文档数据库

文档数据库是以文档为基本单位存储数据的一种 NoSQL 数据库。文档数据库允许创建任意复杂的结构模式，具有非常高的灵活性，而且具有类似关系数据库的强大查询和统计功能。当键值数据库无法满足较为复杂的数据建模需求时，通常会考虑文档数据库。本章从理论方面介绍文档与其描述方法、集合与结构设计、文档关系建模和数据分区等内容；从技术方面以 Mongo 文档数据库为例，介绍相应的操作语句；最后给出一个文档数据库的应用实例。

3.1 文档及其描述方法

3.1.1 文档概念

文档数据库中“文档”的含义不同于一般意义上的“文档”，如 Word、HTML 或者 PDF 等文档。实际上，文档数据库中的文档(Document)指的是若干个键值对(也可以称为“列名称”和“列值”、“字段名”和“字段值”)的有序集合，而键值对是由键和值组成的，且每个键值对只能出现一次。

下面是一个以 JSON 格式描述的文档。

```
{
    name: 'Kitty',
    age: 28,
    status: 'B',
    groups: ['Dance', 'Sports'],
    Address: {street: 'Hangzhou Zhaohui road', ZipCode:'320014'}
}
```

上述文档的开始和结束分别是一对大括号，里面包含 5 个键值对，第一个键值对的键名是 name，键值是字符串"Kitty"；第二个键值对的键名是 age，键值是数值 28；第三个键值对的键名是 status，键值是字符串"B"；第四个键值对的键名是 groups，键值是一个数组，包含两个数组元素；第五个键值对的键名是 Address，键值是一个子文档，该子文档包括两个键值对。由此可见，文档中的键值对既可以是简单数据类型，也可以是复杂数据类型，这为设计文档数据库提供了极大的灵活性。

文档数据库中的文档包含了多个键名及其对应的键值，其实这与关系数据库中的基本

表类似，所不同的是基本表中除表头外都是以元组形式存储数据。而文档数据库中的每个文档都有自己的键（属性）和键值（属性值）。与键值数据库相比，文档数据库损失了一定的性能，但能够一次性存储更多的键值对；与关系数据库相比，文档数据库提供了灵活性，不需要预先设计文档结构，开发者可以在数据写入时根据需要灵活定义文档结构，而在关系数据库中，往往要先设计好表结构。

3.1.2 文档描述

文档通常用 JSON 格式编写，也可以用 XML 格式描述。JSON（JavaScript Object Notation）是一种轻量级的数据交换格式，它采用完全独立于编程语言的文本格式存储和表示数据，其简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言，它易于阅读和编写，同时也易于机器解析和生成，并有效地提升网络传输效率。

本书仅以 JSON 描述文档，在 JSON 使用时需要遵循以下语法规则。

- 以左大括号"{"开头，以右大括号"}"结尾。
- 数据以键值对的形式出现。
- 键值对之间以逗号分隔。
- 键名是字符串，表示某个属性。
- 键值可以是数值、字符串、逻辑值、数组、对象或 Null，表示具体值。
- 数组里的各个元素放在中括号[]中。
- 键值也可以键值对的形式表示，放在一对大括号{}中。

键名一般用字符串表示，也称为属性名、字段名；键值可以是基本的数据类型，如数值、字符串和 Boolean 型，也可以是结构化的数据类型，如数组、对象或子文档等，键值也称为属性值、字段值。因此，文档中既有结构化的信息，又包含数据本身。下面是用 JSON 描述的一个较复杂的文档。

```
{
    book_id: "B01",
    book_name: "红岩",
    author: {
        author_id: "A01",
        author_name: "罗广斌",
        address: "浙江省杭州市西湖区",
        zipcode: "315131",
        email: "guangbin@ yahoo.com"
    },
    price:36,
    publisher:"中国青年出版社"
}
```

上述文档中，键 author 的值是一个子文档，该子文档包括若干个键值对。

3.2 集合及其结构

3.2.1 集合概念

集合(Collection)由一组相关的文档构成,同一集合内的文档结构可以不同。集合没有类似关系数据库基本表的数据类型和约束等要求,无须提前为集合中的文档定义模式,可直接在集合中插入文档,且文档结构由开发者自己定义。从概念上,集合可被视作关系数据库的基本表,文档被视作元组。

下面是一个包含了 2 个文档的集合,第一个文档是“Sue”的信息,第二个文档是“Kitty”的信息,这里两个文档的结构是相同的。

```
{
    name: 'sue',
    age: 26,
    status: 'A',
    groups: ['Sing', 'Sports'],
    address: {street: 'Hangzhou Liuhe road', ZipCode: '003210'}
}
{
    name: 'Kitty',
    age: 28,
    status: 'B',
    groups: ['Dance', 'Sports'],
    address: {street: 'Hangzhou Zhaohui road', ZipCode: '320014'}
}
```

一般地,集合内的文档通常都与同一个主题相关,如产品、学生、课程、事件等。不同的主题虽然也可以存储在同一集合内,但会显得比较混乱,不建议这样存储。相同主题的各个文档不需拥有完全相同的结构。如果有 10%的文档需要记录属性 A 和属性 B,就不应该强迫另外 90%的文档也记录这些内容。例如,下面的集合中存储了三个学生文档,其结构并不相同。

```
{
    UserId: "1001",
    UserName: "张三",
    PassWord: "123456"
}
{
    UserId: "1002",
    UserName: "李四",
    PassWord: "123456",
    Detail: { "Address": "湖北", "Age": 20, "Email": "lisi@ 163.com" }
```

```
}
{
    UserId: "1003",
    UserName: "赵六",
    PassWord: "123456",
    Detail: { "Address": "湖北" },
}
```

多个不同的文档集合构成一个文档数据库。

3.2.2 集合结构

集合包括一组类似的文档，而文档自身也可以包含子文档，这些子文档称为嵌入式文档。通过嵌入式文档，能够将相关的文档放在一块，从而不需要使用连接操作就可以直接查询完整的数据。图 3-1 是一个一般的集合结构，该集合包含多个文档，每个文档又包含嵌入式文档。

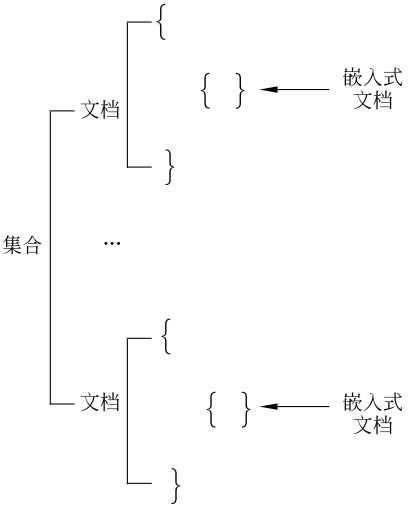

图 3-1 包含了嵌入式文档的集合

在关系数据库中，数据与联系存储在不同的基本表中。例如，如果要存储学生基本信息、课程信息以及学生的选修课信息，至少要用到 3 张基本表，然后通过外键在这些基本表之间建立联系。在查询学生的选修课情况时，再通过连接操作将这些基本表连接起来。连接操作涉及大量数据，导致高昂的读写开销，将会影响查询性能。

在文档数据库中，数据与联系可以存储在同一文档中，其目的是提高数据查询的性能，这种设计方法称为“去规范化”。去规范化与规范化的设计方法正好相反，其目的是减少连接操作引起的读写开销，以改善查询性能。当然，去规范化很容易产生数据冗余和数据不一致性，数据一旦被写入，就不再更新，这样做也是合理的。如图 3-2 所示，左侧有两个文档，一个文档存储了订单的信息，另一个文档存储了订单对应的商品信息下，右侧将这两个文档合并为一个文档，商品信息作为一个子文档存储在订单文档中。

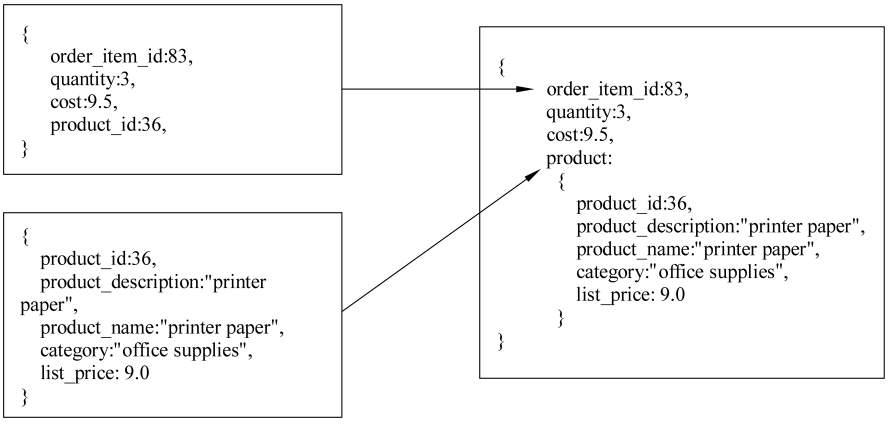

图 3-2 去规范化示例

3.2.3 无模式数据库

模式在数据库中是一个重要的概念。一般地，在数据库中模式指数据的“结构”或“纲要”。在关系数据库中，模式就是关系模式，也称为表结构。关系数据库要求设计者预先设计关系模式(表结构)，即先定义模式名、列、主键、外键和其他约束等。此外，为了减少数据冗余、消除数据异常，关系数据库采取关系模式分解，将一个比较大的关系模式分解为若干小的关系模式，分解后的关系模式一般都要满足一些范式要求，如第一范式、第二范式、第三范式等。这种将一个低级别的关系模式通过分解转换为一组高级别范式的过程称为规范化。规范化是设计关系模式的重要理论依据，因此关系数据库中的关系模式是规范化的模式集。

然而，关系数据库规范化也带来以下两个问题：

(1) 规范化之后得到许多小的关系模式，在进行查询操作时必须将这些关系模式连接(Join)起来才能查询到比较全面的信息，而连接操作往往需要占用不少的内存资源，查询性能不高。

(2) 关系模式的模式结构往往是固定的，无法满足应用灵活多变的需求。例如，要设计一个基本表来存储商品数据，但商品的属性个数可能不一致，甚至商品的属性个数是未知的，此时就难以设计一个固定的模式。

与关系数据库相比，文档数据库不需要预先指定模式结构，属于无模式数据库或去规范化数据库。这个特点使得文档数据库具有更高的灵活性，由应用程序在使用过程中灵活地定义具体的模式，可以随时向集合中添加文档，并定义键值对。

实际上，文档数据库的同一个集合可以存储不同类型的文档，也可以具有不同的结构，没有限定性规范来约束文档的结构。文档数据库的这种无模式结构为应用程序提供了灵活性。

3.3 文档关系建模

根据文档与文档之间的逻辑关系，可以使用嵌入式文档或引用标识符对文档结构进行建模。

3.3.1 一对多的文档关系

若对于文档集合 A 中的每一个文档，文档集合 B 中有多个文档与之联系，反之，对于文档集合 B 中的每一个文档，文档集合 A 中至多有一个文档与之联系，则称文档集合 A 与文档集合 B 具有一对多的文档关系。

在一对多的文档关系中，表示“一”的文档称为主文档，而表示“多”的文档则用嵌入式文档构成的数组表示，并通过数组作为主文档的键值。例如，某客户有 2 个地址，此时可以通过嵌入式文档描述客户和地址的关系，示例如下。

```
{
    person_id:319,
```

```
    name:"RuiZhao",
    occupation:"teacher",
    address: [
              {street:"Zhaohui road", zip: 310014},
              {street:"Liuhe road", zip:310032}
            ]
}
```

上述文档中的主文档是客户的基本信息，地址以数组的形式存储了两个嵌入式文档，每个嵌入式文档对应一个地址。通过嵌入式文档所构成的数组能够灵活地表示一对多的文档关系。

3.3.2 多对多的文档关系

若对于文档集合 A 中的每一个文档，文档集合 B 中有多个文档与之联系，反之，对于文档集合 B 中的每一个文档，文档集合 A 中也有多个文档与之联系，则称文档集合 A 与文档集合 B 具有多对多的文档关系。例如，一名学生可以选修多门课程，一门课程也可以有多名学生选修；一个订单可以包含多个产品，一个产品也可以包含在多个订单中。

对于多对多的文档关系，首先对这两类实体集分别构建一个文档集合，例如先构建一个学生文档集合，该集合存储了每个学生的文档；再构建一个课程文档集合，该集合存储了每个课程的文档；然后在各自的文档中分别加入一份其对应文档的标识符列表。比如课程 maths 有三个学生选修，学生 Hua Zhang 选修了三门课，那么这两个文档可以描述如下。

```
{                                          {
  course_id: C01,                            StudentID: 'S01',
  name:"maths",                              name:"HuaZhang",
  credits: 4                                 courses: [ 'C01','C02','C03']
  enrolledStudents:[ 'S01','S02','S03']    }
}
```

在文档数据库中，文档标识符是一个文档的唯一标识。可以通过文档标识符引用相关的文档，而不是把整份文档直接嵌入，这有助于减少数据冗余。需要注意的是，在文档数据库中没有参照完整性约束，文档之间的引用与被引用关系需要由开发者管理，在使用的时候要特别注意。

实际上，对于一对多的文档关系，也可以采用文档标识符的方法在文档之间建立相应的关系，例如计算机学院包含计算机系、软件工程系和物联网系，那么在计算机学院的文档中可以增加一个列表，指向这三个系的文档标识符。

```
{
    College:'C01',
    name:'College of Computer Science and Technology'
    head: 'Gang Zhang'
```

```
    dept:['D01', 'D02','D03']
}
```

通过嵌入式文档和文档标识符能够对文档的多种关系进行建模，具体采用何种方式取决于查询文档的效率以及维护文档的方便性，可以灵活设计。

3.4 文档数据分区

互联网环境下，文档的体量巨大，如果将所有文档存储在一台服务器上，会导致服务器访问压力过大。为此，需要将数据进行分区，即按照某种策略将数据从逻辑上划分为若干组成部分，每个组成部分称为一个数据分区，然后将这些分区分别存储在不同的集群节点上，从而提高数据访问性能和可扩展性。文档数据库的分区策略主要有两种：一种是文档垂直分区；另一种是文档水平分区。

3.4.1 文档垂直分区

文档垂直分区是指将一个文档的键值对按照某种策略分别存储在不同的集群节点上，此时每个分区包含了文档的不同键值对。例如，学生文档包含了学号、姓名、年龄、成绩和课外活动键值对，可以将学号、姓名、年龄和成绩键值作为一个数据分区存储在一个节点，将课外活动作为另外一个数据分区存储在另外一个节点，如图 3-3 所示。

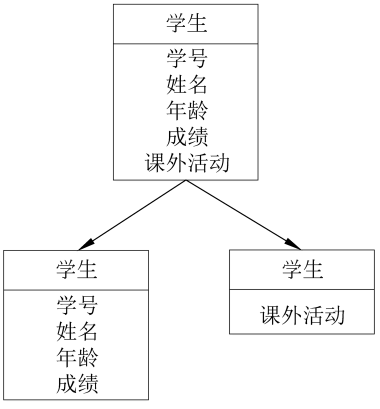

图 3-3 垂直分区：按键值对进行分区

显然，垂直分区把一个文档的数据分别存储在不同的分区上，这种策略往往很少采用，除非一个文档的一部分数据很少被访问，而另外一部分数据被频繁使用，这时候垂直分区才是有益的。

3.4.2 文档水平分区

文档水平分区是指将文档按照某一策略分别存储在不同的集群节点上。划分之后得到的数据分区称为“分片”，每个分片拥有文档的全部键值对。

键值数据库中每个键值对存储在哪个分区是由键名决定的，实现的一般策略是通过哈

希函数得到键名的哈希值，然后根据哈希值决定将该键值对存储到哪个节点上。文档数据库的每个文档有多个键值对，可以选择其中一个键作为依据对文档进行分区。用来对文档进行分区的键称为分区键。分区算法根据分区键将文档映射到不同的集群节点上。分区键一般选择便于对文档进行分类的键，如类型、日期、地域等。图 3-4 是按照季度对文档进行分区，其中每个季度是一个分区，这样文档数据会被均衡地存储在集群节点上。

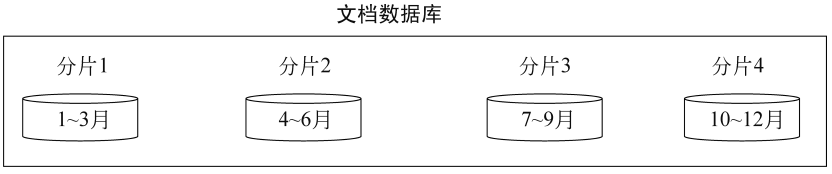

图 3-4 水平分区：按季度分区

分区算法一般采用哈希算法实现，也可以采用其他算法，例如根据属性的不同类别将文档存储在不同的分区上。

3.5 MongoDB 数据库

3.5.1 概述

MongoDB 自 2009 年 2 月推出，经过多年的发展已经趋于成熟，是一个分布式开源文档数据库管理系统，提供了具有可扩展性的高性能数据存储方案，具有以下特点。

(1) 每个集合可以存储不同的文档，不同文档的结构和大小有差异。

(2) 不支持复杂的连接操作。

(3) 提供了丰富的查询功能，支持对文档的动态查询。

(4) 支持索引，通过创建索引可以提高查询性能。

(5) 支持分布式文件，易扩展。

(6) 使用内存存储文档，速度快。

(7) 支持 C、C++ 、Java、PHP、Python 等多种应用程序开发语言。

MongoDB 使用 BSON(Binary JSON)格式描述文档。BSON 是一种类 JSON 的存储格式，支持内嵌的文档对象和数组对象，具有轻量性、可遍历性、高效性等特点。MongoDB 与关系数据库之间的术语对应关系如表 3-1 所示，熟悉这些术语便于更快地理解 MongoDB。

表 3-1 MongoDB 与关系数据库之间的术语对应关系

MongoDB 术语	关系数据库术语	含 义 说 明
数据库	数据库	数据库
集合	基本表	集合对应基本表
文档	元组	文档对应表中的元组
键值对	字段和字段值	键名对应字段名，键值对应字段值
索引	索引	索引

续表

MongoDB 术语	关系数据库术语	含义说明
文档 ID	主键	MongoDB 自动将_ID 键设置为主键
/	连接操作	MongoDB 不支持连接操作;关系数据库支持连接操作

下面以图书、作者、客户及其订单信息表为例,介绍 MongoDB 提供的查询语句。

(1) Books 图书信息表(见表 3-2)。

表 3-2 Books 图书信息表

book_id	book_name	author_id	price	publisher
B01	3D MAX 标准教程	A01	38	人民邮电出版社
B02	Windows 2000 网络管理	A02	40	北京航空航天大学出版社
B03	MySQL 数据库开发教程	A01	45	人民邮电出版社
B04	3D MAX 从入门到精通	A01	45	国防工业出版社
B05	西湖民间故事	A03	19	浙江文化出版社
B06	世界尽头与冷酷仙境	A04	19	作家出版社
B07	挪威的森林	A04	19	作家出版社
B08	寻羊冒险记	A04	20	作家出版社
B09	Linux 常见问题与技巧	A06	24	铁道出版社
B10	Office XP 入门与提高教程	A06	33	铁道出版社
B11	Office XP 办公自动化教程	A06	31	铁道出版社

(2) Authors 作者信息表(见表 3-3)。

表 3-3 Authors 作者信息表

author_id	author_name	address	zipcode	email
A01	刘耀儒	浙江省杭州市西湖区	310023	633423434@qq.com;
A02	王晓明	北京市东城区	100010	lxm@163.com
A03	卫慧	湖北省武汉市洪山区	430070	weihui@163.com
A04	村上春树	江苏省苏州市相城区	215131	cunshang@yahoo.com
A05	陈丹燕	江苏省南京市江宁区	211100	danyan@zjut.edu.cn
A06	张星	浙江省杭州市下城区	310000	zhangxing@163.com

(3) Clients 客户信息表(见表 3-4)。

表 3-4 Clients 客户信息表

client_id	client_name	address	zipcode	tel
C01	赵军	上海市浦东新区川	200120	13749302134
C02	李铁	海南省海口市美兰区	570100	15908204668
C03	夏添	北京市海淀区	100191	13925888532

(4) Orders 订单信息表(见表 3-5)。

表 3-5 Orders 订单信息表

order_id	book_id	book_number	order_date	client_id	comments
O01	B01	500	2003-01-01	C01	A
O02	B05	350	2003-02-28	C02	A
O03	B04	800	2001-10-11	C01	A
O04	B10	1000	2002-07-04	C03	B

3.5.2 数据库管理

MongoDB 拥有如下三个自带的数据库。

(1) admin 数据库：用于存储用户和角色等信息，如果将一个用户添加到这个数据库，这个用户自动继承所有数据库的权限。

(2) local 数据库：用来存储副本集的配置信息，数据不能被复制到其他节点。

(3) config 数据库：在分片设置时用于保存分片的相关信息。

除上述三个自带的数据库外，用户可以自己创建数据库，但数据库名称应该符合以下命名规则：

(1) 数据库名不能是空的字符串，不能以数字开头。

(2) 数据库名不能含有空格、.、$、/、\和\0 (空字符)等特殊符号。

(3) 数据库名应大小写敏感。

(4) 数据库名长度最多为 64B。

如果用户不定义自己的数据库名称，则默认操作的数据库名称为"test"。

1. 创建数据库

```
➢ use DATABASE_NAME
```

如果数据库名不存在，则创建该数据库；如果数据库名存在，则切换到指定数据库。实际上，只有当插入文档时才会真正创建该数据库。

例 3-1 创建 BookDB 文档数据库，其语句为

```
use BookDB
```

以上语句创建一个名为 BookDB 的文档数据库，并切换到该数据库。在向新创建的数据库插入文档之前，该数据库不会被真正创建。

2. 显示数据库

```
➢ show dbs
```

上述语句可以显示数据库的列表。

例 3-2 查看当前服务器上的所有文档数据库，其语句为

```
show dbs
```

刚才创建的数据库 BookDB 并不在数据库的列表中，如果要显示它，需要向 BookDB 数据库插入一些文档，这时数据库 BookDB 才能被真正创建，并通过上述语句显示出来。

3. **删除数据库**

```
➢ db.dropDatabase()
```

上述语句将删除当前数据库，其中的 db 将引用当前操作的数据库。

3.5.3 集合管理

在 MongoDB 中，集合是一组相关的文档，类似于关系数据库中的基本表。

1. **创建集合**

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

参数说明如下。

- name：要创建的集合名称。
- options：可选参数，指定相关选项，具体包括以下四个。

(1) capped：布尔值，如果为 true，则创建固定集合(Capped Collections)，即集合的大小是固定的，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。

(2) autoIndexId：布尔值，3.2 版本之后不再支持该参数，默认为 false；如果为 true，则自动在_id 键名创建索引。

(3) size：数值，定义固定集合的大小，单位为千字节。

(4) max：数值，指定固定集合文档的最大数量。

一般情况下创建的集合是没有大小的，可以一直向集合中添加文档，集合可以动态增长。如果创建的是固定集合，那么这种集合的大小是固定的。例如，在创建的时候设置该集合中文档的数量为 1000 个，那么当插入的文档数量达到 1000 个时，再向集合中插入文档，则只会保留最新的 1000 个文档，之前的文档会被删除。此外，固定集合的文档按照插入顺序存储，默认情况下查询就是按照插入顺序返回。

此外，对固定集合中的文档可以进行更新，但更新不能导致文档大小变化，否则更新将失败。例如，假设集合中有一个 key，其 value 对应的数据长度为 100B，如果要更新这个 key 对应的 value，更新后的值也必须为 100B。

例 3-3 在 BookDB 中创建一个集合 Books，其大小为 2048KB，若其已满，则删除旧的文档，最多存放 1000 个，其语句为

```
db.createCollection("Books",{capped:true,size:2048,max:1000})
```

2. 查看集合

➢ **show collections 或 show tables**

上述语句可以查看当前数据库上已创建了哪些集合。

例 3-4 显示 BookDB 中的所有集合，其语句为

```
showcollections
```

可以看到，Books 集合已经创建成功。

3. 删除集合

➢ **db.COLLECTION.drop()**

上述语句可以删除指定的集合，**COLLECTION 是要删除的集合名称**。

例 3-5 删除 Books 集合，其语句为

```
db.Books.drop()
```

上述语句将会删除 Books 集合，集合中的文档也被同时删除。

3.5.4 文档管理

1. 插入文档

1）insert 语句

➢ **db.COLLECTION.insert(document)**

将文档插到集合 COLLECTION 中。若插入的数据主键已经存在，则会抛出异常，提示主键重复，不保存当前数据。

例 3-6 在 Books 集合中插入“挪威的森林”文档，将作者信息作为文档的嵌入式文档，其语句为

```
db.Books.insert(
    {
        book_id:"B07",
        book_name:"挪威的森林",
        author:{
        author_id:"A04",
        author_name:"村上春树",
        address: "江苏省苏州市相城区",
        zipcode:"215131",
        email:"cunshang@yahoo.com"
```

```
    },
    price:19,
    publisher:"作家出版社"
    }
)
```

上述语句将插入一个文档，键名 author 的键值是一个嵌入式文档。

2）insertOne 语句

```
➢ db.COLLECTION.insertOne(document)
```

通过上述语句，可以向集合 **COLLECTION** 中插入一个文档。

例 3-7 在 Books 集合中插入图书编号是 B02 的图书信息，其语句为

```
db.Books.insertOne(
  {
     book_id:"B02",
     book_name: "Windows2000 网络管理",
     author_id: "A02",
     price:40,
     publisher:"北京航空航天大学出版社"
  })
```

3）insertMany 语句

```
➢ db.COLLECTION.insertMany([<document1>, <document2>, ... ], {writeConcern: <
  document>, ordered: <boolean>})
```

参数说明如下。

- writeConcern：表示指定写入策略，默认为 1，即要求确认写操作，0 为不要求。
- ordered：表示是否按顺序写入，默认为 true，要求按顺序写入。

该语句可以向集合中一次插入一个或多个文档。

例 3-8 在 Books 集合中插入“Office XP 入门与提高教程”和“Office XP 办公自动化教程”两个文档，其语句为

```
db.Books.insertMany(
   [
      {
         book_id:"B10",
         book_name: "Office XP 入门与提高教程",
         author_id: "A06",
         price:33,
         publisher:"铁道出版社"
```

```
    },
    {
            book_id:"B11",
            book_name: "Office XP 办公自动化教程",
            author_id: "A06",
            price:31,
            publisher:"铁道出版社"
    }
])
```

除用户可以为每个文档设置有意义的标识符以区分不同文档外，MongoDB 还自动为每个文档设置了一个_id 主键，默认情况下是 Objectid 对象，是自动生成的，这个值在当前集合中是唯一的，应用程序可以使用_id 值标识文档。

2. 更新文档

(1) Update 语句

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

参数说明如下。

- query：更新的条件；如果有条件，则只更新符合条件的文档。
- update：更新的对象和更新的操作符，具体包括以下更新操作符。
  {$set:{field:value}}：把文档中某个 field 的值设为 value。
  {$inc:{field:value}}：把文档中某个数值型的 field 增加一个 value 值。
  {$unset:{field:0}}：删除某个 field。
  {$push:{field:value}}：把 value 追加到数组 field 里，如果数组 field 不存在，则会自动插入一个数组类型。
  {$addToSet:{field:value}}：加一个值到数组 field 中，而且只有当这个值在数组中不存在时才增加。
  {$pull:{field:value}}：从数组 field 中删除一个等于 value 的值。
  {$rename:{old_field_name:new_field_name}}：对 field 进行重命名。
- upsert：可选参数，如果不存在所要更新的记录，是否插入一个新文档，true 为插入，默认是 false，不插入。
- multi：可选参数，默认是 false，只更新找到的第一个文档，如果这个参数为 true，就把符合条件的文档全部更新。
- writeConcern：可选，抛出异常的级别。

例 3-9 将 Books 集合中所有图书的价格统一增加 10 元，其语句为

```
db.Books.update({"price":{"$exists":true}},{$inc:{price:10}},false,true)
```

上述语句中 $exists 的作用是判断键 price 是否存在，$inc 的作用是将键 price 加上一

个数值;false 的作用是如果不存在键 price,则取消更新;true 的作用是如果有多个文档符合条件,则全部更新。

例 3-10 将 Books 集合中的书名“挪威的森林”修改为“森林甲壳虫”,其语句为

```
db.Books.update({"book_name":"挪威的森林"},{$set:{"book_name":"森林甲壳虫"}})
```

由于 Books 是固定集合,因此要求更新后的键值长度与之前的长度保持一致。

例 3-11 将 Books 集合中书名为“Windows 2000 网络管理”的文档中的键“price”删除,其语句为

```
db.Books.update({"book_name":"Windows 2000 网络管理"},{$unset:{"price":0}})
```

在上述语句中,注意 unset 这个操作符只识别键名,值可以是任意的(true、1 或者其他值),只要给出要删除的键名即可。

为展示更多的例子,假设 MongoDB 中已创建集合 Students,且存在两个相关文档,信息如表 3-6 所示。

表 3-6 两个相关文档的信息

stu_name	subjects
李明	C、C++
小刚	MySQL、SQL

例 3-12 将“MySQL”这门课程加到李明的 subjects 中,其语句为

```
db.Students.update({"stu_name":"李明"},{$push:{"subjects":"MySQL"}})
```

例 3-13 将“C”和“C++”这两门课程加到小刚的 subjects 中,其语句为

```
db.Students.update({"stu_name":"小刚"},{$push:{"subjects":{$each:["C","C++"]}}})
```

例 3-14 将“SQL”这门课程加到李明的 subjects 中,其语句为

```
db.Students.update({"stu_name":"李明"},{$addToSet:{"subjects":"SQL"}})
```

例 3-15 将“SQL”这门课程从李明的 subjects 中删除,其语句为

```
db.Students.update({"stu_name":"李明"},{$pull:{"subjects":"SQL"}})
```

例 3-16 将 Students 集合中的 stu_name 键名改为 name,其语句为

```
db.Students.updateMany( {}, { $rename: { "stu_name": "name" } } )
```

2) save 语句

插入或更新已存在的文档。

```
➢ db.COLLECTION.save(<document})
```

如果指定_id 键，则会更新该_id 的数据，根据其键值对对原有键值对重新修改；如果不指定_id 键，则是插入文档，类似 insert()。

3. **删除文档**

1）remove 语句

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

参数说明如下。

- query：设置删除文档的条件；如果没有条件，则删除所有文档。
- justOne：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：可选参数，抛出异常的级别。

例 3-17 将 Books 集合中 book_id 是 B14 的文档删除，其语句为

```
db.Books.remove({"book_id":"B14 "})
```

上述语句可以删除多个匹配的文档。

2）deleteOne 语句

```
➢ db.COLLECTION.deleteOne ()
```

例 3-18 将 Books 集合中 book_id 是 B14 的文档删除，其语句为

```
db.Books.deleteOne({"book_id":"B14 "})
```

上述语句只删除第一个匹配的文档。

3）deleteMany 语句

```
➢ db.COLLECTION.deleteMany ()
```

例 3-19 删除集合 Books 中“铁道出版社”的文档，其语句为

```
db.Books.deleteMany ({"publisher":"铁道出版社"})
```

上述语句将删除所有匹配到的相关文档。

3.5.5 文档查询

1. **查询语句**

MongoDB 的查询语句是 find 语句，用于查询符合条件的文档。

➢ **db.COLLECTION.find(query,projection)**

- query：可选参数，使用查询操作符指定查询条件。
- projection：可选参数，使用投影操作符指定返回的键值对；若没有该参数，则返回文档中所有的键值对。

例 3-20 查询 Books 集合中的所有文档，显示所有的键值对，其语句为

```
db.Books.find()
```

上述查询语句将查询 Books 集合中所有的文档和文档内所有的键值对。

2. 格式化输出

如果要对查询的结果进行格式化输出，就需要使用 pretty() 语句，其语法格式如下。

➢ **db.COLLECTION.find().pretty()**

例 3-21 查询 Books 集合中的所有文档，显示所有的键值对，并将结果格式化，其语句为

```
db.Books.find().pretty()
```

通过 pretty()语句能够将结果输出格式化，各键值对分行输出，方便阅读。

3. 查询表达式

在 find 语句中，可以通过 query 参数设置查询条件，这与 SQL 的 Where 条件语句类似。常用的查询条件如表 3-7 所示。

表 3-7 常用的查询条件

条 件	格 式	范 例	含 义
等于	{<key>:<value>}	find({"name":"SQL 教程"})	name = 'SQL 教程'
小于：$lt	{<key>:{$lt:<value>}}	find({"price":{$lt:50}})	price < 50
小于或等于：$lte	{<key>:{$lte:<value>}}	find({"price":{$lte:50}})	price <= 50
大于：$gt	{<key>:{$gt:<value>}}	find({"price":{$gt:50}})	price > 50
大于或等于：$gte	{<key>:{$gte:<value>}}	find({"price":{$gte:50}})	price >= 50
不等于：ne:	{<key>:{$ne:<value>}}	find({"price":{$ne:50}})	price! = 50

例 3-22 查询 Books 集合中 book_id 是 B07 的文档，并格式化显示，其语句为

```
db.Books.find({"book_id":"B07"}).pretty()
```

例 3-23 查询 Books 集合中价格大于 20 元的文档，并格式化显示，其语句为

```
db.Books.find({"price":{$gt:20}}).pretty()
```

4. 逻辑表达式

在 find 语句中，可以通过 query 参数设置多个查询条件，此时需要用到逻辑表达式。

1）逻辑与

query 参数可以传入多个键，每个键以逗号隔开，这些条件之间的关系是 AND 条件。

```
>find({key1: value1, key2:value2})
```

2）逻辑或

采用 $ or 关键字表达逻辑或。

```
>find({$or: [{key1: value1}, {key2:value2}]})
```

例 3-24 查询 Books 集合中 book_id 是 B07 或 B09 的文档，并格式化显示，其语句为

```
db.Books.find({$or:[{"book_id":"B07"},{"book_id":"B09"}]}).pretty()
```

例 3-25 查询 Books 集合中出版社为“铁道出版社”且价格超过 31 元的图书，并格式化显示，其语句为

```
db.Books.find({"publisher":"铁道出版社","price":{$gt:31}}).pretty()
```

5. 投影查询

使用 projection 投影操作符指定返回的键，如果没有该参数，则返回所有键值对。

例 3-26 查询 Books 集合中价格等于 29 元或等于 30 元的文档，只查询 book_name 和 publisher，并格式化显示，其语句为

```
db.Books.find({$or:[{ "price": 29}, { "price": 30}]}, {_id: 0, book_name: 1,
publisher:1}).pretty()
```

对于需要显示的键，设置为 1 即可，不设置即不显示。_id 主键默认显示，如果不显示，则需要明确设置为 0。

6. 模糊查询

MongoDB 模糊查询使用反斜杠“/”，并结合一些转义字符实现模糊查询。下面通过例子说明具体的使用方法。

例 3-27 查询图书名称中包含“office”的文档，其语句为

```
db.Books.find({"book_name":/office/}).pretty()
```

上述语句中，将查询键值包含“office”的文档，前后的反斜杠通配任意的字符串。

例 3-28 查询图书名称中以“office”开头的文档，其语句为

```
db.Books.find({"book_name":/^office/}).pretty()
```

上述语句的第一个反斜杠后加了转义字符“^”，将失去通配符的含义。

例 3-29 查询图书名称中以“管理”结尾的文档，其语句为

```
db.Books.find({"book_name":/管理$/}).pretty()
```

上述语句的第二个反斜杠前加了转义字符“$”，将失去通配符的含义。

例 3-30 查询图书名称中包含“SQL”的图书，忽略大小写。

```
db.member.find({"book_name":/SQL/i})
```

上述语句的第二个反斜杠后面加了转义字符“i”，表示忽略大小写。

此外，MongoDB 还可以用 $ regex 操作符设置匹配字符串的正则表达式，上面给出的是简化版的正则表达式。

例 3-27 也可以使用 $ regex 操作符进行查询，其语句为

```
db.Books.find({"book_name":{$regex:"office"}}).pretty()
```

例 3-30 也可以使用 $ regex 操作符进行查询，其语句为

```
db.Books.find({"book_name":{$regex:"SQL",$options:"$i"}})
```

7. 限定查询数量

如果要限定查询的文档数量，此时需要使用 limit 方法，该方法接受一个数字参数，限定查询的文档数量。

```
➢ db.COLLECTION.find().limit(Num)
```

例 3-31 查询价格大于 30 元的图书的书名和价格，只显示前 2 个文档，并格式化查询结果，其语句为

```
db.Books.find({"price":{$gt:30}},{_id:0,book_name:1,price:1}).limit(2).pretty
()
```

8. 跳过一些文档

如果需要跳过指定数量的文档，则须使用 skip 方法，该方法接受一个数字参数作为跳过的文档数。

```
➢ db.COLLECTION.find().skip(Num)
```

例 3-32 查询价格大于 30 元的图书的书名和价格，跳过前 2 个文档，并格式化查询结果，其语句为

```
db.Books.find({"price":{$gt:30}},{_id:0,book_name:1,price:1}).skip(2).pretty
()
```

9. 查询结果排序

如果要对查询的结果进行排序，需要使用 sort 方法，该方法根据指定的键值进行排序。

```
➢ db.COLLECTION.find().sort({KEY:1})
```

使用 sort 方法对数据进行排序，可以通过参数指定排序的键，并使用 1 和－1 指定排序的方式，其中 1 为升序排列，而－1 用于降序排列。

例 3-33 查询 Books 集合中的所有文档，显示书名和价格，按价格降序排列，并格式化查询结果，其语句为

```
db.Books.find({},{_id:0,book_name:1,price:1}).sort({price:-1}).pretty()
```

3.5.6 文档聚合

MongoDB 使用文档聚合(aggregate)对文档进行过滤、分组、投影、拆分或者排序等操作，功能非常丰富。

```
➢ db.COLLECTION.aggregate(([{$ pipeline1},{$ pipeline2},{$ pipeline3}…])
```

其中的 $ pipeline1、$ pipeline2、$ pipeline3 等称为管道操作命令，前一个管道命令的输出作为下一个管道命令的输入，通过若干个管道操作可以对结果进一步聚合处理。

下面以表 3-8 中的数据为例，对管道操作命令进行举例说明，假设这些文档已经存储在 Orders 集合中。

表 3-8 集合 Orders 存储的文档

order_id	book_id	book_number	order_date	client_id	comments
O01	B01	500	2003-1-1	C01	A
O02	B05	350	2003-2-28	C02	A
O03	B04	800	2001-10-11	C01	A
O04	B10	1000	2002-7-4	C02	B

1. Match 管道命令

Match 是过滤管道命令，它的作用是按照指定的条件过滤文档，只输出符合条件的文档。