TheLittle MongoDBBook中文

极客学院出版

前言

本书大部分内容将会专注于 MongoDB 的核心功能。我们会用到 MongoDB 的 shell。因为 shell 不但有助于学习，而且还是个很有用的管理工具。实际代码中你需要用到 MongoDB 驱动。

我们通过学习 MongoDB 的基本工作原理，开始我们的 MongoDB 之旅。当然，这是学习 MongoDB 的核心，它也能帮助我们回答诸如，MongoDB 适用于哪些场景这些更高层次的问题。

预备知识

开始之前，这有六个简单的概念我们需要了解一下。

1. MongoDB中的 `database` 有着和你熟知的"数据库"一样的概念 (对 Oracle 来说就是 schema)。一个 MongoDB 实例中，可以有零个或多个数据库，每个都作为一个高等容器，用于存储数据。
2. 数据库中可以有零个或多个 `collections` (集合)。集合和传统意义上的 table 基本一致，你可以简单的把两者看成是一样的东西。
3. 集合是由零个或多个 `documents` (文档)组成。同样，一个文档可以看成是一 `row` 。
4. 文档是由零个或多个 `fields` (字段)组成。, 没错，它就是 `columns` 。
5. `Indexes` (索引)在 MongoDB 中扮演着和它们在 RDBMS 中一样的角色。
6. `Cursors` (游标)和上面的五个概念都不一样，但是它非常重要，并且经常被忽视，因此我觉得它们值得单独讨论一下。其中最重要的你要理解的一点是，游标是，当你问 MongoDB 拿数据的时候，它会给你返回一个结果集的指针而不是真正的数据，这个指针我们叫它游标，我们可以拿游标做我们想做的任何事情，比如说计数或者跨行之类的，而无需把真正的数据拖下来，在真正的数据上操作。

> 致谢
> 内容撰写：https://github.com/geminiyellow/the-little-mongodb-book

更新日期	更新内容
2015-05-25	The Little MongoDB Book 中文版

目录

1 简介

> 这章那么短不是我的错，MongoDB 就真的很易学。

都说技术在飞速发展。确实，有接连不断的新技术新方法出现。但是，我一直认为，程序员用到的基础技术的发展却是相当缓慢的。你可以好几年不学习但还能混得下去。令人惊讶的其实是成熟技术的被替换速度。就像在一夜之间，那些长期稳定成熟的技术发现它们不再被开发者关注。

最好的例子就是 NoSQL 技术的发展，以及它对稳定的关系型数据库市场的蚕食。看起来就像，昨天网络还是由 RDBMS 们来驱动的，而今天，就冒出五种左右的 NoSQL 解决案已经证明了它们都是值得拥有的。

虽然这些转变看起来都是一夜之间发生的，实际上他们他们可能花了数年的时间来取得公众的认可。最开始是由一小波开发者和公司在推动。解决方案被不断细化，吸取教训，然后一个新技术就这样诞生了，慢慢的后来者也开始了尝试。再次重申，NoSQL 的许多解决方案并不是为了取代传统的存储方案，而是解决一些特殊需求，填补了传统解决方案的一些空白。

说了那么多，我们第一件应该解决的事情是解释一下什么是 NoSQL。它是一个宽松的概念，不同的人有不同的见解。就个人而言，我通常认为它是数据存储系统的一部分。换而言之，NoSQL (重申, 就我而言)，的好处是让你的持久层不需要一个独立的系统。历史上，传统的关系数据库厂商尝试把他们的产品当作一揽子解决方案，NoSQL 倾向于扮演，在特定的工作中充当最好的工具这种角色。因此，你的 NoSQL 架构中还是可以用到关系型数据库，比如说 MySQL，但是可以也可以用 Redis 作为系统中某部分的持久层，或者是用到 Hadoop 来处理大数据。简而言之，NoSQL 就是需要用开放的可代替的意识，使用现有的或者未来的方式和工具来管理你的数据。

你会想知道，MongoDB 是不是适用于这一切。作为一个面向文档数据库，MongoDB 是最通用的 NoSQL 解决案。它可以看成是关系型数据库的代替方案。和关系型数据库一样，它也可以和其他的 NoSQL 解决案搭配在一起更好的工作。MongoDB 有优点也有缺点，我们将会在本书后面的章节中介绍。

如你所见，我们混用了 MongoDB 和 Mongo 两个术语。

第 1 章 开始

本书大部分内容将会专注于 MongoDB 的核心功能。我们会用到 MongoDB 的 shell。因为 shell 不但有助于学习，而且还是个很有用的管理工具。实际代码中你需要用到 MongoDB 驱动。

这也引出了关于 MongoDB 你所需要知道的第一件事: 它的驱动。MongoDB 有各种语言的 官方驱动 (http://docs.mongodb.org/ecosystem/drivers/) 。这些驱动可以认为是和你所熟悉的各种数据库驱动一样的东西。基于这些驱动，开发社区又创建了更多的语言/框架相关库。比如说，NoRM (https://github.com/atheken/NoRM) 是一个 C# 语言库，用 LINQ 实现,而 MongoMapper (https://github.com/jnunemaker/mongomapper) 是一个 Ruby 库，ActiveRecord-friendly。你可以选择直接对 MongoDB 核心进行开发，或选择高级库。之所以要指出，是因为许多新手都觉得迷惑，为什么这里有官方版本和社区版本 – 前者通常关心和 MongoDB 核心的通讯/连接，而后者有更多的语言和框架的实现。

说到这，我希望你可以在 MongoDB 环境中尝试一下我的例子，并且在尝试解决可能遇到的问题。MongoDB 很容易安装和运行，所以让我们花几分钟把所有的东西运行起来。

1. 先打开 官方下载页面 (http://www.mongodb.org/downloads) ，从你选择的操作系统下面的第一行(推荐稳定版本)下载二进制文件。根据开发实际，你可以选择 32位 或者 64位。
2. 解压缩文件 (随便你放哪) 然后进入 `bin` 子目录。现在还不要执行任何命令，只要记住 `mongod` 用来打开服务进程， `mongo` 打开客户端 shell – 大部分时间我们将要使用这两个命令。
3. 在 `bin` 子目录下创建一个文本文件，命名为 `mongodb.config` 。
4. 在 mongodb.config 中添加一行: `dbpath=PATH_TO_WHERE_YOU_WANT_TO_STORE_YOUR_DATABASE_FILES` 。比如，在 Windows 你可以写 `dbpath=c:\mongodb\data` ，在 Linux 可能是 `dbpath=/var/lib/mongodb/data` 。
5. 确保你指定的 `dbpath` 确实存在。
6. 执行 mongod ，带上参数 `--config /path/to/your/mongodb.config` 。

以 Windows 用户为例，如果你解压下载文档到 `c:\mongodb\` ，并且你创建了 `c:\mongodb\data\` ,那么在 `c:\mongodb\bin\mongodb.config` 你要指定 `dbpath=c:\mongodb\data\` 。 然后你可以在 CMD 执行 `mongod` 如下命令行 `c:\mongodb\bin\mongod --config c:\mongodb\bin\mongodb.config` 。

为省心你可以把 `bin` 文件夹路径添加到环境变量 PATH 中，可以简化命令。MacOSX 和 Linux 用户方法几乎一样。唯一需要改变的是路径。

希望你现在已经可以启动 MongoDB 了。如果出现异常，仔细阅读一下异常信息 – 服务器对异常的解释做得非常好。

现在你可以执行 `mongo`（没有 *d*），链接 shell 到你的服务器上了。尝试输入 `db.version()` 来确认所有都正确执行了。你应该能拿到一个已安装的版本号。

2

基础知识

我们通过学习 MongoDB 的基本工作原理，开始我们的 MongoDB 之旅。当然，这是学习 MongoDB 的核心，它也能帮助我们回答诸如，MongoDB 适用于哪些场景这些更高层次的问题。

开始之前，这有六个简单的概念我们需要了解一下。

1. MongoDB中的 `database` 有着和你熟知的"数据库"一样的概念 (对 Oracle 来说就是 schema)。一个 MongoDB 实例中，可以有零个或多个数据库，每个都作为一个高等容器，用于存储数据。
2. 数据库中可以有零个或多个 `collections` (集合)。集合和传统意义上的 `table` 基本一致，你可以简单的把两者看成是一样的东西。
3. 集合是由零个或多个 `documents` (文档)组成。同样，一个文档可以看成是一 `row` 。
4. 文档是由零个或多个 `fields` (字段)组成。, 没错，它就是 `columns` 。
5. `Indexes` (索引)在 MongoDB 中扮演着和它们在 RDBMS 中一样的角色。
6. `Cursors` (游标)和上面的五个概念都不一样，但是它非常重要，并且经常被忽视，因此我觉得它们值得单独讨论一下。其中最重要的你要理解的一点是，游标是，当你问 MongoDB 拿数据的时候，它会给你返回一个结果集的指针而不是真正的数据，这个指针我们叫它游标，我们可以拿游标做我们想做的任何事情，比如说计数或者跨行之类的，而无需把真正的数据拖下来，在真正的数据上操作。

综上，MongoDB 是由包含 `collections` 的 `databases` 组成的。而 `collection` 是由 `documents` 组成。每个 `document` 是由 `fields` 组成。 `Collections` 可以被 `indexed` ，以便提高查找和排序的性能。最后，当我们从 MongoDB 获取数据的时候，我们通过 `cursor` 来操作，读操作会被延迟到需要实际数据的时候才会执行。

那为什么我们需要新的术语(collection vs. table, document vs. row and field vs. column)? 为了让看起来更复杂点? 事实上，虽然这些概念和关系型数据中的概念类似，但是还是有差异的。核心差异在于，关系型数据库是在 `table` 上定义的 `columns` ，而面向文档数据库是在 `document` 上定义的 `fields` 。也就是说，在 `collection` 中的每个 `document` 都可以有它自己独立的 `fields` 。因此，对于 `collection` 来说是个简化了的 `table` ，但是一个 `document` 却比一 `row` 有更多的信息。

虽然这些概念很重要，但是如果现在搞不明白也不要紧。多插几条数据就明白上面说的到底是什么意思了。反正，要点就是，集合不对存储内容严格限制 (所谓的无模式(schema-less))。字段由每个独立的文档进行跟踪处理。这样做的优点和缺点将在下面章节一一讨论。

好了我们开始吧。如果你还没有运行 MongoDB，那么快去运行 `mongod` 服务和开启 mongo shell。shell 用的是 JavaScript。你可以试试一些全局命令，比如 `help` 或者 `exit` 。如果要操作当前数据库，用 `db` ，比如 `db.help()` 或者 `db.stats()` 。如果要操作指定集合，大多数情况下我们会操作集合而不是数据库，用 `db.COLLECTION_NAME` ，比如 `db.unicorns.help()` 或者 `db.unicorns.count()` 。

我们继续，输入 db.help() ，就能拿到一个对 db 能执行的所有的命令的列表。

顺便说一句:因为这是一个 JavaScript shell，如果你输入的命令漏了 () ，你会看到这个命令的源码，而不是执行这个命令。我提一下，是为了避免你执行漏了括号的命令，拿到一个以 function (...){ 开头的返回的时候，觉得神奇不可思议。比如说，如果你输入 db.help (不带括号), 你会看到 help 方法的内部实现。

首先我们用全局的 use 来切换数据库，继续，输入 use learn 。这个数据库实际存在与否完全没有关系。我们在里面生成集合的时候， learn 数据库会自动建起来。现在，我们在一个数据库里面了，你可以开始尝试一下数据库命令，比如 db.getCollectionNames() 。执行之后，你会得到一个空数组 ([])。因为集合是无模式的，我们不需要特地去配置它。我们可以简单的插入一个文档到一个新的集合。像这样，我们用 insert 命令，在文档中插入:

```
db.unicorns.insert({name: 'Aurora',
    gender: 'f', weight: 450})
```

这行命令对集合 unicorns 执行了 insert 命令，并传入一个参数。MongoDB 内部用二进制序列化 JSON 格式，称为 BSON。外部，也就是说我们多数情况应该用 JSON，就像上面的参数一样。然后我们执行 db.getCollectionNames() ，我们将能拿到两个集合: unicorns 和 system.indexes 。在每个数据库中都会有一个 system.indexes 集合，用来保存我们数据的的索引信息。

你现在可以对用 unicorns 执行 find 命令，然后返回文档列表:

```
db.unicorns.find()
```

请注意，除你指定的字段之外，会多出一个 _id 字段。每个文档都会有一个唯一 _id 字段。你可以自己生成一个，或者让 MongoDB 帮你生成一个 ObjectId 类型的。多数情况下，你会乐意让 MongoDB 帮你生成的。默认的 _id 字段是已被索引的 - 这就说明了为什么会有 system.indexes 集合。你可以看看 system.indexes :

```
db.system.indexes.find()
```

你可以看到索引的名字，被索引的数据库和集合，以及在索引中的字段。

现在，回到我们关于数组无模式的讨论中来。往 unicorns 插入一个完全不同的文档，比如:

```
db.unicorns.insert({name: 'Leto',
    gender: 'm',
    home: 'Arrakeen',
    worm: false})
```

然后，再用 find 列出文档。等我们理解再深入一点的时候，将会讨论一下 MongoDB 的有趣行为。到这里，我希望你开始理解，为什么那些传统的术语在这里不适用了。

掌握选择器(Selector)

除了我们介绍过的六个概念，在开始讨论更深入的话题之前，MongoDB 还有一个应该掌握的实用概念:查询选择器。MongoDB 的查询选择器就像 SQL 语句里面的 `where` 一样。因此，你会在对集合的文档做查找，计数，更新，删除的时候用到它。选择器是一个 JSON 对象，最简单的是就是用 `{}` 匹配所有的文档。如果我们想找出所有母独角兽，我们可以用 `{gender:'f'}` 。

开始深入学习选择器之前，让我们先做些准备。首先，把刚才我们插入 `unicorns` 集合的数据删除，通过: `db.unicorns.remove({})` 。现在，再插入一些用来演示的数据 (你不会手打吧):

```
db.unicorns.insert({name: 'Horny',
    dob: new Date(1992,2,13,7,47),
    loves: ['carrot','papaya'],
    weight: 600,
    gender: 'm',
    vampires: 63});
db.unicorns.insert({name: 'Aurora',
    dob: new Date(1991, 0, 24, 13, 0),
    loves: ['carrot', 'grape'],
    weight: 450,
    gender: 'f',
    vampires: 43});
db.unicorns.insert({name: 'Unicrom',
    dob: new Date(1973, 1, 9, 22, 10),
    loves: ['energon', 'redbull'],
    weight: 984,
    gender: 'm',
    vampires: 182});
db.unicorns.insert({name: 'Roooooodles',
    dob: new Date(1979, 7, 18, 18, 44),
    loves: ['apple'],
    weight: 575,
    gender: 'm',
    vampires: 99});
db.unicorns.insert({name: 'Solnara',
    dob: new Date(1985, 6, 4, 2, 1),
    loves:['apple', 'carrot',
        'chocolate'],
    weight:550,
    gender:'f',
    vampires:80});
db.unicorns.insert({name:'Ayna',
```

```
    dob: new Date(1998, 2, 7, 8, 30),
    loves: ['strawberry', 'lemon'],
    weight: 733,
    gender: 'f',
    vampires: 40});
db.unicorns.insert({name:'Kenny',
    dob: new Date(1997, 6, 1, 10, 42),
    loves: ['grape', 'lemon'],
    weight: 690,
    gender: 'm',
    vampires: 39});
db.unicorns.insert({name: 'Raleigh',
    dob: new Date(2005, 4, 3, 0, 57),
    loves: ['apple', 'sugar'],
    weight: 421,
    gender: 'm',
    vampires: 2});
db.unicorns.insert({name: 'Leia',
    dob: new Date(2001, 9, 8, 14, 53),
    loves: ['apple', 'watermelon'],
    weight: 601,
    gender: 'f',
    vampires: 33});
db.unicorns.insert({name: 'Pilot',
    dob: new Date(1997, 2, 1, 5, 3),
    loves: ['apple', 'watermelon'],
    weight: 650,
    gender: 'm',
    vampires: 54});
db.unicorns.insert({name: 'Nimue',
    dob: new Date(1999, 11, 20, 16, 15),
    loves: ['grape', 'carrot'],
    weight: 540,
    gender: 'f'});
db.unicorns.insert({name: 'Dunx',
    dob: new Date(1976, 6, 18, 18, 18),
    loves: ['grape', 'watermelon'],
    weight: 704,
    gender: 'm',
    vampires: 165});
```

现在我们有数据了，我们可以开始来学习掌握选择器了。`{field: value}` 用来查找那些 `field` 的值等于 `value` 的文档。 `{field1: value1, field2: value2}` 相当于 `and` 查询。还有 `$lt`，`$lte`，`$gt`，`$gte` 和 `$ne` 被用来处理小于，小于等于，大于，大于等于，和不等于操作。比如，获取所有体重大于700磅的公独角兽，我们可以这样:

```
db.unicorns.find({gender: 'm',
    weight: {$gt: 700}})
//or (not quite the same thing, but for
//demonstration purposes)
db.unicorns.find({gender: {$ne: 'f'},
    weight: {$gte: 701}})
```

`$exists` 用来匹配字段是否存在，比如:

```
db.unicorns.find({
    vampires: {$exists: false}})
```

会返回一条文档。'$in' 被用来匹配查询文档在我们传入的数组参数中是否存在匹配值，比如:

```
db.unicorns.find({
    loves: {$in:['apple','orange']}})
```

会返回那些喜欢 `apple` 或者 `orange` 的独角兽。

如果我们想要 OR 而不是 AND 来处理选择条件的话，我们可以用 `$or` 操作符，再给它一个我们要匹配的数组:

```
db.unicorns.find({gender: 'f',
    $or: [{loves: 'apple'},
        {weight: {$lt: 500}}]})
```

上面的查询会返回那些喜欢 `apples` 或者 `weigh` 小于500磅的母独角兽。

在我们最后两个例子里面有个非常赞的特性。你应该已经注意到了， `loves` 字段是个数组。MongoDB 允许数组作为基本对象(first class objects)处理。这是个令人难以置信的超赞特性。一旦你开始用它，你都不知道没了它你怎么活下去了。最有趣的是，基于数组的查询变得非常简单: `{loves: 'watermelon'}` 会把文档中 `loves` 中有 `watermelon` 的值全部查询出来。

除了我们介绍的这些，还有更多可用的操作。所有这些都记载在 MongoDB 手册上的 Query Selectors (http://docs.mongodb.org/manual/reference/operator/query/#query-selectors) 这一章。我们介绍的仅仅是那些你学习时所需要用到的，同时也是你最经常用到的操作。

我们已经学习了选择器是如何配合 `find` 命令使用的了。还大致介绍了一下如何配合 `remove` 命令使用， `count` 命令虽然没介绍，不过你肯定知道应该怎么做，而 `update` 命令，之后我们会花多点时间来详细学习它。

MongoDB 为我们的 `_id` 字段生成的 `ObjectId` 可以这样查询:

```
db.unicorns.find(
    {_id: ObjectId("TheObjectId")})
```

小结

我们还没有看到 update ，或是能拿来做更华丽事情的 find 。不过，我们已经安装好 MongoDB 并运行起来了，简略的介绍了一下 insert 和 remove 命令 (完整版也没比我们介绍的多什么)。 我们还介绍了 find 以及了解了 MongoDB selectors 是怎么一回事。 我们起了个很好的头，并为以后的学习奠定了坚实基础。 信不信由你，其实你已经掌握了学习 MongoDB 所必须的大多数知识 – 它真的是易学易用。 我强烈建议你在继续学习之前在本机上多试试多玩玩。 插入不同的文档，可以试试看在不同的集合中，习惯一下使用不同的选择器。试试 find , count 和 remove 。 多试几次之后，你会发现原来看起来那么格格不入的东西，用起来居然水到渠成。

3

更新

在第一章，我们介绍了 CRUD 的四分之三(create, read, update 和 delete) 操作。这章，我们来专门来讨论我们跳过的那个操作: `update` 。 `Update` 有些独特的行为，这是为什么我们把它独立成章。

Update: 覆盖还是 $set

最简单的情况， `update` 有两个参数: 选择器 (`where`) 和需要更新字段的内容。假设 Rooooooodles 长胖了，你会希望我们这样操作:

```
db.unicorns.update({name: 'Rooooooodles'},
  {weight: 590})
```

(如果你已经把 `unicorns` 集合玩坏了，它已经不是原来的数据了的话，再执行一次 `remove` 删除所有数据，然后重新插入第一章中所有的代码。)

现在，如果你查一下被更新了的记录:

```
db.unicorns.find({name: 'Rooooooodles'})
```

你会发现 `update` 的第一个惊喜，没找到任何文档。因为我们指定的第二个参数没有使用任何的更新选项，因此，它 replace 了原始文档。也就是说， `update` 先根据 `name` 找到一个文档，然后用新文档(第二个参数)覆盖替换了整个文档。这和 SQL 的 `update` 命令的完全不一样。在某些情况下，这非常理想，可以用于某些完全动态更新上。但是，如果你只希望改变一个或者几个字段的值的时候，你应该用 MongoDB 的 `$set` 操作。继续，让我们来更新重置这个丢失的数据:

```
db.unicorns.update({weight: 590}, {$set: {
  name: 'Rooooooodles',
  dob: new Date(1979, 7, 18, 18, 44),
  loves: ['apple'],
  gender: 'm',
  vampires: 99}})
```

这里不会覆盖新字段 `weight` 因为我们没有指定它。现在让我们来执行:

```
db.unicorns.find({name: 'Rooooooodles'})
```

我们拿到了期待的结果。因此，在最开始的时候，我们正确的更新 weight 的方式应该是:

```
db.unicorns.update({name: 'Rooooooodles'},
  {$set: {weight: 590}})
```

Update 操作符

除了 $set，我们还可以用其他的更新操作符做些有意思的事情。所有的更新操作都是对字段起作用 – 所以你不用担心整个文档被删掉。比如，$inc 可以用来给一个字段增加一个正/负值。假设说 Pilot 获得了非法的两个 vampire kills 点，我们可以这样修正它:

```
db.unicorns.update({name: 'Pilot'},
  {$inc: {vampires: -2}})
```

假设 Aurora 忽然长牙了，我们可以给她的 loves 字段加一个值，通过 $push 操作:

```
db.unicorns.update({name: 'Aurora'},
  {$push: {loves: 'sugar'}})
```

MongoDB 手册的 Update Operators (http://docs.mongodb.org/manual/reference/operator/update/#update-operators) 这章，可以查到更多可用的更新操作符的信息。

Upserts

用 update 还有一个最大的惊喜，就是它完全支持 upserts。所谓 upsert 更新，即在文档中找到匹配值时更新它，无匹配时向文档插入新值，你可以这样理解。要使用 upsert 我们需要向 update 写入第三个参数 {upsert:true}。

一个最常见的例子是网站点击计数器。如果我们想保存一个实时点击总数，我们得先看看是否在页面上已经有点击记录，然后基于此再决定执行更新或者插入操作。如果省略 upsert 选项(或者设为 false)，执行下面的操作不会带来任何变化:

```
db.hits.update({page: 'unicorns'},
  {$inc: {hits: 1}});
db.hits.find();
```

但是，如果我们加上 upsert 选项，结果会大不同:

```
db.hits.update({page: 'unicorns'},
  {$inc: {hits: 1}}, {upsert:true});
db.hits.find();
```

由于没有找到字段 page 值为 unicorns 的文档，一个新的文档被生成插入。当我们第二次执行这句命令的时候，这个既存的文档将会被更新，且 hits 会被增加到 2。

```
db.hits.update({page: 'unicorns'},
   {$inc: {hits: 1}}, {upsert:true});
db.hits.find();
```

批量 Updates

关于 `update` 的最后一个惊喜，默认的，它只更新单个文档。到目前为止，我们的所有例子，看起来都挺符合逻辑的。但是，如果你执行一些像这样的操作的时候:

```
db.unicorns.update({},
   {$set: {vaccinated: true }});
db.unicorns.find({vaccinated: true});
```

你肯定会希望，你所有的宝贝独角兽都被接种疫苗了。为了达到这个目的， `multi` 选项需要设为 true:

```
db.unicorns.update({},
   {$set: {vaccinated: true }},
   {multi:true});
db.unicorns.find({vaccinated: true});
```

小结

本章中我们介绍了集合的基本 CRUD 操作。我们详细讲解了 `update` 及它的三个有趣的行为。 首先，如果你传 MongoDB 一个文档但是不带更新操作, MongoDB 的 `update` 会默认替换现有文档。因此，你通常要用到 `$set` 操作 (或者其他各种可用的用于修改文档的操作)。 其次， `update` 支持 `upsert` 操作，当你不知道文档是否存在的时候，非常有用。 最后，默认情况下， `update` 只更新第一个匹配文档，因此当你希望更新所有匹配文档时，你要用 `multi` 。

4

掌握查询

在第一章中我们对 find 命令做了一个初步的了解。除了 selectors 以外 find 还有更丰富的功能。我们已经说过， find 返回的结果是一个 cursor 。我们将进一步看看它到底是什么意思。

字段选择

在开始 cursors 的话题之前，你应该知道 find 有第二个可选参数，叫做 "projection"。这个参数是我们要检索或者排除字段的列表。比如，我们可以仅查询返回独角兽的名字而不带别的字段:

```
db.unicorns.find({}, {name: 1});
```

默认的， _id 字段总是会返回的。我们可以通过这样显式的把它从返回结果中排除 {name:1, _id: 0} 。

除了 _id 字段，你不能把检索和排除混合使用。仔细想想，这是有道理的。你只能显式的检索或者排除某些字段。

排序(Ordering)

到目前位置我已经提到好多次， find 返回的是一个游标，它只有在需要的时候才会执行。但是，你在 shell 中看确实到的是 find 被立刻执行了。这只是 shell 的行为。 我们可以通过一个 find 的链式方法，观察到 cursors 的真正行为。我们来看看 sort 。我们指定我们希望排序的字段，以 JSON 方式，其中 1 表示升序 -1 表示降序。比如:

```
//heaviest unicorns first
db.unicorns.find().sort({weight: -1})

//by unicorn name then vampire kills:
db.unicorns.find().sort({name: 1,
  vampires: -1})
```

就像关系型数据库那样，MongoDB 允许对索引进行排序。我们再稍后将详细讨论索引。那，你应该知道的是，MongoDB 对未经索引的字段进行排序是有大小限制的。就是说，如果你试图对一个非常大的没有经过索引的结果集进行排序的话，你会得到个异常。有些人认为这是一个缺点。说实话，我是多希望更多的数据库可以有这种能力去拒绝未经优化的查询。(我不是把每个 MongoDB 的缺点硬说成优点，但是我已经看够了那些缺乏优化的数据库了，我真心希望他们能有一个 strict-mode。)

分页(Paging)

对结果分页可以通过 limit 和 skip 游标方法来实现。比如要获取第二和第三重的独角兽，我们可以这样:

```
db.unicorns.find()
  .sort({weight: -1})
  .limit(2)
  .skip(1)
```

通过 `limit` 和 `sort` 的配合，可以在对非索引字段进行排序时避免引起问题。

计数(Count)

shell 中可以直接对一个集合执行 `count` ，像这样:

```
db.unicorns.count({vampires: {$gt: 50}})
```

实际上， `count` 是一个 `cursor` 的方法，shell 只是简单的提供了一个快捷方式。以不提供快捷方式的方法来执行的时候需要这样(在 shell 中同样可以执行):

```
db.unicorns.find({vampires: {$gt: 50}})
  .count()
```

小结

使用 `find` 和 `cursors` 非常简单。还讲了一些我们后面章节会用到的或是非常特殊情况才用的命令，不过不管怎样，现在，你应该已经非常熟练使用 mongo shell 以及理解 MongoDB 的基本原则了。

5

数据建模

让我们换换思维，对 MongoDB 进行一个更抽象的理解。介绍一些新的术语和一些新的语法是非常容易的。而要接受一个以新的范式来建模，是相当不简单的。事实是，当用新技术进行建模的时候，我们中的许多人还在找什么可用的什么不可用。在这里我们只是开始新的开端，而最终你需要去在实战中练习和学习。

与大多数 NoSQL 数据库相比，面向文档型数据库和关系型数据库很相似 – 至少，在建模上是这样的。但是，不同点非常重要。

No Joins

你需要适应的第一个，也是最根本的区别就是 mongoDB 没有链接(join) 。我不知道 MongoDB 中不支持链接的具体原因，但是我知道链接基本上意味着不可扩展。就是说，一旦你把数据水平扩展，无论如何你都要放弃在客户端(应用服务器)使用链接。事实就是，数据 *有* 关系, 但 MongoDB 不支持链接。

没别的办法，为了在无连接的世界生存下去，我们只能在我们的应用代码中自己实现链接。我们需要进行二次查询 `find` ，把相关数据保存到另一个集合中。我们设置数据和在关系型数据中声明一个外键没什么区别。先不管我们那美丽的 `unicorns` 了，让我们来看看我们的 `employees` 。 首先我们来创建一个雇主 (我提供了一个明确的 `_id` ，这样我们就可以和例子作成一样)

```
db.employees.insert({_id: ObjectId(
  "4d85c7039ab0fd70a117d730"),
  name: 'Leto'})
```

然后让我们加几个工人，把他们的管理者设置为 `Leto` :

```
db.employees.insert({_id: ObjectId(
  "4d85c7039ab0fd70a117d731"),
  name: 'Duncan',
  manager: ObjectId(
  "4d85c7039ab0fd70a117d730")});
db.employees.insert({_id: ObjectId(
  "4d85c7039ab0fd70a117d732"),
  name: 'Moneo',
  manager: ObjectId(
  "4d85c7039ab0fd70a117d730")});
```

(有必要再重复一次， `_id` 可以是任何形式的唯一值。因为你很可能在实际中使用 `ObjectId` ，我们也在这里用它。)

当然，要找出 Leto 的所有工人，只需要执行:

```
db.employees.find({manager: ObjectId(
  "4d85c7039ab0fd70a117d730")})
```

这没什么神奇的。在最坏的情况下，大多数的时间，为弥补无链接所做的仅仅是增加一个额外的查询(可能是被索引的)。

数组和内嵌文档

MongoDB 不支持链接不意味着它没优势。还记得我们说过 MongoDB 支持数组作为文档中的基本对象吗？这在处理多对一(many-to-one)或者多对多(many-to-many)的关系的时候非常方便。举个简单的例子，如果一个工人有两个管理者，我们只需要像这样存一下数组:

```
db.employees.insert({_id: ObjectId(
   "4d85c7039ab0fd70a117d733"),
   name: 'Siona',
   manager: [ObjectId(
   "4d85c7039ab0fd70a117d730"),
   ObjectId(
   "4d85c7039ab0fd70a117d732")] })
```

有趣的是，对于某些文档，`manager` 可以是单个不同的值，而另外一些可以是数组。而我们原来的 `find` 查询依旧可用:

```
db.employees.find({manager: ObjectId(
   "4d85c7039ab0fd70a117d730")})
```

你会很快就发现，数组中的值比多对多链接表(many-to-many join-tables)要容易处理得多。

数组之外，MongoDB 还支持内嵌文档。来试试看向文档插入一个内嵌文档，像这样:

```
db.employees.insert({_id: ObjectId(
   "4d85c7039ab0fd70a117d734"),
   name: 'Ghanima',
   family: {mother: 'Chani',
      father: 'Paul',
      brother: ObjectId(
   "4d85c7039ab0fd70a117d730")}})
```

像你猜的那样，内嵌文档可以用 dot-notation 查询:

```
db.employees.find({
   'family.mother': 'Chani'})
```

我们只简单的介绍一下内嵌文档适用情况，以及你怎么使用它们。

结合两个概念，我们甚至可以内嵌文档数组:

```
db.employees.insert({_id: ObjectId(
   "4d85c7039ab0fd70a117d735"),
   name: 'Chani',
   family: [ {relation:'mother',name: 'Chani'},
      {relation:'father',name: 'Paul'},
      {relation:'brother', name: 'Duncan'}]})
```

反规范化(Denormalization)

另外一个代替链接的方案是对你的数据做反规范化处理(denormalization)。从历史角度看，反规范化处理是为了解决那些对性能敏感的问题，或是需要做快照的数据(比如说审计日志)。但是，随着日益增长的普及的 NoSQL，对链接的支持的日益丧失，反规范化作为规范化建模的一部分变得越来越普遍了。这不意味着，应该对你文档里的每条数据都做冗余处理。而是说，与其对冗余数据心存恐惧，让它影响你的设计决策，不如在建模的时候考虑什么信息应当属于什么文档。

比如说，假设你要写一个论坛应用。传统的方式是通过 `posts` 中的 `userid` 列，来关联一个特定的 `user` 和一篇 `post` 。这样的建模，你没法在显示 `posts` 的时候不查询 (链接到) `users` 。一个代替案是简单的在每篇 `post` 中把 `name` 和 `userid` 一起保存。你可能要用到内嵌文档，比如 `user: {id: ObjectId('Something'), name: 'Leto'}` 。是的，如果你让用户可以更新他们的名字，那么你得对所有的文档都进行更新(一个多重更新)。

适应这种方法不是对任何人都那么简单的。很多情况下这样做甚至是无意义的。不过不要害怕去尝试。它只是在某些情况下不适用而已，但在某些情况下是最好的解决方法。

你的选择是?

在处理一对多(one-to-many)或者多对多(many-to-many)场景的时候，id 数组通常是一个正确的选择。但通常，新人开发者在面对内嵌文档和 "手工" 引用时，左右为难。

首先，你应该知道的是，一个独立文档的大小当前被限制在 16MB 。知道了文档的大小限制，挺宽裕的，对你考虑怎么用它多少有些影响。在这点上，看起来大多数开发者都愿意手工维护数据引用关系。内嵌文档经常被用到，大多数情况下多是很小的数据块，那些总是被和父节点一起拉取的数据块。现实的例子是为每个用户保存一个 `addresses` ，看起来像这样:

```
db.users.insert({name: 'leto',
   email: 'leto@dune.gov',
   addresses: [{street: "229 W. 43rd St",
            city: "New York", state:"NY",zip:"10036"},
           {street: "555 University",
            city: "Palo Alto", state:"CA",zip:"94107"}]})
```

这并不意味着你要低估内嵌文档的能力，或者仅仅把他们当成小技巧。把你的数据模型直接映射到你的对象，这会使得问题更简单，并且通常也不需要用到链接了。尤其是，当你考虑到 MongoDB 允许你对内嵌文档和数组的字段进行查询和索引时，效果特别明显。

大而全还是小而专的集合?

由于对集合没做任何的强制要求，完全可以在系统中用一个混合了各种文档的集合，但这绝对是个非常烂的主意。大多数 MongoDB 系统都采用了和关系型数据库类似的结构，分成几个集合。换而言之，如果在关系型数据库中是一个表，那么在 MongoDB 中会被作成一个集合 (many-to-many join tables being an important exception as well as tables that exist only to enable one to many relationships with simple entities)。

当你把内嵌文档考虑进来的时候，这个话题会变的更有趣。常见的例子就是博客。你是应该分成一个 `posts` 集合和一个 `comments` 集合呢，还是应该每个 `post` 下面嵌入一个 `comments` 数组? 先不考虑那个 16MB 文档大小限制 (*哈姆雷特* 全文也没超过 200KB，所以你的博客是有多人气?)，许多开发者都喜欢把东西划分开来。这样更简洁更明确，给你更好的性能。MongoDB 的灵活架构允许你把这两种方式结合起来，你可以把评论放在独立的集合中，同时在博客帖子下嵌入一小部分评论 (比如说最新评论) ，以便和帖子一同显示。这遵守以下的规则，就是你到想在一次查询中获取到什么内容。

这没有硬性规定(好吧，除了16MB限制)。尝试用不同的方法解决问题，你会知道什么能用什么不能用。

小结

本章目标是提供一些对你在 MongoDB 中数据建模有帮助的指导, 一个新起点，如果愿意你可以这样认为。在一个面向文档系统中建模，和在面向关系世界中建模，是不一样的，但也没多少不同。你能得到更多的灵活性并且只有一个约束，而对于新系统，一切都很完美。你唯一会做错的就是你不去尝试。

6

MongoDB 适用场景

现在你应该有感觉，何时何地把 MongoDB 融入你现有的系统是最棒的了。这有超多的新的类似的存储技术，肯定会让你在选择的时候晕头转向。

对我来说，最重要的教训，跟 MongoDB 无关，是说你不用再依赖单一的解决案来处理你的数据了。毫无疑问，一个单一的解决案有明显的优势，对于许多项目来说 – 或者说大多数 – 单一解决案是一个明智的选择。意思不是说你 *必须* 使用不同的技术，而是说你 *可以*。 只有你自己才知道，引进新技术是否利大于弊。

说了那么多，我希望你到目前为止学到知识让你觉得 MongoDB 是一个通用的解决案。我们已经提到很多次了，面向文档的数据库和关系型数据库有很多方面类似。因此，与其绕开这些相同点，不如我们可以简单的这样认为， MongoDB 是关系型数据库的一个代替案。比如说用 Lucene 作为关系型数据库的全文检索索引的加强，或者用 Redis 作为持久型 key-value 存储，MongoDB 就是用来保存你的数据的。

注意，我没有说用 MongoDB *取代* 关系型数据库，而是 *代替* 案。它能做的有很多工具也能做。有些事情 MongoDB 可以做的更好，另外一些 MongoDB 做得差点。我们来进一步来讨论一下。

无模式(Flexible Schema)

面向文档数据库经常吹嘘的一个好处就是，它不需要一个固定的模式。这使得他们比传统的数据库表要灵活得多。我同意无模式是一个很不错的特性，但不是大多数人说的那样。

人们讲到无模式的时候，好像你就会把一堆乱七八糟的数据统统存起来一样。确实有些领域有些数据用关系型数据库来建模很痛苦，不过我觉得这些都是不常见的特例。无模式是酷，可是大多数情况下你的数据结构还是应当好好设计的。真正需要处理混乱时是不错，比如当你添加一个新功能的时候，不过事实是，大多数情况下，一个空列基本可以解决问题。

对我来说，动态模式的真正好处在于无需很多设置以及可以降低在 OOP 中使用的阻力。这在你使用静态语言的时候尤其明显。我在 C# 和 Ruby 中用过 MongoDB ，差异非常明显。Ruby 的动态特性以及它的流行的 ActiveRecord 实现，已经大幅降低面向对象/关系开发之间差异所带来的阻力。这不是说 MongoDB 和 Ruby 不配，而是是说它们太配了。真的，我觉得许多 Ruby 开发者眼中的的 MongoDB 只是有些许改进而已，而在 C# 或者 Java 开发者眼中，MongoDB 带来的是处理数据交互方式的翻天覆地变化。

假设从驱动开发者角度来看这个问题。你想保存一个对象? 把它串行化成 JSON (严格来说是 BSON, 不过差不多) 然后把它传给 MongoDB。不需要做任何属性映射或者类型映射。这种简单性的好处就这样传递给了你，终端开发者。

写操作(Writes)

MongoDB 可以胜任的一个特殊角色是在日志领域。有两点使得 MongoDB 的写操作非常快。首先，你可以选择发送了写操作命令之后立刻返回，而无须等到操作完成。其次，你可以控制数据持久性的写行为。这些设置，加上，可以定义一个成功的提交，需要在多少台服务器上成功拿到你的数据之后才算成功，并且每个写操作都是可设置, 这就给予你很高的权限用以控制写性能和数据持久性。

除了这些性能因素，日志数据还是这样一种数据集，用无模式集合更有优势。最后，MongoDB 还提供了 受限集合(capped collection) (http://docs.mongodb.org/manual/core/capped-collections/) 。到目前为止，所有我们默认创建的集合都是普通集合。我们可以通过 db.createCollection 命令来创建一个受限集合并标记它的限制:

```
//limit our capped collection to 1 megabyte
db.createCollection('logs', {capped: true,
    size: 1048576})
```

当我们的受限集合到达 1MB 上限的时候，旧文档会被自动清除。另外一种限制可以基于文档个数，而不是大小，用 max 标记。受限集合有一些非常有趣的属性。比如说，你可以更新文档但是你不能改变它的大小。插入顺序是被设置好了的，因此不需要另外提供一个索引来获取基于时间的排序，你可以 "tail" 一个受限集合，就和你在 Unix 中通过 tail -f <filename> 来处理文件一样，获取最新的数据，如果存在数据的话，而不需要重新查询它。

如果想让你的数据 "过期" ，基于时间而不是整个集合的大小，你可以用 TTL 索引 (http://docs.mongodb.org/manual/tutorial/expire-data/) ，所谓 TTL 是 "time-to-live" 的缩写。

持久性(Durability)

在 1.8 之前的版本，MongoDB 不支持单服务器持久性。就是说，如果一个服务器崩溃了，可能会导致数据的丢失或者损坏。解决案是在多服务器上运行 MongoDB 副本 (MongoDB 支持复制)。日志(Journaling)是 1.8 版追加的一个非常重要的功能。从 2.0 版的 MongoDB 开始，日志是默认启动的，该功能允许快速恢复服务器，比如遭遇到了服务器崩溃或者停电的情况。

持久性在这里只是提一下，因为围绕 MongoDB 过去缺乏单服务器持久的问题，人们取得了众多成果。这个话题在以后的 Google 检索中也许还会继续出现。但是关于缺少日志功能这一缺点的信息，都是过时了的。

全文检索(Full Text Search)

真正的全文检索是在最近加入到 MongoDB 中的。它支持十五国语言，支持词形变化(stemming)和干扰字(stop words)。除了原生的 MongoDB 的全文检索支持，如果你需要一个更强大更全面的全文检索引擎的话，你需要另找方案。

事务(Transactions)

MongoDB 不支持事务。这有两个代替案，一个很好用但有限制，另外一个比较麻烦但灵活。

第一个方案，就是各种原子更新操作。只要能解决你的问题，都挺不错。我们已经看过几个简单的了，比如 `$inc` 和 `$set` 。还有像 `findAndModify` 命令，可以更新或删除文档之后，自动返回修改过的文档。

第二个方案，当原子操作不能满足的时候，回到两段提交上来。对于事务，两段提交就好像给链接手工解引用。这是一个和存储无关的解决方案。两段提交实际上在关系型数据库世界中非常常用，用来实现多数据库之间的事务。 MongoDB 网站 有个例子 (http://docs.mongodb.org/manual/tutorial/perform-two-phase-commits/) 演示了最典型的场合 (资金转账)。通常的想法是，把事务的状态保存到实际的原子更新的文档中，然后手工的进行 init-pending-commit/rollback 处理。

MongoDB 支持内嵌文档以及它灵活的 schema 设计，让两步提交没那么痛苦，但是它仍然不是一个好处理，特别是当你刚开始接触它的时候。

数据处理(Data Processing)

在2.2 版本之前的 MongoDB 依赖 MapReduce 来解决大部分数据处理工作。在 2.2 版本，它追加了一个强力的功能，叫做 aggregation framework or pipeline (http://docs.mongodb.org/manual/core/aggregation-pipeline/) ，因此你只要对那些尚未支持管道的，需要使用复杂方法的，不常见的聚合使用 MapReduce。下一章我们将看看聚合管道和 MapReduce 的细节。现在，你可以把他们想象成功能强大的，用不同方法实现的 `group by` (打个比方)。对于非常大的数据的处理，你可能要用到其他的工具，比如 Hadoop。值得庆幸的是，这两个系统是相辅相成的，这里有个 MongoDB connector for Hadoop (http://docs.mongodb.org/ecosystem/tools/hadoop/) 。

当然，关系型数据库也不擅长并行数据处理。MongoDB 有计划在未来的版本中，改善增加处理大数据集的能力。

地理空间查询(Geospatial)

一个很强大的功能就是 MongoDB 支持 geospatial 索引 (http://docs.mongodb.org/manual/applications/geospatial-indexes/) 。这允许你保存 geoJSON 或者 x 和 y 坐标到文档，并查询文档，用如 `$near` 来获取坐标集，或者 `$within` 来获取一个矩形或圆中的点。这个特性最好通过一些可视化例子来演示，所以如果你想学更多的话，可以试试看 5 minute geospatial interactive tutorial (http://mongly.openmymind.net/geo/index) 。

工具和成熟度

你应该已经知道这个问题的答案了，MongoDB 确实比大多数的关系型数据要年轻很多。这个问题确实是你应当考虑的，但是到底有多重要，这取决于你要做什么，怎么做。不管怎么说，一个好的评估，不可能忽略 MongoDB 年轻这一事实，而可用的工具也不是很好 (虽然成熟的关系型数据库工具有些也非常渣!)。举个例子，它缺乏对十进制浮点数的支持，在处理货币的系统来说，明显是一个问题 (尽管也不是致命的) 。

积极的一方面，它为大多数语言提供了驱动，协议现代而简约，开发速度相当快。MongoDB 被众多公司用到了生产环境中，虽然有所担心，但经过验证后，担心很快就变成了过去。

小结

本章要说的是，MongoDB，大多数情况下，可以取代关系型数据库。它更简单更直接；更快速并且通常对应用开发者的约束更少。不过缺乏事务支持也许值得慎重考虑。当人们说起 *MongoDB 在新的数据库阵营中到底处在什么位置?* 时，答案很简单: **中庸**(*2*)。

数据聚合

聚合管道(Aggregation Pipeline)

聚合管道提供了一种方法用于转换整合文档到集合。你可以通过管道来传递文档，就像 Unix 的 "pipe" 一样，将一个命令的输出传递到另第二个，第三个，等等。

最简单的聚合，应该是你在 SQL 中早已熟悉的 `group by` 操作。我们已经看过 `count()` 方法，那么假设我们怎么才能知道有多少匹公独角兽，有多少匹母独角兽呢?

```
db.unicorns.aggregate([{$group:{_id:'$gender',
   total: {$sum:1}}}])
```

在 shell 中，我们有 `aggregate` 辅助类，用来执行数组的管道操作。对于简单的对某物进行分组计数，我们只需要简单的调用 `$group` 。这和 SQL 中的 `GROUP BY` 完全一致，我们用来创建一个新的文档，以 `_id` 字段表示我们以什么来分组(在这里是以 `gender`)，另外的字段通常被分配为聚合的结果，在这里，我们对匹配某一性别的各文档使用了 `$sum` 1 。你应该注意到了 `_id` 字段被分配为 `'$gender'` 而不是 `'gender'` – 字段前面的 `'$'` 表示，该字段将会被输入的文档中的有同样名字的值所代替，一个占位符。

我们还可以用其他什么管道操作呢? 在 `$group` 之前(之后也很常用)的一个是 `$match` – 这和 `find` 方法完全一样，允许我们获取文档中某个匹配的子集，或者在我们的结果中对文档进行筛选。

```
db.unicorns.aggregate([{$match: {weight:{$lt:600}}},
   {$group: {_id:'$gender', total:{$sum:1},
     avgVamp:{$avg:'$vampires'}}},
   {$sort:{avgVamp:-1}} ])
```

这里我们介绍另外一个管道操作 `$sort` ，作用和你想的完全一致，还有和它一起用的 `$skip` 和 `$limit` 。以及用 `$group` 操作 `$avg` 。

MongoDB 数组非常强大，并且他们不会阻止我们往保存中的数组中写入内容。我们需要可以 "flatten" 他们以便对所有的东西进行计数:

```
db.unicorns.aggregate([{$unwind:'$loves'},
    {$group: {_id:'$loves', total:{$sum:1},
    unicorns:{$addToSet:'$name'}}},
     {$sort:{total:-1}},
     {$limit:1} ])
```

这里我们可以找出独角兽最喜欢吃的食物，以及拿到独角兽们喜欢吃的食物名单。 `$sort` 和 `$limit` 的组合能让你拿到 "top N" 这种查询的结果。

还有另外一个强大的管道操作叫做 `$project` (http://docs.mongodb.org/manual/reference/operator/aggregation/project/#pipe._S_project) (类似于 `find`)，不但允许你拿到指定字段，还可以根据现存字段进行创建或计算一个新字段。比如，可以用数学操作，在做平均运算之前，对几个字段进行加法运算，或者你可以用字符串操作创建一个新的字段，用于拼接现有字段。

这只是用聚合所能做到的众多功能中的皮毛， 2.6 的聚合拥有了更强大的力量，比如聚合命令可以返回结果集的游标(我们已经在第一章学过了) 或者可以将结果写到另外一个新集合中，通过 `$out` 管道操作。你可以从 MongoDB 手册 (http://docs.mongodb.org/manual/core/aggregation-pipeline/) 得到关于管道操作和表达式操作更多的例子。

MapReduce

MapReduce 分两步进行数据处理。首先是 map，然后 reduce。在 map 步骤中，转换输入文档和输出一个 key=>value 对(key 和/或 value 可以很复杂)。然后, key/value 对以 key 进行分组，有同样的 key 的 value 会被收入一个数组中。在 reduce 步骤中，获取 key 和该 key 的 value 的数组，生成最终结果。map 和 reduce 方法用 JavaScript 来编写。

在 MongoDB 中我们对一个集合使用 `mapReduce` 命令。 `mapReduce` 执行 map 方法， reduce 方法和 output 指令。在我们的 shell 中，我们可以创建输入一个 JavaScript 方法。许多库中，支持字符串方法 (有点丑)。第三个参数设置一个附加参数，比如说我们可以过滤，排序和限制那些我们想要分析的文档。我们也可以提供一个 `finalize` 方法来处理 `reduce` 步骤之后的结果。

在你的大多数聚合中，也许无需用到 MapReduce , 但如果需要，你可以读到更多关于它的内容，从 我的 blog (http://openmymind.net/2011/1/20/Understanding-Map-Reduce/) 和 MongoDB 手册 (http://docs.mongodb.org/manual/core/map-reduce/) 。

小结

在这章中我们介绍了 MongoDB 的 聚合功能(aggregation capabilities) (http://docs.mongodb.org/manual/aggregation/) 。一旦你理解了聚合管道(Aggregation Pipeline)的构造，它还是相对容易编写的，并且它是一个聚合数据的强有力工具。 MapReduce 更难理解一点，不过它强力无边，就像你用 JavaScript 写的代码一样。

8

性能和工具

在这章中，我们来讲几个关于性能的话题，以及在 MongoDB 开发中用到的一些工具。我们不会深入其中的一个话题，不过我们会指出每个话题中最重要的方面。

索引(Index)

首先我们要介绍一个特殊的集合 `system.indexes` ，它保存了我们数据库中所有的索引信息。索引的作用在 MongoDB 中和关系型数据库基本一致: 帮助改善查询和排序的性能。创建索引用 `ensureIndex` :

```
// where "name" is the field name
db.unicorns.ensureIndex({name: 1});
```

删除索引用 `dropIndex` :

```
db.unicorns.dropIndex({name: 1});
```

可以创建唯一索引，这需要把第二个参数 `unique` 设置为 `true` :

```
db.unicorns.ensureIndex({name: 1},
  {unique: true});
```

索引可以内嵌到字段中 (再说一次，用点号) 和任何数组字段。我们可以这样创建复合索引:

```
db.unicorns.ensureIndex({name: 1,
  vampires: -1});
```

索引的顺序 (1 升序, -1 降序) 对单键索引不起任何影响，但它会在使用复合索引的时候有所不同，比如你用不止一个索引来进行排序的时候。

阅读 indexes page (http://docs.mongodb.org/manual/indexes/) 获取更多关于索引的信息。

Explain

需要检查你的查询是否用到了索引，你可以通过 `explain` 方法:

```
db.unicorns.find().explain()
```

输出告诉我们，我们用的是 `BasicCursor` (意思是没索引), 12 个对象被扫描，用了多少时间，什么索引，如果有索引，还会有其他有用信息。

如果我们改变查询索引语句，查询一个有索引的字段，我们可以看到 `BtreeCursor` 作为索引被用到填充请求中去:

```
db.unicorns.find({name: 'Pilot'}).explain()
```

复制(Replication)

MongoDB 的复制在某些方面和关系型数据库的复制类似。所有的生产部署应该都是副本集，理想情况下，三个或者多个服务器都保持相同的数据。写操作被发送到单个服务器，也即主服务器，然后从它异步复制到所有的从服务器上。你可以控制是否允许从服务器上进行读操作，这可以让一些特定的查询从主服务器中分离出来，当然，存在读取到旧数据的风险。如果主服务器异常关闭，从服务中的一个将会自动晋升为新的主服务器继续工作。另外，MongoDB 的复制不在本书的讨论范围之内。

分片(Sharding)

MongoDB 支持自动分片。分片是实现数据扩展的一种方法，依靠在跨服务器或者集群上进行数据分区来实现。一个最简单的实现是把所有的用户数据，按照名字首字母 A-M 放在服务器 1 ，然后剩下的放在服务器 2。谢天谢地，MongoDB 的拆分能力远比这种分法要强。分片不在本书的讨论范围之内，不过你应当有分片的概念，并且，当你的需求增长超过了使用单一副本集的时候，你应该考虑它。

尽管复制有时候可以提高性能(通过将长时间查询隔离到从服务器，或者降低某些类型的查询的延迟),它的主要目的是维护高可用性。分片是扩展 MongoDB 集群的主要方法。把复制和分片结合起来实现可扩展和高可用性是禁术。

状态(Stats)

你可以通过 `db.stats()` 查询数据库的状态。基本上都是关于数据库大小的信息。你还可以查询集合的状态，比如说 `unicorns` 集合，可以输入 `db.unicorns.stats()` 。基本上都是关于集合大小的信息，以及集合的索引信息。

分析器(Profiler)

你可以这样执行 MongoDB profiler :

```
db.setProfilingLevel(2);
```

启动之后，我们可以执行一个命令:

```
db.unicorns.find({weight: {$gt: 600}});
```

然后检查 profiler:

```
db.system.profile.find()
```

输出会告诉我们:什么时候执行了什么，有多少文档被扫描，有多少数据被返回。

你要停止 profiler 只需要再调用一次 `setProfilingLevel` ，不过这次参数是 `0` 。指定 `1` 作为第一个参数，将会过滤统计超过 100 milliseconds 的任务. 100 milliseconds 是默认的阈值，你可以在第二个参数中，指定不同的阈值时间，以 milliseconds 为单位:

```
//profile anything that takes
//more than 1 second
db.setProfilingLevel(1, 1000);
```

备份和还原

在 MongoDB 的 `bin` 目录下有一个可执行文件 `mongodump` 。简单执行 `mongodump` 会链接到 localhost 并备份你所有的数据库到 `dump` 子目录。你可以用 `mongodump --help` 查看更多执行参数。常用的参数有 `--db DBNAME` 备份指定数据库和 `--collection COLLECTIONNAME` 备份指定集合。你可以用 `mongorestore` 可执行文件，同样在 `bin` 目录下，还原之前的备份。同样， `--db` 和 `--collection` 可以指定还原的数据库和/或集合。 `mongodump` 和 `mongorestore` 使用 BSON，这是 MongoDB 的原生格式。

比如，来备份我们的 `learn` 数据库导 `backup` 文件夹，我们需要执行(在控制台或者终端中执行该命令，而不是在 mongo shell 中):

```
mongodump --db learn --out backup
```

如果只还原 `unicorns` 集合，我们可以这样做:

```
mongorestore --db learn --collection unicorns \
  backup/learn/unicorns.bson
```

值得一提的是， `mongoexport` 和 `mongoimport` 是另外两个可执行文件，用于导出和从 JSON/CSV 格式文件导入数据。比如说，我们可以像这样导出一个 JSON:

```
mongoexport --db learn --collection unicorns
```

CSV 格式是这样:

```
mongoexport --db learn \
  --collection unicorns \
  --csv --fields name,weight,vampires
```

注意 `mongoexport` 和 `mongoimport` 不一定能正确代表数据。真实的备份中，只能使用 `mongodump` 和 `mongorestore` 。你可以从 MongoDB 手册中读到更多的 备份须知 (http://docs.mongodb.org/manual/core/backups/) 。

小结

在这章中我们介绍了 MongoDB 的各种命令，工具和性能细节。我们没有涉及所有的东西，不过我们已经把常用的都看了一遍。MongoDB 的索引和关系型数据库中的索引非常类似，其他一些工具也一样。不过，在 MongoDB 中，这些更易于使用。

第 8 章 总结

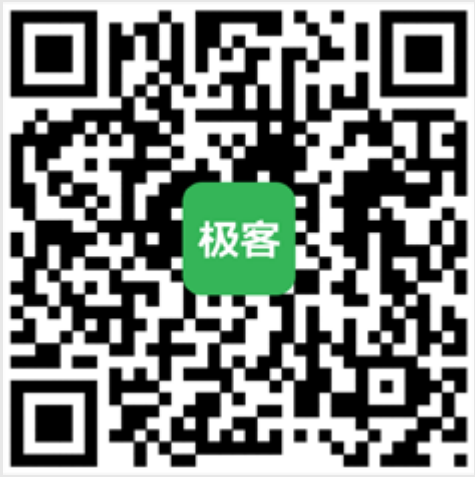
你现在应该有足够的能力开始在真实项目中使用 MongoDB 了。虽然 MongoDB 远不止我们学到的这些内容，但是你要作的下一步是，把学到的知识融会贯通，熟悉我们需要用到的功能。MongoDB website (http://www.mongodb.org/) 有许多有用的信息。官网的 MongoDB user group (http://groups.google.com/group/mongodb-user) 是个问问题的好地方。

NoSQL 不光是为需求而生，它同时还是不断尝试创新的成果。不得不承认，我们的领域是不断前行的。如果我们不尝试，一旦失败，我们就绝不会取得成功。就是这样的，我认为，这是让你在职业生涯一路走好的方法。

极客学院

jikexueyuan.com

中国最大的IT职业在线教育平台



更多信息请访问

http://wiki.jikexueyuan.com/project/the-little-mongodb-book/