# 文档数据库的基本功能

了解数据模型之后，我们来看看文档数据库提供的基本功能和使用方法。市面上有不少文档数据库系统，如MongoDB、微软的CosmosDB、亚马逊的DocumentDB等。这些系统的基本功能大同小异，但具体的使用接口却不尽相同。由于MongoDB是目前使用最为广泛的文档数据库系统，本节将它作为范例来讲解文档数据库的功能。读者在使用其他文档数据库的时候需查阅相应文档。

数据管理系统最基本的功能是存取数据。文档数据库将数据存取的功能抽象为创建文档（Create）、查询文档（Read）、更新文档（Update）和删除文档（Delete）四类操作，简称为CRUD操作。在介绍这四种类型操作之前，我们先来看看文档数据库的数据组织体系。

## 文档的组织体系

上一节讲到文档数据库将数据组织在一个个的文档中。通常，每个文档描述了现实世界中的一个实体或对象。为了方便文档的管理，文档数据库将相同类型的文档归为一类，放置在同一个**文档集**（Collection）中。此外，属于同一个应用的所有文档集又被放置在同一个**数据库**（Database）中。

换言之，一个文档数据库系统通常会维护多个数据库，每个数据库对应一个应用，为这个应用提供数据管理功能。一个数据库包含若干文档集，每个文档集用于存放某一类对象的数据。比如，关于人的数据存放在一个文档集中，关于书的数据存放在另一个文档集中。一个文档集又包含了若干文档，每个文档描述了一个对象，比如某个人或某本书。文档，文档集和数据库三者之间的关系如下：

$$文档(Document) \in 文档集(Collection) \in 数据库(Database)$$

具备关系型数据库基础知识的读者不难发现，文档数据库中的文档集相当于关系型数据库中的表，文档则相当于表中的元组（表中的行）。

在使用文档数据库系统时，我们首先需要指定一个数据库作为访问对象。在MongoDB中，我们使用下面的指令：

```
use myDB
```

其中，myDB是数据库的名称。选择myDB数据库作为访问对象之后，接下来的操作指令都实施在myDB上。对MongoDB而言，如果myDB数据库不存在，那么该指令会在系统中直接创建一个名为myDB的数据库。

当新建一个数据库时，数据库里面没有任何的文档集和文档。因此，需要在该数据库中创建文档集并插入文档。MongoDB一般不用显示地创建文档集，而是在插入文档的同时隐式地创建文档集，具体指令如下：

```
db.myNewCollection.insertOne( { x: 1 } )
```

上述指令表示的是向文档集myNewCollection中插入一个文档 {x:1}。如果myNewCollection文档集不存在，系统则会自动创建该文档集。指令中的db指代当前正在使用的数据库，即之前指定的myDB数据库。也就是说，新文档集myNewCollection将被放置在myDB中。此外，MongoDB也允许用户使用createCollection指令来显示地创建文档集，指令的使用如下:

```
db.createCollection("myBook", {max : 5000} )
```

该指令要求系统在当前使用的数据库中创建一个名为myBook的文档集，并且设置这个文档集最多只能容纳5000个文档。

# 文档的创建

文档数据库系统允许用户创建任意形式的文档，并将它放置在任意一个文档集中。在MongoDB中，用户可以使用insertOne指令来创建一个新文档，并将该文档插入某个文档集中，比如：

```
db.person.insertOne( {
  "name": "Jason Chang",
  "birthdate": "Jan 20, 2001",
  "gender": "male",
  "address": "20 Yamaha Street",
  "city": "Beijing"
} )
```

上面的指令表示创建一个关于Jason Chang的文档，并将它插入到person文档集中。上一节讲到，每一个文档有一个ID属性，作为文档的唯一标识。如果在插入文档时用户没有显示地设置ID属性值，那么MongoDB会自动生成一个全局唯一的值并赋给该文档的ID属性。

此外，MongoDB也支持使用insertMany指令向一个文档集中一次性插入多个文档，比如：

```
db.book.insertMany( [
  {"title":"An Apple Tree", "author":"Steven Tang"},
  {"title":"Home", "author":"Jing Ba","year":"Sep 1,1997"},
  {"title":"Step by Step", "author":"Newton Wei"}
] )
```

上例表示向文档集book插入三个关于书的文档。与传统的关系型数据库不同，文档数据库允许同时插入不同结构的多个文档。

# 文档的查询

文档数据库支持文档读取操作。假设我们使用下面的命令创建了文档集person，并且往里面插入了关于Jason Chang和Jessis Li的两个文档：

```
db.person.insertMany([
  {
    "name": "Jason Chang",
    "birthdate": {
      "day":20,
      "month":"Jan",
      "year":2001
```

```
    },
    "gender": "male",
    "address": "20 Yamaha Street",
    "city": "Beijing"
  },
  {
    "name": "Jessie Li",
    "birthdate": {
      "day":4,
      "month":"Dec",
      "year":1992
    },
    "gender": "female",
    "address": "200 Sichuan Street",
    "city": "Shanghai"
  }
])
```

MongoDB提供find指令来实现文档查询，比如：

```
db.person.find( {
  "gender": "female",
  "city": "Shanghai"
} )
```

该指令表示在文档集person中查找gender属性为"female"并且city属性为"Shanghai"的文档，实质上就是在文档集person中查询和{"gender":"female", "city": "Shanghai"}相匹配的文档。基于文档匹配这种运算方式，我们可以这样理解：指令x.find(y)的目的是在x中找到y的所有匹配。上述指令的查询结果为关于Jessie Li的文档。

文档匹配是一种基本运算。MongoDB在其上增加了很多灵活性，便于用户表达更广泛的需求。除了上面提到的文档属性的单个值匹配之外，MongoDB还支持属性的多个值匹配、范围匹配等。

MongoDB支持通过关键字**in**来实现属性的多个值匹配。以下指令表示在person文档集中查询city属性为“Shanghai”或“Beijing”的文档：

```
db.person.find(
  { "city": { $in: [ "Shanghai", "Beijing" ] } }
)
```

同样地，多值匹配也可以使用逻辑符号**or**来实现：

```
db.person.find(
 { $or: [ { "city": "Shanghai" }, { "city": "Beijing" } ] }
)
```

此外，MongoDB支持使用关键字**lt,lte,gt,gte**来实现范围匹配，其中lt表示小于，lte表示小于等于，gt表示大于，gte表示大于等于。以下指令表示查询在属性birthdate的子属性year上取值大于2000并且小于2002的文档：

```
db.person.find(
  { "birthdate.year": { $gt: 2000, $lt: 2002} }
)
```

通常，在不做特殊要求的前提下，find指令将找到所有满足条件的文档，并返回这些文档的所有属性。比如，下面的例子中返回了查询结果Jessie Li文档的所有属性和属性值（包括ID属性）。

```
db.person.find(
  { "gender": "female", "city": "Shanghai" }
)
结果为：
{
  "_id": ObjectId("4b2b9f67a1f631733d917a7a"),
  "name": "Jessie Li",
  "birthdate": {
    "day":4,
    "month":"Dec",
    "year":1992
  },
  "gender": "female",
  "address": "200 Sichuan Street",
  "city": "Shanghai"
}
```

但是，很多时候，我们并不需要一个文档的所有属性。find指令允许指定需要返回的属性。例如：

```
db.person.find(
  { "gender": "female", "city": "Shanghai" },
  { "name": 1, "city": 1 }
)
结果为：
{
  "_id": ObjectId("4b2b9f67a1f631733d917a7a"),
  "name": "Jessie Li",
  "city": "Shanghai"
}
```

这里，find指令包含两个参数，第一个参数指定查询条件，第二个参数指定需返回的属性。上例只要求返回查询结果文档中的name和city两个属性。因此，所得到的查询结果就被大大简化了。值得注意的是，ID属性是文档的标识属性，即使不被指定，它也会返回。

MongoDB的查询指令还有很多使用细节，这里将不再赘述。在实际使用时，读者可以查阅MongoDB的使用手册。

## 文档的更新

MongoDB使用update指令实现对文档的更新。其中，updateOne用于更新单个文档，updateMany用于更新多个文档。

update指令包含三个参数，第一个参数指定查询条件，即表示将对什么文档进行更新；第二个参数指定具体的更新操作，即更新文档的哪些属性；第三个参数为可选参数。

```
db.person.updateOne(
   { "name": "Jason Chang" },
   {  $set: { "address": "889 Alibaba Street", "city": "Hangzhou" } }
)
```

updateOne指令首先找到属性name取值为“Jason Chang”的文档，然后将该文档的city属性改为“Hangzhou”，address属性改为“889 Alibaba Street”。updateOne指令只更新找到的第一个文档，也就是说，如果文档集person中存在两个名叫Jason Chang的人，那么只有第一个被找到的Jason Chang文档会被修改。

```
db.person.updateMany(
   { "birthdate.year": {$lt: 2000} },
   {  $set: { "group": "adult" } }
)
```

updateMany指令首先找到在属性birthdate的子属性year上取值小于2000的文档（lt代表“小于”（less than）），即在2000年之前出生的人，然后将这些人的group属性改为“adult”。updateMany指令允许同时更新多个文档，因此凡是在2000年之前出生的人，无论多少，都会被更新。有时，满足查询条件的文档可能没有group属性。遇到这种情况，MongoDB会自动地为这些文档添加group属性，然后设置group的取值为"adult"。

## 文档的删除

MongoDB使用delete指令实现从一个文档集中删除文档。具体指令分为deleteOne和deleteMany。

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

deleteMany指令允许同时删除满足查询条件的所有文档。上例中，deleteMany指令中没有指定查询条件，即没有指定删除对象的条件，它表示将文档集person中的所有文档全部删除。

```
db.person.deleteMany( { "name": "Jason Chang" } )
```

上例中，deleteMany指令首先找到属性name取值为"Jason Chang"的所有文档，然后将查询到的所有文档从文档集person中删除。

```
db.person.deleteOne( { "_id": ObjectId("4b2b9f67a1f631733d917a7a") } )
```

deleteOne指令只允许删除满足查询条件的第一个文档。上例中的指令表示删除某一特定ID的文档。

以上简单地介绍了文档数据库MongoDB的CRUD操作。在使用不同的文档数据库系统时，读者需要查阅对应系统的相关文档，从而才能准确掌握CRUD指令的具体使用方法。

**上一页<<** | **>>下一页**