1　文档数据模型

MongoDB 将数据以 BSON 格式存储于文档中。这是考虑数据的最自然方法,比传统的行/列模型更具表现力和功能。传统的关系数据库需要对表结构进行预先定义和严格的要求,而这样的严格要求,导致了处理数据的过程更加烦琐,甚至降低了执行效率。在数据量达到一定规模的情况下,传统关系数据库反应迟钝,想解决这个问题就需要反其道而行之,尽可能去掉传统关系数据库的各种规范约束,甚至事先无须定义数据存储结构,这一点在 MongoDB 的文档数据模型中很好地体现了出来。MongoDB 的文档数据模型如图 3-1 所示。

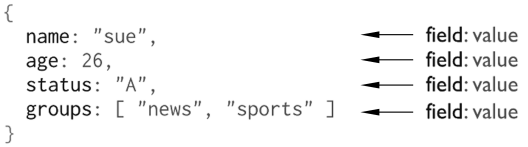

图 3-1　MongoDB 的文档数据模型

从图 3-1 可以看出,一个大括号中包含若干个键值对,大括号中的内容就被称为一条文档,文档中存储的值(Value)可以是字符串、数值、数组、文档等类型。

3.2.2　文档存储结构

MongoDB 文档数据库的存储结构分为四个层次,从小到大依次是:键值对、文档(Document)、集合(Collection)、数据库(Database)。MongoDB 的存储逻辑结构为文档,文档中采用键值对结构,文档中的_id 为主键,默认创建主键索引。表 3-1 通过对比 MongoDB 数据库的存储与传统关系数据库存储描述了其存储结构。

表 3-1　MongoDB 存储与 MySQL 存储的对比

MySQL 术语/概念	MongoDB 术语/概念
数据库(Database)	数据库(Database)
表(Table)	集合(Collection)

续表

MySQL 术语/概念	MongoDB 术语/概念
数据记录行(Row)	数据记录文档(Document)
数据字段(Column)	数据域(Field)
索引(Index)	索引(Index)
表连接(Table Joins)	不支持
主键(Primary Key)	主键,MongoDB 自动将_id 字段设置为主键(Primary Key)

通过表 3-1 可以看出,MongoDB 的文档、集合、数据库对应于关系数据库中的行数据、表、数据库。

键值对是文档数据库存储结构的基本单位,具体包含数据和类型。键值对的数据包含键和值,键的格式一般为字符串,除了少数例外情况,键可以使用任意 UTF-8 字符。值的格式可以包含字符串、数值、数组、文档等类型。按照键值对的复杂程度,可以将键值对分为基本键值对和嵌套键值对。键值对中的键为字符串,值为基本类型的键值对就称为基本键值对。嵌套键值对类型的键对应的值为一个文档,文档中又包含相关的键值对。键起唯一索引的作用,确保一个键值结构里数据记录的唯一性,同时也具有信息记录的作用。值是键所对应的数据,其内容通过键来获取,可存储任何类型的数据,甚至可以为空。它们之间的关系是一一对应的。

文档是 MongoDB 的核心概念,是数据的基本单元。文档是一组有序的键值对集合。文档的数据结构与 JSON 基本相同,所有存储在集合中的数据都是 BSON 格式。BSON 是一种类 JSON 的二进制存储格式,是 Binary JSON 的简称。MongoDB 的文档不需要设置相同的字段,并且相同的字段不需要相同的数据类型,这与关系数据库有很大的区别,也是 MongoDB 非常突出的特点。需要注意的是,文档中的键值对是有序的,且键不能重复。

文档中键有一些命名规范。

(1) 键不能含有空字符(\0),这个字符用来表示键的结尾。

(2) “.”和“$”有特别的意义,只有在特定环境下才能使用。

(3) 以下画线“_”开头的键是保留的,一般不以“_”开头。

(4) 命名区分类型和大小写。

MongoDB 将文档存储在集合中,一个集合是一些文档构成的对象。如果说 MongoDB 中的文档类似于关系数据库中的“行”,那么集合就如同“表”。集合存在于数据库中,没有固定的结构,这意味着用户对集合可以存入不同格式和类型的数据。但通常情况下存入集合的数据都会有一定的关联性,即一个集合中的文档应该具有相关性。集合具有明确要求,合法的集合名不能是空字符串"",也不能含有空字符(\0,这个字符表示集合名的结尾),集合名不能以“system.”开头,这是为系统集合保留的前缀。用户创建的集合名字不能含有保留字符。有些驱动程序的确支持在集合名里面包含,这是因为某些系统生成的集合中包含该字符。除非访问这种系统创建的集合,否则不要在名字里出现“$”。

MongoDB中的数据具有灵活的架构，集合不强制要求文档结构。但数据建模的不同可能会影响程序性能和数据库容量。文档之间的关系是数据建模需要考虑的重要因素，而文档与文档之间的关系包括嵌入和引用两种。关系数据库的数据模型在设计时，将不同的键值对分到两个表中，在查询时进行关联，这就是引用的使用方式。如果在实际查询中，需要频繁地通过_id获得某个键值对信息，那么就需要频繁地通过关联引用来返回查询结果。在这种情况下，一个更合适的数据模型就是嵌入。将该键值对信息嵌入一个完整的键值对中，这样通过一次查询就可获得完整的键值对和所需信息。如果具有多个类似的所需信息，也可以将其嵌入键值对中，通过一次查询就可获得完整的 键值对和多个所需信息。但在某种情况下，引用比嵌入更有优势。嵌入式的关系可能导致信息重复，这时可采用引用的方式描述集合之间的关系。使用引用时，关系的增长速度决定了引用的存储位置。如果每个键值对对应的所需信息很少且增长有限，那么使用嵌入更有优势。但如果所需信息数量很多，则此数据模型将导致可变的、不断增长的数组。为了避免可变的、不断增长的数组应该选择使用引用。

在MongoDB中，数据库由集合组成。一个MongoDB实例可承载多个数据库，互相之间彼此独立，在开发过程中，通常将一个应用的所有数据存储到同一个数据库中，MongoDB将不同数据库存放在不同文件中。

3.2.3　数据类型

1. Object ID

MongoDB数据库中有一种特有的数据类ObjectID(文档自生成的_id)，是MongoDB生成的类似关系数据库表主键的唯一Key，具体由24个字节组成：1～8字节是时间戳；9～14字节的机器标识符，表示MongoDB实例所在机器的不同；15～18字节是进程ID，表示相同机器的不同MongoDB进程；19～24字节是计数器。例3-1的代码就是一个MongoDB自动生成的ObjectID。

【例3-1】 Object ID案例。

```
"_id" : ObjectId("5b151f8536409809ab2e6b26")
```

在这段Object ID中可以得到以下四种信息。

(1) “5b151f85”代指的是时间戳，也就是这条数据的产生时间。

(2) 接着的“364098”代指某台机器的机器码，用来表示存储这条数据时的机器编号。

(3) 再接下来的“09ab”代指进程ID，进程ID在多进程存储数据的时候非常有用。

(4) 之后的“2e6b26”代指计数器，这里要注意的是计数器的数字可能会出现重复，不是唯一的。以上四种标识符拼凑成世界上唯一的Object ID。

只要是支持MongoDB的语言都会有一个或多个方法对Object ID进行转换。要注意的是，Object ID类型是不能被JSON序列化的。

2. 常见数据类型

(1) 字符串(String)是存储数据常用的数据类型。在MongoDB中，UTF-8编码的字

符串才是合法的。

(2) 布尔值(Boolean)包含 true 和 false,用于存储布尔值。

(3) 整型数值(Integer)用于存储数值。根据用户所采用的服务器,可分为 32 位或 64 位。

(4) 双精度浮点值(Double)用于存储浮点值,因为没有单精度浮点类型(Float 类型),所以所有的浮点值都使用双精度浮点类型存储。

(5) 空数据类型(Null)用于创建空值。

(6) 数组或者列表(Arrays)将多个值存储到一个键。

3. 其他数据类型

(1) Min/Max Keys 用于将一个值与 BSON 元素的最低值和最高值相对比。

(2) 内嵌文档(Object),这种数据类似于 Python 中的字典。

(3) 符号(Symbol)基本上等同于字符串类型,不同的是它一般用于采用特殊符号类型的语言。

(4) 日期时间(Date)用 UNIX 时间格式来存储当前日期或时间。

(5) 二进制数据(Binary Data)用于存储二进制数据。

(6) 代码类型(Code)用于在文档中存储 JavaScript 代码。

(7) 正则表达式类型(Regular Expression)用于存储正则表达式。

3.3 MongoDB 的安装与配置

3.3.1 单机环境部署

在 MongoDB 官网下载安装包,地址为 http://www.mongodb.org/downloads,本书以在 Windows 系统上安装 MongoDB 4.4 社区版为例。

首先要在官网下载安装程序。完成安装程序的下载后就可以双击运行该 msi 文件安装程序。图 3-2 展示了 msi 安装程序界面。

在安装程序运行过程中,首先要选择设置类型,也就是向导自动完整的安装(Complete)还是用户自定义安装(Custom)。要注意的是,如果选择自动安装,系统将会自动将 MongoDB 和其工具安装到默认位置,所以建议读者选择自定义安装选项以选择合适的位置安装可执行文件。图 3-3 和图 3-4 分别展示了如何选择安装方式和安装位置。

选择了安装类型和安装位置后的步骤是服务配置。

从 MongoDB 4.0 开始,安装过程中还包括将 MongoDB 设置为 Windows 服务的选择和仅安装二进制文件的选择。

以下内容将为安装 MongoDB 并将其配置为 Windows 服务的安装程序流程。如果选择了 Install MongoDB as a Service 复选框也就是将 MongoDB 设置成为 Windows 服务,接下来就需要对以何种身份运行服务进行选择。如果选择网络用户身份,就是用 Windows 内置的 Windows 用户账户作为网络服务用户身份。如果选择以本地用户或域用户身份运行服务,则需要为本地用户账户设置账户域、账户名和密码。要注意的是,如果设置为本地用户而不是域用户,则账户域为“.”。

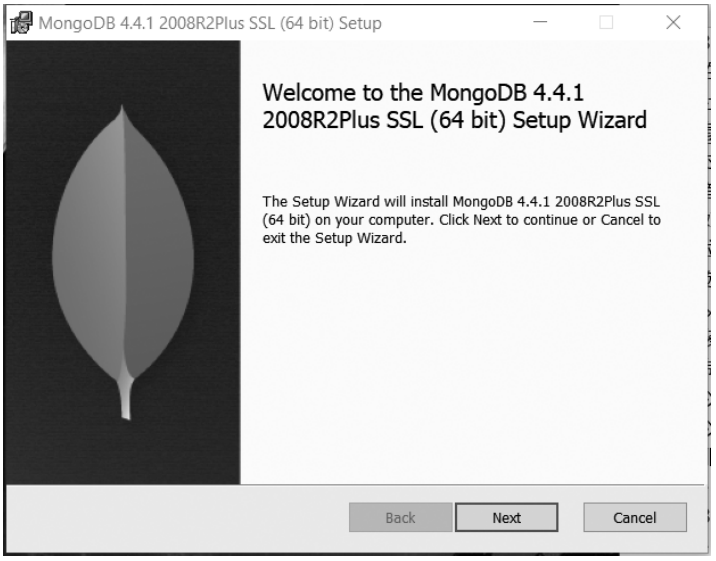

图 3-2　MongoDB 安装程序

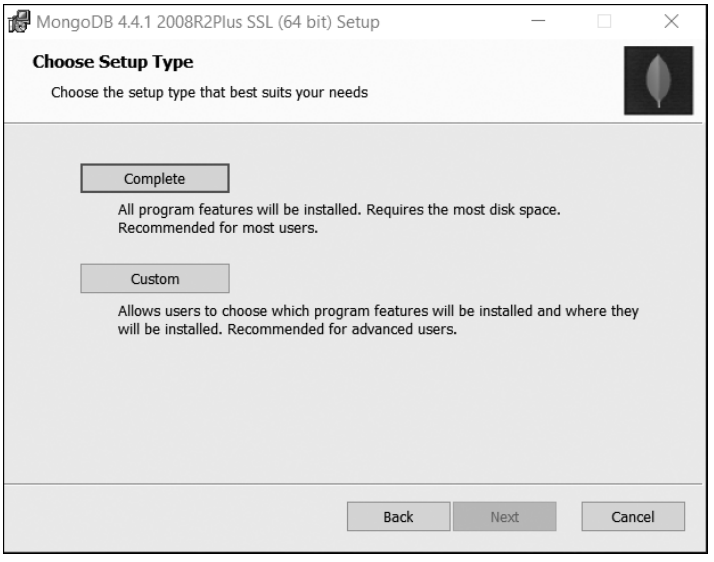

图 3-3　安装方式选择

接下来的三个选项分别是服务名称(Service Name)、数据目录(Data Directory)和日志目录(Log Directory)。图 3-5 为服务配置界面。

通过图 3-5 可以看出,安装 MongoDB 过程中,其服务名称默认为 MongoDB。如果已经拥有使用指定名称的服务,则必须选择另一个名称。如果数据目录不存在,安装程序将创建目录并将目录访问权限设置为服务用户。如果日志目录不存在,安装程序将创建目录并将目录访问权限设置为服务用户。

接下来的步骤可以选择是否安装 MongoDB 的可视化工具——MongoDB Compass。

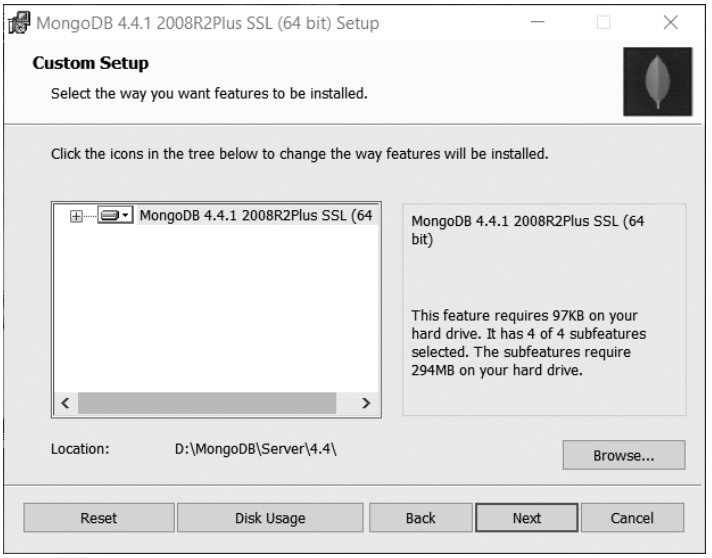

图 3-4　安装位置选择

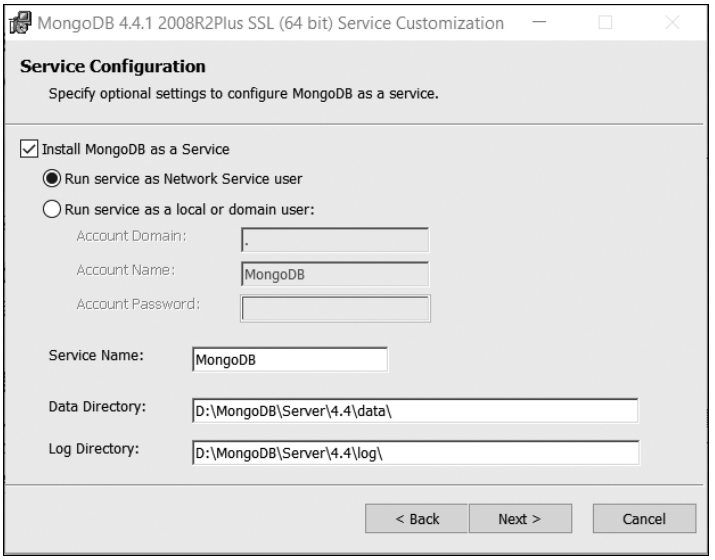

图 3-5　服务配置界面

MongoDB 数据库具有灵活丰富的文档结构，这会帮助开发人员构建更丰富的数据结构，加快开发速度。然而，这样的灵活性也使其中的数据结构变得难以理解。MongoDB Compass 的引入帮助 MongoDB 数据库解决了这个问题，方便开发人员对数据结构的查询。

安装全部完成后在安装目录的 bin 文件夹（笔者的安装位置是 D:\MongoDB\Server\4.4\bin，如果读者选择了自动安装则安装目录应该是 C:\Program Files\MongoDB\Server\4.4\bin）下打开命令行，输入命令“mongod --dbpath F:\data\db”启动 MongoDB 服务，启动成功后打开浏览器，输入网址“http://localhost:27017”（27017 是 MongoDB

数据库服务的端口号)查看,若显示“It looks like you are trying to access MongoDB over HTTP on the native driver port.”,则表示连接成功。如果不成功,可以查看端口是否被占用。连接成功后,可以如图 3-6 所示在任务管理器中查看当前 MongoDB 进程,以此确定 MongoDB 正常启动和运行状态。

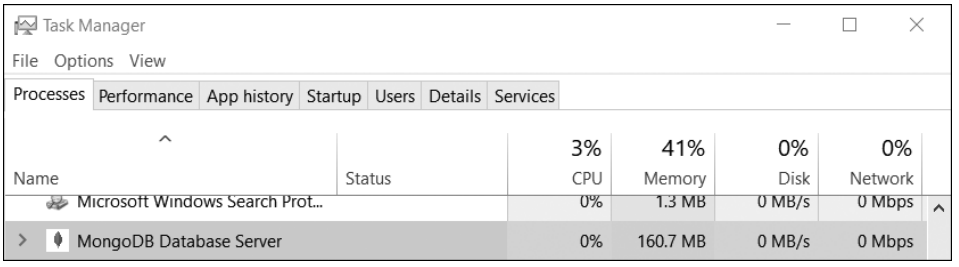

图 3-6　任务管理器中的 MongoDB 进程

3.3.2　MongoDB 的配置文件

将 MongoDB 设置为 Windows 服务后,需要如图 3-7 所示配置 MongoDB 的环境变量。

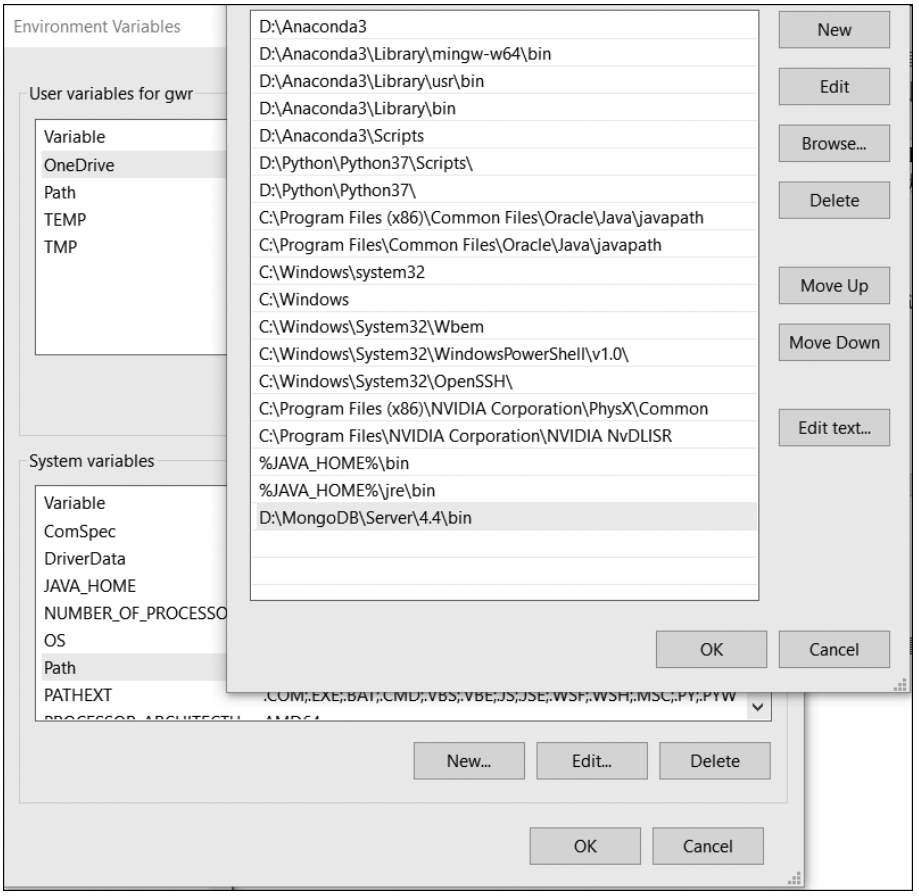

图 3-7　配置 MongoDB 环境变量

如图 3-7 所示，在系统变量 Path 中添加 MongoDB 服务 bin 文件夹路径(笔者使用的路径为“D:\MongoDB\Server\4.4\bin”)。

配置好环境变量后就可以在本地的任何位置使用 MongoDB 了。图 3-8 展示了如何使用命令行在任意目录文件下启动或关闭 MongoDB。

```
C:\Windows\system32>net start MongoDB
The MongoDB Server (MongoDB) service is starting.
The MongoDB Server (MongoDB) service was started successfully.


C:\Windows\system32>net stop MongoDB
The MongoDB Server (MongoDB) service is stopping.
The MongoDB Server (MongoDB) service was stopped successfully.
```

图 3-8　启动和关闭 MongoDB 服务

图 3-8 中，系统用户通过命令行指令“net start MongoDB”启动或通过命令行指令“net stop MongoDB”关闭 MongoDB 服务。

3.4　MongoDB 的基本操作

3.4.1　Mongo Shell 的使用

MongoDB 自带简洁但功能强大的 JavaScript Shell。在 JavaScript Shell 中输入一个变量会将变量的值转换为字符串打印到控制台上。

首先，以系统用户身份运行命令行，执行命令“net start MongoDB”开启 MongoDB 服务。执行命令“mongo”启动 Mongo Shell[5]。Shell 默认连接 test 数据库。要使用别的数据库，需要在服务器地址后添加斜杠和数据库名。

另一种连接数据库的方法像 SQL Shell 中一样，使用命令“use databaseName”，如果数据库不存在，则会创建一个新的数据库。

首先如图 3-9 所示操作使用--nodb 选项启动 Shell，而不连接数据库。

```
C:\Windows\system32>mongo --nodb
MongoDB shell version v4.4.1
> help
        db.help()                    help on db methods
        db.mycoll.help()             help on collection methods
        sh.help()                    sharding helpers
        rs.help()                    replica set helpers
        help admin                   administrative help
        help connect                 connecting to a db help
        help keys                    key shortcuts
        help misc                    misc things to know
        help mr                      mapreduce

        show dbs                     show database names
        show collections             show collections in current database
        show users                   show users in current database
        show profile                 show most recent system.profile entries with time >= 1ms
        show logs                    show the accessible logger names
        show log [name]              prints out the last segment of log in memory, 'global' is default
        use <db_name>                set current database
        db.mycoll.find()             list objects in collection mycoll
        db.mycoll.find( { a : 1 } )  list objects in mycoll where a == 1
        it                           result of the last line evaluated; use to further iterate
        DBQuery.shellBatchSize = x   set default number of items to display on shell
        exit                         quit the mongo shell
>
```

图 3-9　使用--nodb 选项启动 Shell

在如图 3-9 所示的 Mongo Shell 中，可用“help”查看常用指令，如“exit”指令用于退出 Mongo Shell，“db.help()”指令用于查看数据库级别的命令的帮助，“db.mycoll.help()”用于查看集合的相关帮助。

启动 Mongo 后可以如例 3-2 所示使用“db”查看当前连接数据库名词。

【例 3-2】 使用“db”连接数据库。

```
>use mongodb
Switched to db mongodb
>db
mongodb
```

使用“db.集合名”的方式来访问集合一般不会有问题，但如果集合名恰好是数据库类的一个属性时就不行了，JavaScript 只有找不到指定的属性时，才会将其作为集合返回。当有属性与目标集合同名时，可以使用 getCollection 函数。例如，要访问 version 这个集合，因为 db.version 是个数据库函数，就会返回正在运行的 MongoDB 服务器的版本。所以输入 db.version 会显示该函数的 JavaScript 源代码，而不是想显示的集合。例 3-3 是一个 db.version 的例子。

【例 3-3】 db.version 指令的使用和输出。

```
>db.version
function(){
return this.serverBuildInfo().version;
}
>db.version()
4.4.1
>db.getCollection("version")
test.version
```

因为 Shell 是功能完整的 JavaScript 解释器，所以 Mongo Shell 中可以运行任何 JavaScript 程序，比如例 3-4 中的基本运算、JavaScript 标准库调用和函数的调用。

【例 3-4】 Mongo Shell 中运行 JavaScript 程序。

```
> x=1
1
> x+x
2
> Math.sin(Math.PI/2)
1
> function func(n){ if(n<10) return 10; return n; }
> func(6)
10
```

例 3-4 中首先定义变量，输入后程序返回变量定义的值。然后对变量进行基本运算

时,就可以得到运算的结果。案例的第三条输入是 JavaScript 标准库函数 Math 函数的调用,返回运算结果。最后是一个自定义函数定义和运算的示例。

3.4.2 数据库和集合操作

表 3-2 展示了数据库基础操作及其指令。

表 3-2 数据库操作

数据库操作	指　令
创建数据库	use 数据库名
查看当前连接的数据库	db
查看所有数据库	show dbs
删除数据库	db.dropDatabase()

如表 3-2 所示,MongoDB 创建数据库的语法格式是“use 数据库名”。查看当前连接的数据库的指令是“db”。如果想查看所有数据库,可以使用指令“show dbs”。MongoDB 中默认的数据库为 test,如果没有创建新的数据库,集合将存放在 test 数据库中。删除数据库需要先切换到要销毁的数据库,然后执行指令“db.dropDatabase()”。

MongoDB 数据库对集合的操作对应于关系数据库中表的概念,包括集合的创建删除以及查询数据库中所有的集合。表 3-3 展示了 MongoDB 中对集合的操作。

表 3-3 集合操作

集 合 操 作	指　令
创建集合	db.createCollections("集合名")
删除集合	db.集合名.drop()
获取所有集合	show collections

如表 3-3 所示,使用“db.createCollections("集合名")”指令创建集合,使用 db.集合名.drop()指令删除集合,使用指令“show collections”获取所有集合的信息。

3.4.3 基本增删改查操作

1. 存入数据

MongoDB 数据库使用 insert()方法或 save()方法向集合存入文档,使用的语法如下。

```
db.集合名称.insert(条件)
```

save()在不指定_id 字段的情况下与 insert()方法类似。但如果指定_id 字段,save()方法会更新该_id 的数据,这一点与 insert()方法不同。可以使用命令 db.col.save

(document) 存入文档。save()方法通过传入的文档来替换已有文档。语法格式如下。

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

在这段 save()方法的代码中 document 指文档数据，writeConcern 是可选项，表示抛出异常的级别。

2. 删除数据

MongoDB 数据库使用 remove()函数移除集合中的数据。在删除集合中的数据前首先需要执行命令“use ＜数据库名＞”切换到指定数据库，这之后执行删除指令：db.集合名称.remove(条件)。在执行 remove()函数前先执行 find()命令来判断执行的条件是否正确，是一个比较好的习惯。请慎用 remove({})，它会删除集合中的所有文档。remove()方法的基本语法格式如下。

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

MongoDB 的 2.6 版本以后，其 remove()方法的语法格式发生了一些改变，具体语法格式如下。

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

上文代码中，query 是可选的，代表删除的文档的条件。justOne 也是可选的，如果设为 true 或 1，则只删除一个文档。writeConcern 表示抛出异常的级别，同样是可选择的。

3. 更新数据

update()方法用于更新已存在的文档。语法格式如下。

```
db.collection.update(
        <query>,
```

```
        <update>,
        {
                upsert: <boolean>,
                multi: <boolean>,
                writeConcern: <document>
        }
)
```

在上文代码中，query 字段表示 update()方法的查询条件，类似 SQL 语句的查询语句内 WHERE 字段后的内容。update 参数是 update()方法的对象和一些更新的操作符（如 $，$ inc…），也可以理解为 SQL 语句中 SET 字段后的内容。upsert 参数是可选的，它的意思是当不存在这一条 update 对应的记录时是否需要存入该记录。multi 可选字段的默认值是 false，表示只更新找到的第一条记录，当设置这个参数为 true 时，就把条件查出来的多条记录全部更新。可选的 writeConcern 参数抛出异常的级别。

要注意的是，直接使用 update()方法修改数据是十分危险的，因为如果被更新的列数小于总列数，更新后其他没有被更新的列就变成了空值。因此，实际应用中需要使用 MongoDB 提供的修改器 $ set 修改数据，这样在更新某一字段的同时可以完整地保留其他字段。

4. 查询数据

查询数据需要执行“use ＜数据库名＞”切换到指定数据库，然后执行“db.＜集合名＞.find()”查询全部符合条件的内容。为了避免游标可能带来的开销，MongoDB 还提供 findOne()的方法，用来返回结果集的第一条记录。执行“db.＜集合名＞.findOne()”查询一条符合条件的内容。要注意的是，find()方法和 findOne()方法后面都可以添加 JSON 条件。当需要返回查询结果的前几条记录时，可以使用 limit()方法“db.集合名称.find().limit(条件)”。

MongoDB 数据库查询数据时有很多灵活的高级查询方法。

（1）模糊查询：MongoDB 的模糊查询是通过正则表达式的方式实现的。格式为“/模糊查询字符串/”。

（2）对 Null 值进行处理。

（3）不等符号、存在判断符号、包含符号、条件符号等的使用：＜，＜＝，＞，＞＝，！＝等操作符和一些关系操作符也是很常用的。格式如例 3-5 所示。

【例 3-5】 find()函数中使用操作符。

```
db.collection.find({ "field" : { $gt: value } } );
db.collection.find({ "field" : { $lt: value } } );
db.collection.find({ "field" : { $gte: value } } );
db.collection.find({ "field" : { $lte: value } } );
db.collection.find({ "field" : { $ne: value } } );
db.collection.find({ "field" : { $exists: value } } );
```

```
db.collection.find({ "field" : { $in: value } } );
db.collection.find({ "field" : { $nin: value } } );
```

例 3-5 中，第一条语句表示查询 field 中所有大于 value 的值，第二条语句表示查询所有小于 value 的值，接下来的 $gte、$lte、$ne 分别表示大于等于、小于等于和不等于条件，$exists 表示判断 field 是否等于 value，最后的 $in 和 $nin 用于判断 field 和 value 的包含关系，如果 field 包含于 value 则 $in 成立，反之 $nin 成立。关系符通常用于连接操作符，比如如果需要查询同时满足上述两个条件的值，需要使用 $and 操作符将条件进行关联（相当于 SQL 的 and），格式为“$and:[{ },{ },{ }]”。

统计记录条件使用 count()方法。例 3-6 的两条语句分别表示查询 student 集合的文档条数和查询 student 集合中 age 字段小于等于 20 的文档条数。

【例 3-6】 查询 student 文档条数。

```
>db.student.count();
>db.student.count({age:{$lte:20}});
```

3.4.4 聚合和管道

MongoDB 中聚合使用 aggregate()方法，它主要用于处理数据（如统计平均值、求和等），并返回计算后的数据结果。有点类似 SQL 语句中的 count(*)。aggregate() 方法的基本语法格式为“db.集合名称.aggregate(聚合操作)”。表 3-4 展示了常用的聚合表达式。

表 3-4　聚合表达式

表达式	描　　述	实　　例
$sum	计算总和	db.mycol.aggregate([{ $group : { _id : " $by_user", num_tutorial : { $sum : " $likes"}}}])
$avg	计算平均值	db.mycol.aggregate([{ $group : { _id : " $by_user", num_tutorial : { $avg : " $likes"}}}])
$min	获取集合中所有文档对应值的最小值	db.mycol.aggregate([{ $group : { _id : " $by_user",num_tutorial : { $min : " $likes"}}}])
$max	获取集合中所有文档对应值的最大值	db.mycol.aggregate([{ $group : { _id : " $by_user", num_tutorial : { $max : " $likes"}}}])
$push	在结果文档中存入值到一个数组中	db.mycol.aggregate([{ $group : { _id : " $by_user", url : { $push: " $url"}}}])
$addToSet	在结果文档中存入值到一个数组，但不创建副本	db.mycol.aggregate([{ $group : { _id : " $by_user", url : { $addToSet : " $url"}}}])
$first	根据资源文档的排序获取第一个文档数据	db.mycol.aggregate([{ $group : { _id : " $by_user", first_url : { $first : " $url"}}}])
$last	根据资源文档的排序获取最后一个文档数据	db.mycol.aggregate([{ $group : { _id : " $by_user", last_url : { $last : " $url"}}}])

管道在 UNIX 和 Linux 中一般用于将当前命令的输出结果作为下一个命令的参数。MongoDB 的聚合管道将 MongoDB 文档在一个管道处理完毕后将结果传递给下一个管道处理。管道操作是可以重复的。表 3-5 介绍了聚合框架中常用的管道操作。

表 3-5　聚合框架中的管道操作

表达式	描　　述	实　　例
$ project	修改输入文档的结构。可以用来重命名、增加或删除域，也可以用于创建计算结果以及嵌套文档	db.article.aggregate({ $ project : { _id : 0 , title : 1 , author : 1 }});
$ match	用于过滤数据，只输出符合条件的文档。$ match 使用 MongoDB 的标准查询操作	db.articles.aggregate[{ $ match : { score : { $ gt : 70, $ lte : 90 }}}, { $ group: { _id: null, count: { $ sum: 1 }}}]);
$ limit	用来限制 MongoDB 聚合管道返回的文档数	db.article.aggregate({ $ limit : 5 });
$ skip	在聚合管道中跳过指定数量的文档，并返回余下的文档	db.article.aggregate({ $ skip : 5 });
$ unwind	将文档中的某一个数组类型字段拆分成多条，每条包含数组中的一个值	db.article.aggregate({ $ project: { author: 1, title: 1, tags: 1}}, { $ unwind: "$ tags"}) { " result " : [{ " _ id " : ObjectId (" 528751b0e7f3eea3d1412ce2"), "author" : "Jone", " title" : "A book", "tags" : "good" }, { "_id" : ObjectId("528751b0e7f3eea3d1412ce2"), "author" : "Jone", "title" : "A book", "tags" : "fun" }], "ok" : 1}
$ group	将集合中的文档分组，可用于统计结果	db.article.aggregate({ $ group : { _id : "$ author", docsPerAuthor : { $ sum : 1 }, viewsPerAuthor : { $ sum : "$ pageViews" }}});
$ sort	将输入文档排序后输出	db.users.aggregate({ $ sort : { age : −1, posts: 1 } });
$ geoNear	输出接近某一地理位置的有序文档	db.places.aggregate[{ $ geoNear: { near: [40.724, − 73. 997], distanceField: " dist. calculated ", maxDistance: 0.008, query: { type: " public" }, includeLocs: " dist. location ", uniqueDocs: true, num: 5}}])

管道表达式用于处理输入文档并输出，表达式是无状态的，只能用于计算当前聚合管道的文档，不能处理其他文档。每个管道表达式是一个文档结构，它是由字段名、字段值和一些表达式操作符组成的。管道操作符作为“键”，所对应的“值”叫作管道表达式。例如{ $ match: {status: "A"}}中，$ match 称为管道操作符，而{status: "A"}称为管道表达式，它可以看作是管道操作符的操作数(Operand)。

每个管道表达式只能作用于处理当前正在处理的文档，而不能进行跨文档的操作。管道表达式对文档的处理都是在内存中进行的。除了能够进行累加计算的管道表达式外，其他的表达式都是无状态的，也就是不会保留上下文的信息。累加性质的表达式操作符通常和 $ group 操作符一起使用，来统计该组内的最大值、最小值等。

在执行管道聚合时，如果 $ sort、$ skip、$ limit 依次出现的话，例如例 3-7 中执行的语句，这段语句在实际中会按照例 3-8 中的顺序执行。

【例 3-7】 管道聚合时，$ sort、$ skip、$ limit 依次出现。

```
{ $sort: { age : -1 } },
{ $skip: 10 },
{ $limit: 5 }
```

【例 3-8】 实际代码执行顺序。

```
{ $sort: { age : -1 } },
{ $limit: 15 },
{ $skip: 10 }
```

例 3-8 展示了实际过程中的执行顺序——$ limit 会提前到 $ skip 前面去执行。此时 $ limit 等于优化前 $ skip 加上优化前 $ limit。这样做的好处有以下两个。

(1) 在经过 $ limit 管道后，管道内的文档数量个数会“提前”减小，这样会节省内存，提高内存利用效率。

(2) $ limit 提前后，$ sort 紧邻 $ limit 的话，当进行 $ sort 的时候当得到前“$ limit”个文档的时候就会停止。

接下来的例 3-9～例 3-11 用一个更复杂的例子的分析和优化过程展示了这种优化的好处。

【例 3-9】 一个 $ limit 和 $ skip 交替出现的聚合管道案例。

```
{ $limit: 100 },
{ $skip: 5 },
{ $limit: 10},
{ $skip: 2 }
```

在例 3-9 的聚合管道案例中反复出现 $ limit 和 $ skip。例 3-10 对例 3-9 的序列按照例 3-8 的优化方式进行局部优化——将第二个 $ limit 提前。

【例 3-10】 局部优化后的聚合管道案例。

```
{ $limit: 100 },
{ $limit: 15},
{ $skip: 5 },
{ $skip: 2 }
```

可以看到，优化后的例 3-10 聚合管道案例中有连续的 $ limit 和连续的 $ skip。通过对两个 $ limit 取最小值，对两个 $ skip 直接相加的方式进一步优化得到例 3-11 的最终优化结果。

【例 3-11】 最终优化后的聚合管道案例。

```
{ $limit: 15 },
{ $skip: 7 }
```

例 3-11 的优化结果很好地展示了适当调整聚合管道中语句的顺序对整个管道运行效率的优化效果。

对聚合管道进行优化的方法还有很多，如较早地使用 $ project 投影，设置需要使用的字段，去掉不用的字段，可以大大减少内存。较早地使用 $ match、$ limit、$ skip 操作符可以提前减少管道内文档数量，减少内存占用，提高聚合效率。除此之外，尽量将 $ match 放到聚合的第一个阶段（这样的话 $ match 相当于一个按条件查询的语句），可以使用索引，加快查询效率。

要注意的是，管道操作中会有一些引起错误的地方，以下几点为常见的错误。

（1）管道线的输出结果不能超过 BSON 文档的大小（16MB），如果超出会产生错误。

（2）如果一个管道操作符在执行的过程中所占有的内存超过系统内存容量的 10%，会产生一个错误。

（3）执行 $ sort 和 $ group 操作符的时候，整个输入都会被加载到内存中，如果这些占有内存超过系统内存的 5%，会将一个 Warning 记录到日志文件中。当占有的内存超过系统内存容量 10%的时候，会产生一个错误。

3.4.5 索引操作

索引是一种特殊的数据结构，它是对数据库表中一列或多列的值进行排序的一种结构，存储在一个易于遍历读取的数据集合中。索引项的排序支持高效的相等匹配和基于范围的查询操作。

在 MongoDB 中建立索引能提高查询效率。操作时只需要扫描索引（存储集合的一小部分，并把这小部分加载到内存中）。如果没有建立索引，在查询时，MongoDB 必须执行全表扫描，特别是在处理大量的数据时，查询可能要花费几十秒甚至几分钟，这对网站性能的影响是非常致命的。

早期 MongoDB 使用 ensureIndex()方法创建索引。ensureIndex()方法基本语法格式如下。

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

其中，KEY 值为待创建的索引字段，1 为指定按升序创建索引，按降序创建索引需指定 KEY 为－1。

从 MongoDB 3.0 开始，ensureIndex()方法被废弃，开始使用 createIndex()方法创建索引。创建索引的语法如下。

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

上文创建的索引是一个包含该字段的字段和值对的文档，该文档的索引键和该值描述该字段的索引类型。对于某个领域的上升索引，指定一个值为 1；对于下降的索引，指定一个值为－1。

查看索引信息使用 getIndexes()方法，语法格式如下。

```
> db.collection.getIndexes()
```

该函数返回一个数组，这个数组保存标识和描述集合上现有索引的文档列表，可以查看开发人员是否对某集合创建了索引并查看其创建了哪些索引。

创建单列索引的方式如下。

```
>db.collection.createIndex({field: boolean} )
```

上文语句用于对文档单个字段创建索引或者对内嵌文档的单个字段创建索引。field 字段以“.”来指明内嵌文档的路径。对于某个领域的上升索引，指定 boolean 值为 1；对于下降的索引，指定 boolean 值为－1。

除了单字段创建索引外，还可以同时对多个键创建组合索引，创建组合索引语法如下。

```
>db.collection.createIndex({field1:boolean, field2:boolean } )
```

内嵌文档的索引代码如下。

```
>db.collection.createIndex({field:boolean} )
```

创建索引是为了提高文档查询的效率，往往创建索引后根据条件查询文档的查询时间会大大缩短。在大数据时代，有无索引带来的查询效率差别更为显著。当将查询和排序组合进行时，文档查询效率会进一步提高。在 MongoDB 数据库中建立索引能提高查询效率，但在 MongoDB 中新增和修改的效率就会下降。

删除索引使用 db.collection.dropIndex(index)方法。对已经创建的索引进行删除时，可以针对具体集合中的索引进行删除，也可以对所有集合中的所有索引进行删除。删除具体的索引要根据索引的名称。如果不知道索引的名称，可以通过 db.collection.getIndexes()方法查看索引名称。

要注意的是，创建索引也是有一定限制的。

(1) 索引具有额外开销。存储在 MongoDB 集合中的每个文档都有一个默认的主键，这就是默认索引。如果在添加新的文档时，没有指定主键的值，MongoDB 就会创建一个 ObjectId 值，并会自动创建一个索引在主键上，默认索引的名称是“_id_”，并无法删除。

(2) 索引使用内存(RAM)。因为索引存储在内存中，所以索引的大小不能超过内存的限制。如果索引的大小超过了内存的限制，MongoDB 就会删除一些索引，这将导致性能下降。

(3) 索引具有查询限制。索引不能被正则表达式、非操作符(如 $nin, $not)、算术运算符(如 $mod)和 $where 子句查询使用。所以,检测语句使用索引是一个好的习惯。

(4) 存在索引键限制。从 MongoDB 2.6 版本开始,如果现有的索引字段的值超过索引键的限制,数据库不会再创建索引。如果文档的索引字段值超过了索引键的限制,MongoDB 不会将任何文档转换成索引的集合。

(5) 索引具有最大范围。集合中索引不能超过 64 个,索引名的长度不能超过 125 个字符,并且一个复合索引最多只能有 31 个字段。

3.5 通过 Java 访问 MongoDB

3.5.1 数据库和集合操作

在 Java 中使用 MongoDB 数据库之前,首先需要拥有 Java 连接 MongoDB 的第三方驱动包(jar 包)。将 MongoDB JDBC 驱动加入到 Java 项目后,就可以对 MongoDB 进行操作了。

Java 连接 MongoDB 分为不通过认证连接和通过认证连接两种。

1. 不通过认证连接 MongoDB

Java 不通过认证连接 MongoDB 服务使用 MongoClient,MongoClient 提供连接到 MongoDB 服务器和访问数据的功能。例 3-12 是 MongoClient 的三种构造方法。

【例 3-12】 MongoClient 的三种构造方法。

```
MongoClient mongoClient = new MongoClient();
MongoClient mongoClient1 = new MongoClient("localhost");
MongoClient mongoClient2 = new MongoClient("localhost", 27017);
```

在例 3-12 中,“localhost”表示连接的服务器地址,27017 为端口号,因为系统默认端口号为 27017,所以可以省略端口号不写。同时将服务器地址和端口号都省略时,系统默认设置服务器地址为“localhost”,端口号为 27017。

2. 通过认证连接 MongoDB

通过认证连接 MongoDB 服务在记录端口和服务器地址的前提下要额外记录用户名、数据库名称和密码。通过认证连接 MongoDB 的案例代码如例 3-13 所示。

【例 3-13】 通过认证连接 MongoDB。

```
List<ServerAddress> adds = new ArrayList<>();
ServerAddress serverAddress = new ServerAddress("localhost", 27017);
adds.add(serverAddress);
List<MongoCredential> credentials = new ArrayList<>();
MongoCredential mongoCredential = MongoCredential.createScramSha1Credential
( "username", "databaseName", "password".toCharArray() );
```

```
credentials.add(mongoCredential);
MongoClient mongoClient = new MongoClient(adds, credentials);
```

如例 3-13 所示，首先使用 ServerAddress()记录服务器地址和端口号并添加到 adds 中作为 MongoClient 的第一个参数，然后使用 MongoCredential.createScramSha1Credential()方法记录用户名、数据库名称和密码，最后使用 MongoClient 进行认证连接。具体代码在下文给出。

接着连接到数据库，所用代码如例 3-14 所示。

【例 3-14】 连接到 test 数据库。

```
MongoDatabase mongoDatabase = mongoClient.getDatabase("test");
```

例 3-14 的 test 指数据库名，若指定的数据库不存在，MongoDB 将会在开发人员第一次存入文档时创建数据库。使用 void drop()可以删除当前连接的数据库。

MongoDB 中的数据都是通过文档保存的，而文档又保存在集合中。要对数据进行 CRUD 操作首先要获取待操作的集合。例 3-15 演示了获取集合的方法。

【例 3-15】 获取集合的方法。

```
MongoCollection<Document> collection = MongoDBUtil.getConnect()
.getCollection("user");
```

代码中的 user 表示集合的名字，如果指定的集合不存在，MongoDB 将会在开发人员第一次存入文档时创建集合。当然，也可以使用例 3-16 的方法直接新建集合。

【例 3-16】 创建一个名为 collectionName 的集合。

```
void createCollection(String collectionName);
```

3.5.2　基本增删改查操作

要对数据进行增删改查操作，首先要创建文档对象，例 3-17 展示了如何创建一个文档。

【例 3-17】 创建一个文档。

```
Document document = new Document("name","张三")
.append("state", "A")
.append("age", 42);
```

例 3-17 创建了一个姓名为“张三”，状态为“A”，年龄为 42 的人物文档。

1. 增添文件

创建的文档需要存入数据库保存。MongoDB 提供了保存数据的三种方法，分别是 db.collection.insertOne()，db.collection.insertMany()和 db.collection.insert()。