

MongoDB 从入门到精通

— For Developers, DBAs, Data Architects —



扫码试看/订阅

《MongoDB 高手课》视频课程

内容综述

课程概览 | 学习目标

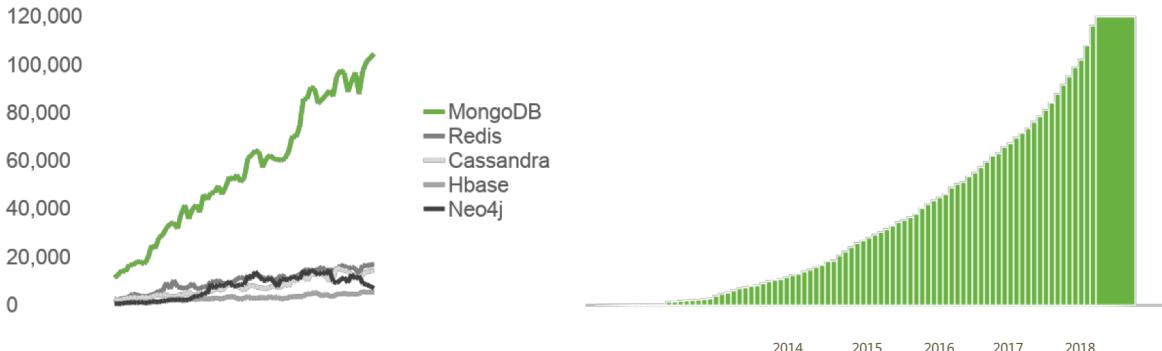
内容综述

MongoDB 再入门	从熟练到精通的开发之路	片集群与高级运维之道	集中实战掌握框架构建之法
课程介绍	文档模型设计理论	分片集群机制及原理	MongoDB 应用场景及选型
内容综述	文档模型设计方法	分片集群设计要领	MongoDB 典型案例
认识 MongoDB	设计模式集锦（一）	实验：分片集群搭建及扩容	关系型数据库替换
MongoDB 特点	设计模式集锦（二）	MongoDB 监控最佳实践	Oracle 替换实战
实验：安装 MongoDB	事务开发：读操作事务	MongoDB 备份与恢复	MongoDB 与数据中台
MongoDB 基本操作	事务开发：写操作事务	MongoDB 安全架构与加固	MongoDB 数据中台实战
实验：Hello World 程序开发	事务开发：多文档事务	MongoDB 索引机制及原理	Mongo + Spark 实时大数据
聚合查询	Change Stream	MongoDB 性能问题排查及优化	Mongo + Spark 实战
实验：聚合操作	MongoDB + 微服务	MongoDB 上线及升级	数据分析 SQL 套接件
MongoDB 复制集机制及原理	MongoDB 版本升级	高级集群设计：两地三中心	MongoDB 可视化
实验：搭建 MongoDB 复制集	MongoDB 开发最佳实践	高级集群设计：全球多写	MongoDB ETL 工具
MongoDB 系列软件及工具		实验：搭建两地三中心集群	

第1.3讲 认识文档数据库 MongoDB

简介 | 历史变迁

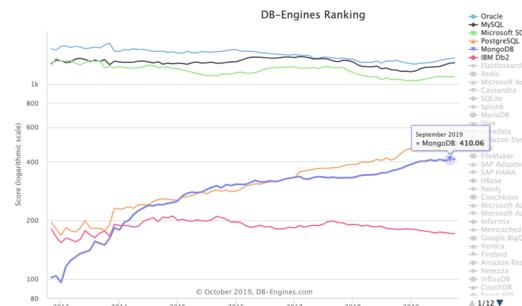
MongoDB：重新定义 OLTP 数据库



Stack Overflow
使用热度增长最快速

全球6000多万下载量
中国地区占第一

Oracle
MySQL
SQLServer
PostgreSQL
MongoDB



DB-Engines 数据库排名第五
唯一的 NOSQL

关于 MongoDB

什么是 MongoDB?	一个以 JSON 为数据模型的文档数据库。
为什么叫文档数据库?	文档来自于 “JSON Document” , 并非我们一般理解的 PDF, WORD 文档。
谁开发 MongoDB?	上市公司 MongoDB Inc. , 总部位于美国纽约。
主要用途	应用数据库, 类似于 Oracle, MySQL 海量数据处理, 数据平台。
主要特点	建模为可选 JSON 数据模型比较适合开发者 横向扩展可以支撑很大数据量和并发
MongoDB 是免费的吗?	MongoDB 有两个发布版本: 社区版和企业版。 社区版是基于 SSPL, 一种和 AGPL 基本类似的开源协议 。 企业版是基于商业协议, 需付费使用。

MongoDB 版本变迁



MongoDB vs. 关系型数据库

	MongoDB	RDBMS
数据模型	文档模型	关系模型
数据库类型	OLTP	OLTP
CRUD 操作	MQL/SQL	SQL
高可用	复制集	集群模式
横向扩展能力	通过原生分片完善支持	数据分区或者应用侵入式
索引支持	B-树、全文索引、地理位置索引、多键(multikey)索引、TTL 索引	B 树
开发难度	容易	困难
数据容量	没有理论上限	千万、亿
扩展方式	垂直扩展+水平扩展	垂直扩展

第1.4讲 MongoDB 特色及优势

开发效率 | 自然模型 | 横向扩展

本章学习内容及目标

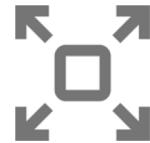
内容大纲	学习目标
文档模型的面向对象特点	理解文档模型的优势
文档模型的灵活性	了解 MongoDB 的高可用和扩展能力
文档模型的快速开发特点	
集群特性	

MongoDB 优势：面向开发者的易用+高效数据库



简单直观:

以自然的方式来建模，以直观的方式
来与数据库交互



结构灵活:

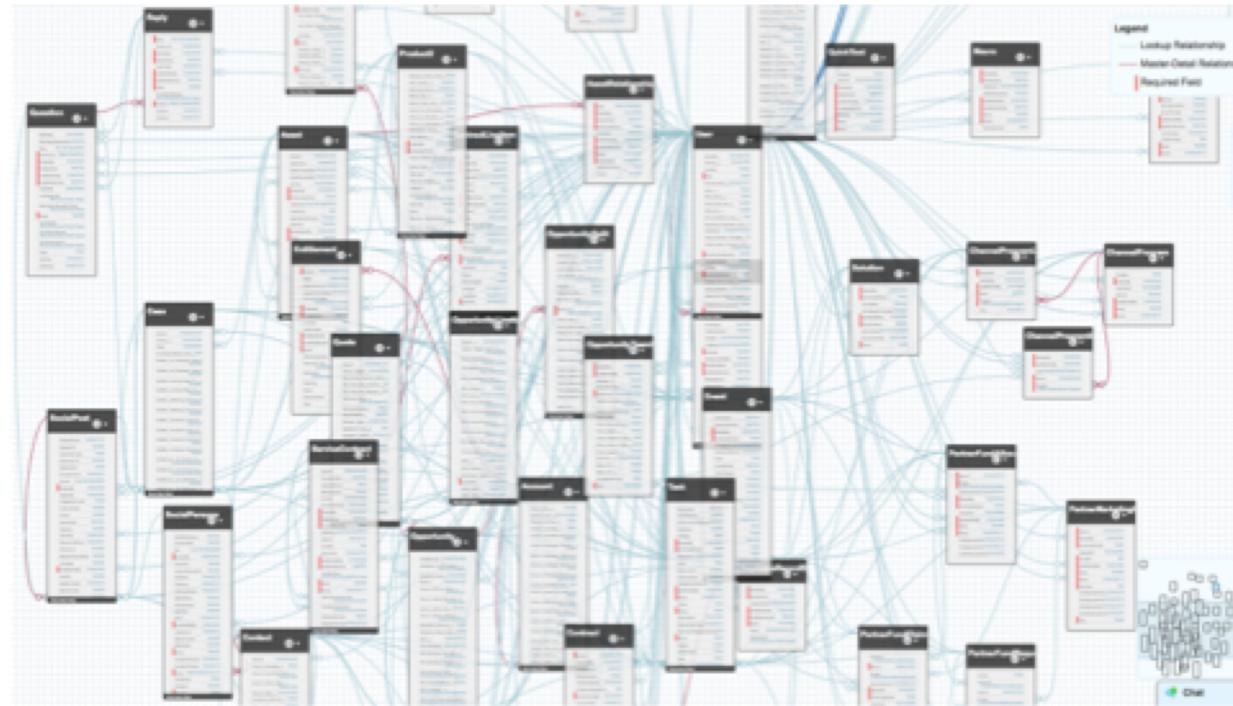
弹性模式从容响应需求的频繁变化



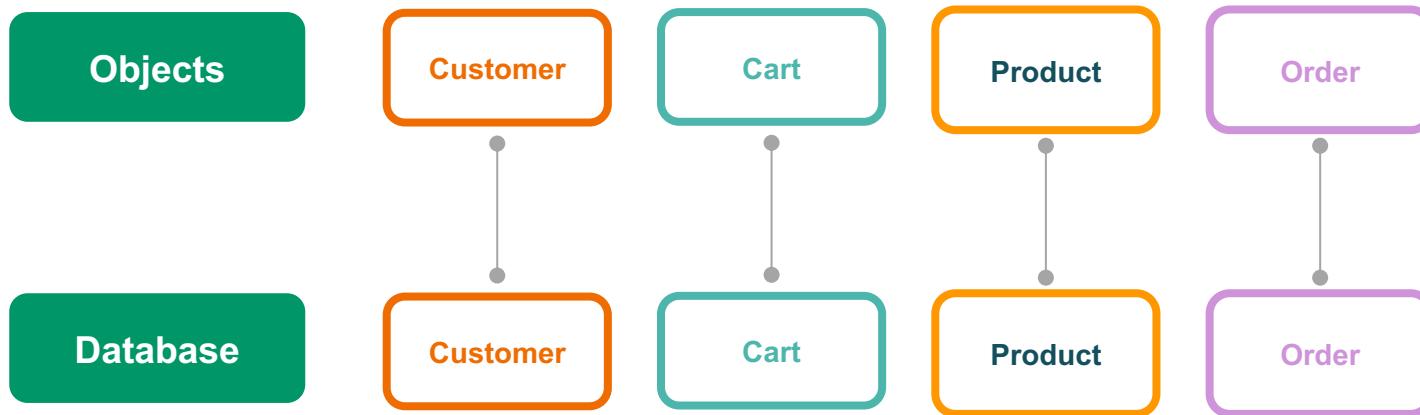
快速开发:

做更多的事，写更少的代码

从错综复杂的关系模型



到一目了然的对象模型



灵活：快速响应业务变化

```
_id: 12345678
> name: Object
> address: Array
> phone: Array
  email: "john.doe@mongodb.com"
  dob: 1966-07-30 01:00:00.000
< interests: Array
  0: "Cycling"
  1: "IoT"
```

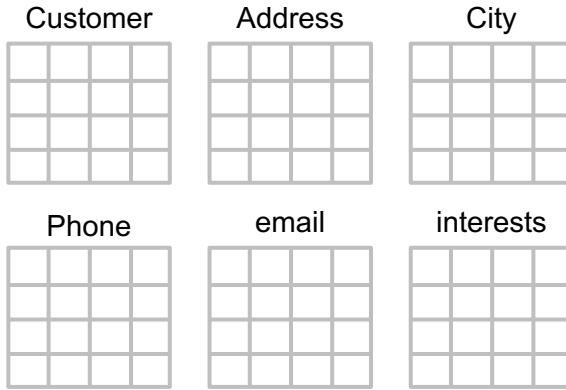


```
_id: 12345678
> name: Object
> address: Array
> phone: Array
  email: "john.doe@mongodb.com"
< social: Array
  < 0: Object
    twitter: "@mongodb"
  < 1: Object
    instagram: "@mongodb"
  annualSpend: 1500
  dob: 1966-07-30 01:00:00.000
< interests: Array
  0: "Cycling"
  1: "IoT"
```

可动态增加新字段

- **多形性**: 同一个集合中可以包含不同字段（类型）的文档对象
- **动态性**: 线上修改数据模式，修改是应用与数据库均无须下线
- **数据治理**: 支持使用 JSON Schema 来规范数据模式。在保证模式灵活动态的前提下，提供数据治理能力

快速：最简单快速的开发方式



```
_id: 12345678
> name: Object
> address: Array
> phone: Array
  email: "john.doe@mongodb.com"
  dob: 1966-07-30 01:00:00.000
~ interests: Array
  0: "Cycling"
  1: "IoT"
```

- JSON 模型之快速特性：

- 数据库引擎只需要在一个存储区读写
- 反范式、无关联的组织极大优化查询速度
- 程序 API 自然，开发快速

SQL: 插入一个客户相关数据

```

import mysql.connector
from mysql.connector import errorcode

def addUser(connection, user):
    cursor = connection.cursor()

    customerInsert = (
        "INSERT INTO customer (first_name, last_name, email, "
        "dob, annual_spend) VALUES "
        "(%(first)s, %(last)s, %(email)s, %(dob)s, %"
        "(spend)s)")

    customerData = {
        'first': user['name']['first'],
        'last': user['name']['second'],
        'email': user['email'],
        'dob': user['dob'],
        'spend': user['annualSpend']
    }

    cursor.execute(customerInsert, customerData)
    customerId = cursor.lastrowid

    cityQuery = ("SELECT city_id FROM city WHERE "
        "city = %(city)s")
    for address in user['address']:
        cursor.execute(cityQuery, {'city': address['city']})
        city_id = cursor.fetchone()[0]

        addressInsert = (
            "INSERT INTO address (address, address2, district,"
            "city_id, postal_code, customer_id, location) "
            "VALUES (%(add)s, %(add2)s, %(dist)s, %(city)s, "
            "%(post)s, %(cust)s, %(loc)s)")

        addressData = {
            'first': user['name']['first'],
            'last': user['name']['second'],
            'email': user['email'],
            'dob': user['dob'],
            'spend': user['annualSpend']
        }

        cursor.execute(addressInsert, addressData)
        addressId = cursor.lastrowid

        addressInsert = (
            "INSERT INTO address (address, address2, district,"
            "city_id, postal_code, customer_id, location) "
            "VALUES (%(add)s, %(add2)s, %(dist)s, %(city)s, "
            "%(post)s, %(cust)s, %(loc)s)")

        addressData = {
            'add': address['number'],
            'add2': address['street'],
            'dist': address['state'],
            'city': city_id,
            'post': address['postalCode'],
            'cust': customerId,
            'loc': address['location']
        }

        cursor.execute(addressInsert, addressData)

        topicQuery = ("SELECT topics_id FROM topics "
            "WHERE subject = %(subj)s")
        interestInsert = (
            "INSERT into interests (topic_id, customer_id) "
            "VALUES (%(topic)s, %(cust)s)")

        for interest in user['interests']:
            topicId = 0
            topicData = {
                'subj': interest['interest']
            }

            cursor.execute(topicQuery, {'subj': interest['interest']})
            topicId = cursor.fetchone()[0]

            cursor.execute(interestInsert, topicData)

            phoneInsert = (
                "INSERT INTO `phone numbers` (customer_id, "
                "phone_number, `Phone number_type`) "
                "VALUES (%(cust)s, %(num)s, %(type)s)")

            for phoneNumber in user['phone']:
                phoneData = {
                    'cust': customerId,
                    'num': phoneNumber['number'],
                    'type': phoneNumber['location']
                }

                cursor.execute(phoneInsert, phoneData)

connection.commit()
cursor.close()
return customerId

```

快速: MongoDB 只需要两行代码

```

import mysql.connector
from mysql.connector import errorcode

def addUser(connection, user):
    cursor = connection.cursor()

    customerInsert = (
        "INSERT INTO customer (first_name, last_name, email, "
        "DOB, annual_spend) VALUES "
        "(%(first)s, %(last)s, %(email)s, %(dob)s, %"
        "(spend)s)")

    customerData = {
        'first': user['name']['first'],
        'last': user['name']['second'],
        'email': user['email'],
        'dob': user['dob'],
        'spend': user['annualSpend']
    }

    cursor.execute(customerInsert, customerData)
    customerId = cursor.lastrowid

    addressData = {
        'add': address['number'],
        'add2': address['street'],
        'dist': address['state'],
        'city': city_id,
        'post': address['postalCode'],
        'cust': customerId,
        'loc': address['location']
    }

    cursor.execute(addressInsert, addressData)

    cursor.execute(topicInsert, topicData)
    topicId = cursor.lastrowid

    else:
        topicId = row[0]

    interestData = {

    Insert, interestData)

    for phoneNumber in user['phone']:
        phoneData = {
            'cust': customerId,
            'num': phoneNumber['number'],
            'type': phoneNumber['location']
        }

        cursor.execute(phoneInsert, phoneData)

    connection.commit()
    cursor.close()
    return customerId
}

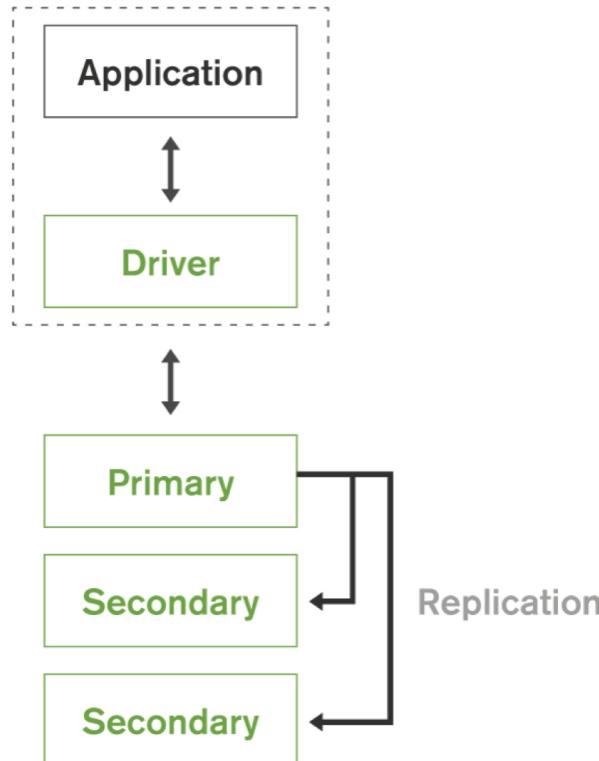
```

```

def addUser(database, user):
    return database.customers.insert_one(user).inserted_id

```

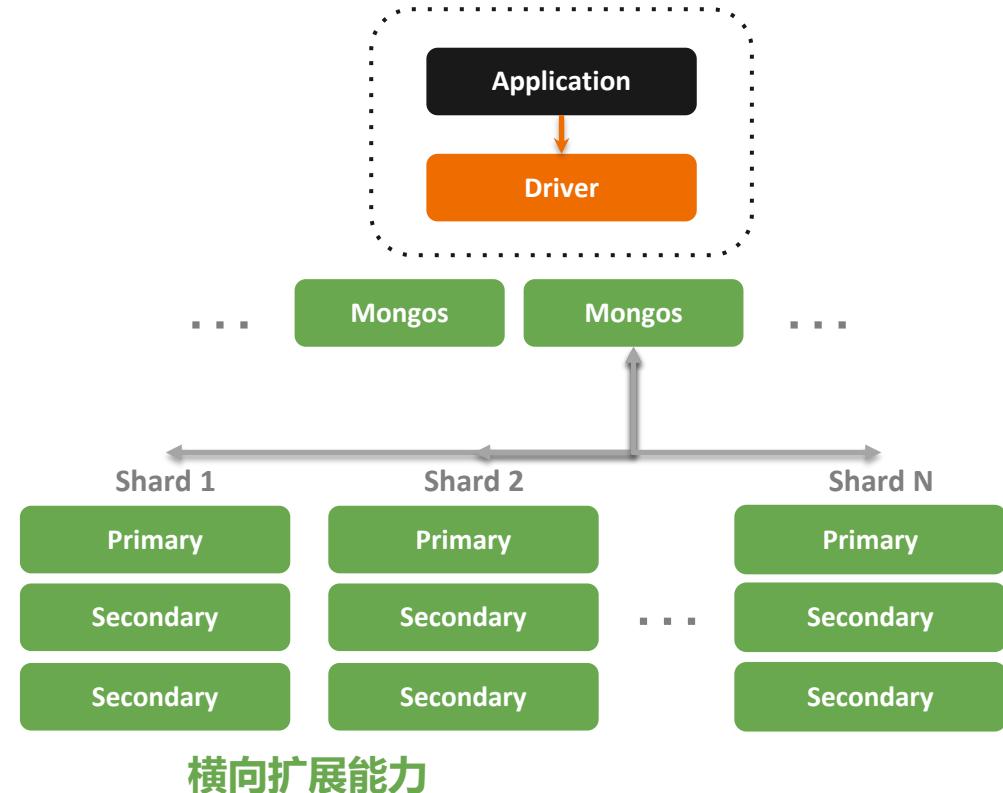
MongoDB 优势：原生的高可用和横向扩展能力



- Replica Set - 2 to 50 个成员
- 自恢复
- 多中心容灾能力
- 滚动服务 - 最小化服务终端

MongoDB 优势：横向扩展能力

- 需要的时候无缝扩展
- 应用全透明
- 多种数据分布策略
- 轻松支持 TB - PB 数量级



MongoDB 技术优势总结

- JSON 结构和对象模型接近，开发代码量低
- JSON 的动态模型意味着更容易响应新的业务需求
- 复制集提供 99.999% 高可用
- 分片架构支持海量数据和无缝扩容

第1.5讲：安装 MongoDB

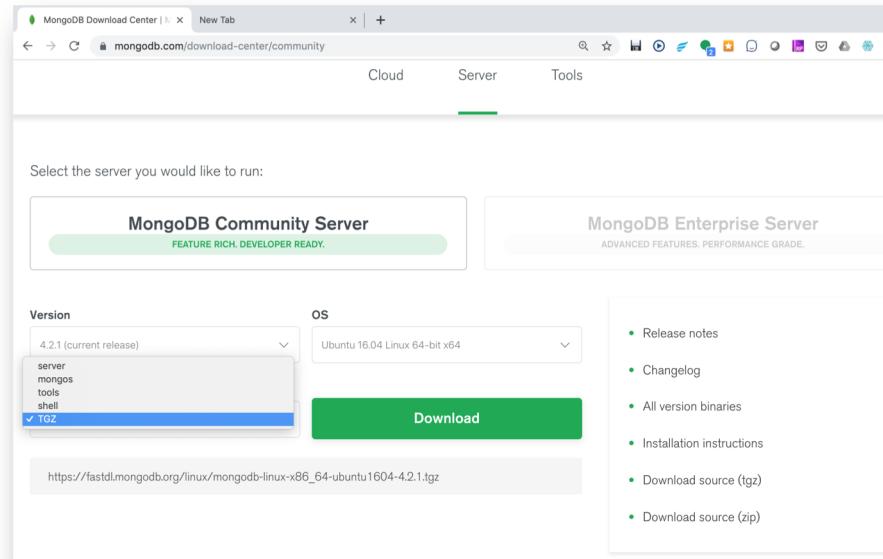
下载安装 MongoDB | Compass

本章学习内容及目标

内容大纲	学习目标
下载 MongoDB 版本	了解单节点 MongoDB 的安装
在本机安装 MongoDB	了解 MongoDB 图形管理工具
安装 MongoDB Compass	了解 MongoDB 命令行管理工具
用 mongo shell 连接 MongoDB	

下载 MongoDB

- mongodb.com/download-center
- 企业版 - 开发环境免费使用
- 社区版 - 所有环境免费使用
- 选择合适的 OS 版本
- TGZ 版本包含 server mongos tools 和 shell



Linux 上安装 MongoDB

```
$ mkdir -p /data /data/db  
$ cd /data  
$ curl -O https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-rhel70-4.2.1.tgz  
$ tar -xvf mongodb-linux-x86_64-rhel70-4.2.1.tgz  
$ export PATH=$PATH:/data/mongodb-linux-x86_64-rhel70-4.2.1/bin  
$ mongod --dbpath /data/db --port 27017 --logpath /data/db/mongod.log --fork --bind_ip 0.0.0.0
```

```
MacBook-Pro-3:~ tjworks$ mongod --dbpath ~/mongodata --port 27017 --logpath ~/mongodata/mongod.log --fork --bind_ip 0.0.0.0  
[about to fork child process, waiting until server is ready for connections.  
forked process: 53542  
child process started successfully, parent exiting  
MacBook-Pro-3:~ tjworks$ mongo localhost:27017  
MongoDB shell version v4.0.10  
connecting to: mongodb://localhost:27017/test?gssapiServiceName=mongodb  
Implicit session: session { "id" : UUID("b81a10f6-5c88-4747-bc0f-6c9040a5ea97") }  
MongoDB server version: 4.0.10  
Server has startup warnings:  
2019-11-03T14:58:06.065+0800 I CONTROL  [initandlisten]  
2019-11-03T14:58:06.065+0800 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.  
2019-11-03T14:58:06.066+0800 I CONTROL  [initandlisten] ** Read and write access to data and configuration is unrestricted.  
2019-11-03T14:58:06.067+0800 I CONTROL  [initandlisten]  
> █
```

使用 Atlas 免费账号

- 官方提供的云托管 MongoDB
- 提供一个终身免费测试账号
- 步骤：
 - 注册账号
 - 创建免费集群
 - 按照提示完成创建并获得连接串
 - 使用 mongo 命令行连接集群

The image consists of three main sections:

- Top Left:** A screenshot of the MongoDB Atlas sign-up page. A large orange circle labeled "1" highlights the "Create a cluster" button at the bottom right of the form.
- Top Right:** A screenshot of the MongoDB Atlas cluster creation interface. It shows three main options: "Starter Clusters" (highlighted with a large orange circle labeled "2"), "Single-Region Clusters", and "Multi-Region Clusters". The "Starter Clusters" section is described as being suitable for learning MongoDB or developing small applications. It lists features like "Highly available auto-healing cluster", "End-to-end encryption", and "Realtime performance metrics".
- Bottom:** A screenshot of a terminal window on a Mac OS X system (version 10.14.5) showing the connection command for a MongoDB cluster named "geektime-mongodb". The command is:


```
mongo "mongodb+srv://geektime-mongodb-bcvi.mongodb.net/test" --username geektime
```

 The terminal output shows the connection process, including host changes and successful connections to shards. A large orange circle labeled "3" highlights the "I have the Mongo Shell installed" button in the "Connect" dialog above the terminal.

Having trouble connecting? [View our troubleshooting documentation](#)

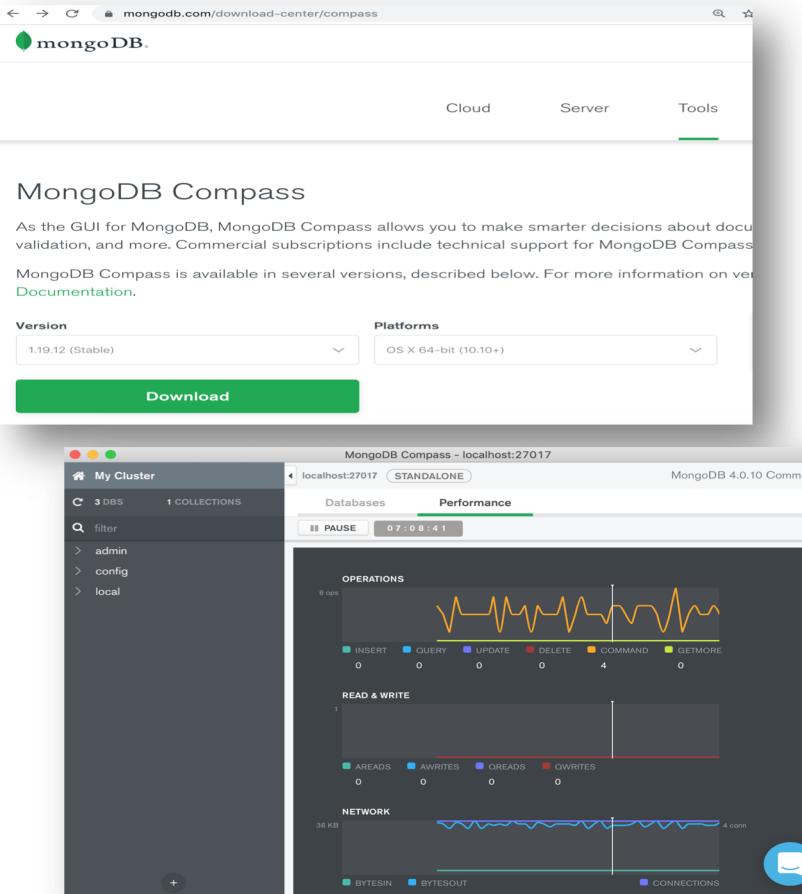
导入样本数据

- curl -O -k <https://raw.githubusercontent.com/tapdata/geektime-mongodb-course/master/aggregation/dump.tar.gz>
- tar -xvf dump.tar.gz
- mongorestore -h localhost:27017

```
[MacBook-Pro-3:aggregation tjworks$ mongorestore -h localhost:27017
2019-11-03T15:55:24.267+0800      using default 'dump' directory
2019-11-03T15:55:24.267+0800      preparing collections to restore from
2019-11-03T15:55:24.269+0800      reading metadata for mock.orders from dump/mock/orders.metadata.json
2019-11-03T15:55:24.299+0800      restoring mock.orders from dump/mock/orders.bson
2019-11-03T15:55:26.298+0800      no indexes to restore
2019-11-03T15:55:26.298+0800      finished restoring mock.orders (100000 documents)
2019-11-03T15:55:26.298+0800      done
MacBook-Pro-3:aggregation tjworks$ ]
```

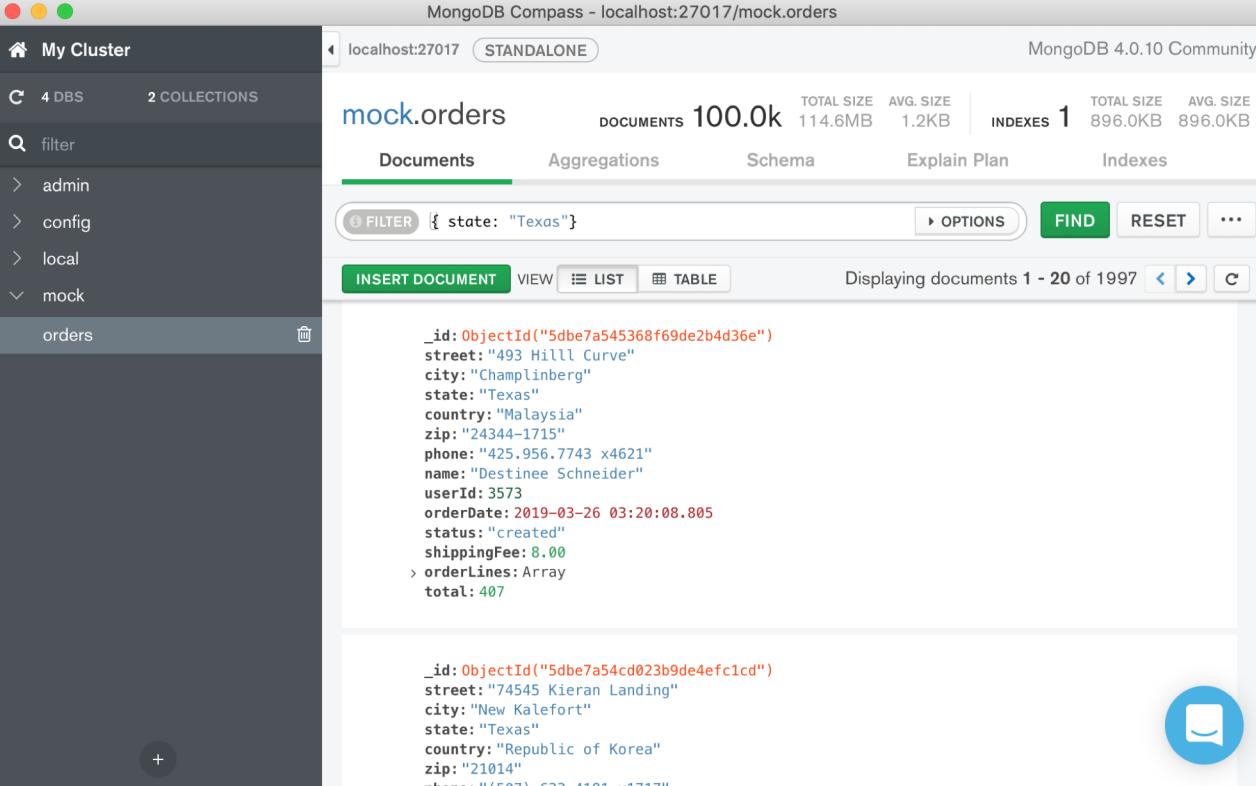
下载并安装 MongoDB Compass

- 官方提供的免费 GUI 管理工具
- 数据管理（增删改查）
- Schema 管理
- 索引管理
- 性能排查
- 实时性能监控



使用 MongoDB Compass (demo)

- 连接
- 数据库/集合列表
- 集合统计
- 数据模式
- 查询解释器
- 索引
- 实时性能监控



MongoDB Compass - localhost:27017/mock.orders

localhost:27017 STANDALONE MongoDB 4.0.10 Community

mock.orders DOCUMENTS 100.0k TOTAL SIZE 114.6MB AVG. SIZE 1.2KB INDEXES 1 TOTAL SIZE 896.0KB AVG. SIZE 896.0KB

Documents Aggregations Schema Explain Plan Indexes

FILTER { state: "Texas"} OPTIONS FIND RESET ...

INSERT DOCUMENT VIEW LIST TABLE Displaying documents 1 - 20 of 1997 < > C

`_id: ObjectId("5dbe7a545368f69de2b4d36e")
street: "493 Hill Curve"
city: "Champlinberg"
state: "Texas"
country: "Malaysia"
zip: "24344-1715"
phone: "425.956.7743 x4621"
name: "Destinee Schneider"
userId: 3573
orderDate: 2019-03-26 03:20:08.805
status: "created"
shippingFee: 8.00
> orderLines: Array
total: 407`

`_id: ObjectId("5dbe7a54cd023b9de4efc1cd")
street: "74545 Kieran Landing"
city: "New Kalefort"
state: "Texas"
country: "Republic of Korea"
zip: "21014"`

第1.6讲 MongoDB 基本操作

增删改查操作

本章学习内容及目标

内容大纲	学习目标
使用 insert 完成插入操作	了解如何完成基本的 CRUD 操作
使用 find 完成基本查询	了解库和集合的操作
使用 remove 删除文档	
使用 update 更新文档	
数据库和集合的删除	

使用 insert 完成插入操作

操作格式：

```
db.<集合>.insertOne(<JSON对象>)
db.<集合>.insertMany([<JSON 1>, <JSON 2>, ...<JSON n>])
```

示例：

```
db.fruit.insertOne({name: "apple"})
db.fruit.insertMany([
    {name: "apple"},
    {name: "pear"},
    {name: "orange"}
])
```

使用 find 查询文档

- 关于 find:

- find 是 MongoDB 中查询数据的基本指令，相当于 SQL 中的 SELECT。
- find 返回的是游标。

- find 示例：

```
db.movies.find( { "year" : 1975 } ) //单条件查询
```

```
db.movies.find( { "year" : 1989, "title" : "Batman" } ) //多条件and查询
```

```
db.movies.find( { $and : [ { "title" : "Batman"}, { "category" : "action"}] } ) // and的另一种形式
```

```
db.movies.find( { $or: [ {"year" : 1989}, {"title" : "Batman"}] } ) //多条件or查询
```

```
db.movies.find( { "title" : /^B/ } ) //按正则表达式查找
```

查询条件对照表

SQL	MQL
a = 1	{a: 1}
a <> 1	{a: {\$ne: 1}}
a > 1	{a: {\$gt: 1}}
a >= 1	{a: {\$gte: 1}}
a < 1	{a: {\$lt: 1}}
a <= 1	{a: {\$lte: 1}}

查询逻辑对照表

SQL	MQL
a = 1 AND b = 1	{a: 1, b: 1} 或 {\$and: [{a: 1}, {b: 1}]}
a = 1 OR b = 1	{\$or: [{a: 1}, {b: 1}]}
a IS NULL	{a: {\$exists: false}}
a IN (1, 2, 3)	{a: {\$in: [1, 2, 3]}}

查询逻辑运算符

- \$lt: 存在并小于
- \$lte: 存在并小于等于
- \$gt: 存在并大于
- \$gte: 存在并大于等于
- \$ne: 不存在或存在但不等于
- \$in: 存在并在指定数组中
- \$nin: 不存在或不在指定数组中
- \$or: 匹配两个或多个条件中的一个
- \$and: 匹配全部条件

使用 find 搜索子文档

- find 支持使用 “field.sub_field” 的形式查询子文档。假设有一个文档：

```
db.fruit.insertOne({  
    name: "apple",  
    from: {  
        country: "China",  
        province: "Guangdon"  
    }  
})
```

- 考虑以下查询的意义：

```
db.fruit.find( { "from.country" : "China" } )
```

```
db.fruit.find( { "from" : {country: "China"} } )
```

使用 find 搜索数组

- find 支持对数组中的元素进行搜索。假设有一个文档：

```
db.fruit.insert([
  { "name" : "Apple", color: ["red", "green"] },
  { "name" : "Mango", color: ["yellow", "green"] }
])
```

- 考虑以下查询的意义：

```
db.fruit.find({color: "red"})
```

```
db.fruit.find({$or: [{color: "red"}, {color: "yellow"}] })
```

使用 find 搜索数组中的对象

- 考虑以下文档，在其中搜索

```
db.movies.insertOne( {  
    "title" : "Raiders of the Lost Ark",  
    "filming_locations" : [  
        "USA" , { "city" : "Los Angeles", "state" : "CA", "country" :  
            { "city" : "Rome", "state" : "Lazio", "country" : "Italy" },  
            { "city" : "Florence", "state" : "SC", "country" : "USA" }  
        ]  
    })
```

- // 查找城市是 Rome 的记录
- db.movies.find({"filming_locations.city": "Rome"})

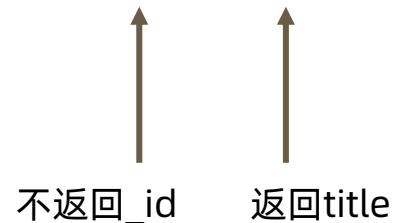
使用 find 搜索数组中的对象

- 在数组中搜索子对象的多个字段时，如果使用 \$elemMatch，它表示必须是同一个子对象满足多个条件。考虑以下两个查询：

```
db.getCollection('movies').find({  
    "filming_locations.city": "Rome",  
    "filming_locations.country": "USA"  
})  
  
db.getCollection('movies').find({  
    "filming_locations": {  
        $elemMatch: {"city": "Rome", "country": "USA"}  
    }  
})
```

控制 find 返回的字段

- find 可以指定只返回指定的字段；
- _id 字段必须明确指明不返回，否则默认返回；
- 在 MongoDB 中我们称这为投影（projection）；
- db.movies.find({"category": "action"}, {"_id:0, title:1})



使用 remove 删除文档

- remove 命令需要配合查询条件使用；
- 匹配查询条件的的文档会被删除；
- 指定一个空文档条件会删除所有文档；
- 以下示例：

```
db.testcol.remove({a: 1}) // 删除a 等于1的记录
```

```
db.testcol.remove({a: {$lt: 5}}) // 删除a 小于5的记录
```

```
db.testcol.remove({}) // 删除所有记录
```

```
db.testcol.remove() // 报错
```

使用 update 更新文档

- Update 操作执行格式：db.<集合>.update(<查询条件>, <更新字段>)
- 以以下数据为例：

```
db.fruit.insertMany([  
    {name: "apple"},  
    {name: "pear"},  
    {name: "orange"}  
])
```

```
db.fruit.updateOne({name: "apple"}, {$set: {from: "China"}})
```



查询 name 为
apple 的记录



将找到记录的 from
设置为 China

使用 update 更新文档

- 使用 updateOne 表示无论条件匹配多少条记录，始终只更新第一条；
- 使用 updateMany 表示条件匹配多少条就更新多少条；
- updateOne/updateMany 方法要求更新条件部分必须具有以下之一，否则将报错：
 - \$set/\$unset
 - \$push/\$pushAll/\$pop
 - \$pull/\$pullAll
 - \$addToSet
- // 报错

```
db.fruit.updateOne({name: "apple"}, {from: "China"})
```

使用 update 更新数组

- `$push`: 增加一个对象到数组底部
- `$pushAll`: 增加多个对象到数组底部
- `$pop`: 从数组底部删除一个对象
- `$pull`: 如果匹配指定的值，从数组中删除相应的对象
- `$pullAll`: 如果匹配任意的值，从数据中删除相应的对象
- `$addToSet`: 如果不存在则增加一个值到数组

使用 drop 删除集合

- 使用 db.<集合>.drop() 来删除一个集合
- 集合中的全部文档都会被删除
- 集合相关的索引也会被删除

```
db.colToBeDropped.drop()
```

使用 dropDatabase 删除数据库

- 使用 db.dropDatabase() 来删除数据库
- 数据库相应文件也会被删除，磁盘空间将被释放

```
use tempDB
db.dropDatabase()
show collections // No collections
show dbs // The db is gone
```

第1.7讲 Hello World 程序开发

目标

- 本实验中，我们使用 Python 为开发语言，向大家演示：
 - MongoDB 连接字符串的格式和配置；
 - 如何使用 Python 驱动连接到 MongoDB；
 - 如何使用驱动执行常规的 CRUD 指令。

1. 安装 Python MongoDB 驱动程序

- 安装 MongoDB 驱动
 - 在 Python 中使用 MongoDB 之前必须先安装用于访问数据库的驱动程序：
- 检查驱动程序
 - 在 python 交互模式下导入 pymongo，检查驱动是否已正确安装：

```
import pymongo  
pymongo.version
```

2. 创建连接

- 确定 MongoDB 连接串
 - 使用驱动连接到 MongoDB 集群只需要指定 MongoDB 连接字符串即可。其基本格式可以参考文档: [Connection String URI Format](#) 最简单的形式是
 - mongodb://数据库服务器主机地址: 端口号
 - 如: mongodb://127.0.0.1:27017
- 初始化数据库连接
 - from pymongo import MongoClient
 - uri = "mongodb://127.0.0.1:27017"
 - client = MongoClient(uri)
 - print client

3. 数据库操作：插入用户

- 初始化数据库和集合

```
db = client["eshop"]

user_coll = db["users"]
```

- 插入一条新的用户数据

```
new_user = {"username": "nina", "password": "xxxx", "email":  
"123456@qq.com"}  
  
result = user_coll.insert_one(new_user)  
  
print result
```

注意：我们并没有提前创建数据库和表/集合

4. 更新用户

- 需求变化，要求修改用户属性，增加字段phone

```
result = user_coll.update_one({ "username": "nina"},  
{ "$set": { "phone": "123456789"} }  
)  
print result
```

注意：我们并没有去数据库修改表结构

总结

- 使用 MongoDB 驱动程序操作 MongoDB 的 API 简单易行。
- 尽管大家使用的编程语言可能各不相同，但 MongoDB 驱动的设计是依照统一的规范制定，无论使用哪种语言，其原理和 API 方法都非常相似。因此只要掌握一种语言的驱动，切换到其他语言将十分容易。
- 演示所用Python源码可以在github上获取：

```
git clone https://github.com/tapdata/geektime-mongodb-course.git
```

第1.8讲 聚合查詢

本章学习内容及目标

内容大纲	学习目标
聚合框架的相关术语	了解聚合框架是什么
聚合框架的使用格式	了解聚合框架的使用方式
常见步骤示例	了解聚合框架的使用场景
使用聚合框架的场景	
聚合查询示例	

什么是 MongoDB 聚合框架

- MongoDB 聚合框架（Aggregation Framework）是一个计算框架，它可以：
 - 作用在一个或几个集合上；
 - 对集合中的数据进行的一系列运算；
 - 将这些数据转化为期望的形式；
- 从效果而言，聚合框架相当于 SQL 查询中的：
 - GROUP BY
 - LEFT OUTER JOIN
 - AS等

管道 (Pipeline) 和步骤 (Stage)

- 整个聚合运算过程称为管道 (Pipeline)，它是由多个步骤 (Stage) 组成的，每个管道：
 - 接受一系列文档（原始数据）；
 - 每个步骤对这些文档进行一系列运算；
 - 结果文档输出给下一个步骤；



聚合运算的基本格式

```
pipeline = [ $stage1, $stage2, ...$stageN ] ;
```

```
db.<COLLECTION>.aggregate (
    pipeline,
    { options }
) ;
```

常见步骤

步骤	作用	SQL等价运算符
\$match	过滤	WHERE
\$project	投影	AS
\$sort	排序	ORDER BY
\$group	分组	GROUP BY
\$skip/\$limit	结果限制	SKIP/LIMIT
\$lookup	左外连接	LEFT OUTER JOIN

常见步骤中的运算符

\$match	\$project	\$group
<ul style="list-style-type: none">• \$eq/\$gt/\$gte/\$lt/\$lte• \$and/\$or/\$not/\$in• \$geoWithin/\$intersect•	<ul style="list-style-type: none">• 选择需要的或排除不需要的字段• \$map/\$reduce/\$filter• \$range• \$multiply/\$divide/\$subtract/\$add• \$year/\$month/\$dayOfMonth/\$hour/\$minute/\$second•	<ul style="list-style-type: none">• \$sum/\$avg• \$push/\$addToSet• \$first/\$last/\$max/\$min•

常见步骤

步骤	作用	SQL等价运算符
\$unwind	展开数组	N/A
\$graphLookup	图搜索	N/A
\$facet/\$bucket	分面搜索	N/A

聚合运算的使用场景

- 聚合查询可以用于OLAP和OLTP场景。例如：

OLTP	OLAP
<ul style="list-style-type: none">● 计算	<ul style="list-style-type: none">● 分析一段时间内的销售总额、均值● 计算一段时间内的净利润● 分析购买人的年龄分布● 分析学生成绩分布● 统计员工绩效

MQL 常用步骤与 SQL 对比

```
| SELECT  
| FIRST_NAME AS `名`,  
| LAST_NAME AS `姓`  
| FROM Users  
| WHERE GENDER = '男'  
| SKIP 100  
| LIMIT 20
```



```
db.users.aggregate([  
  {$match: {gender: "男"}},  
  {$skip: 100},  
  {$limit: 20},  
  {$project: {  
    '名': '$first_name',  
    '姓': '$last_name'  
  }}  
]);
```

MQL 常用步骤与 SQL 对比

```
SELECT DEPARTMENT,  
       COUNT(NULL) AS EMP_QTY  
FROM Users  
WHERE GENDER = '女'  
GROUP BY DEPARTMENT HAVING  
       COUNT(*) < 10
```



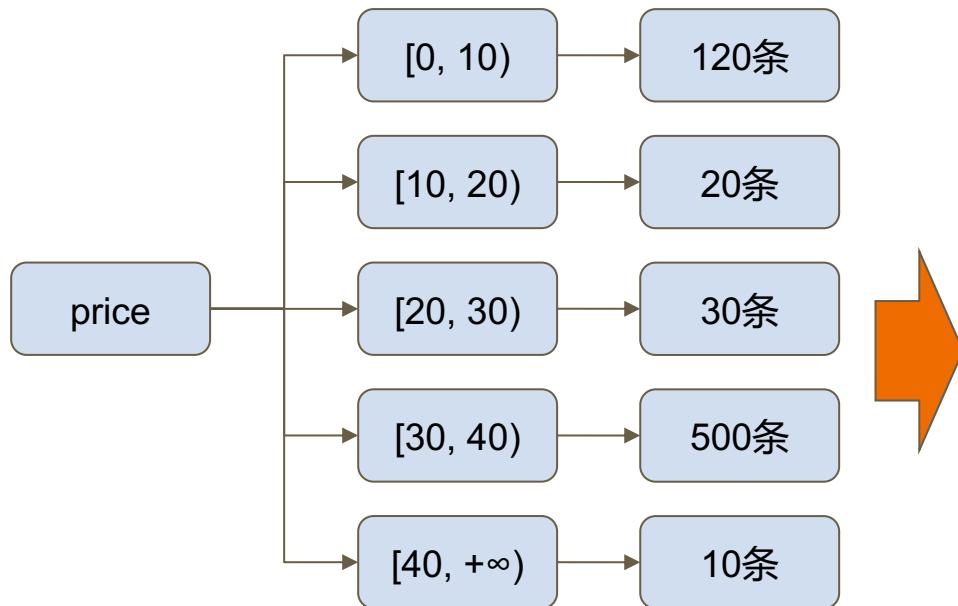
```
db.users.aggregate([  
  {$match: {gender: '女'}},  
  {$group: {  
    _id: '$DEPARTMENT',  
    emp_qty: {$sum: 1}  
  }},  
  {$match: {emp_qty: {$lt: 10}}}]));
```

MQL 特有步骤 \$unwind

```
> db.students.findOne()  
{  
    name: '张三',  
    score: [  
        {subject: '语文', score: 84},  
        {subject: '数学', score: 90},  
        {subject: '外语', score: 69}  
    ]  
}
```

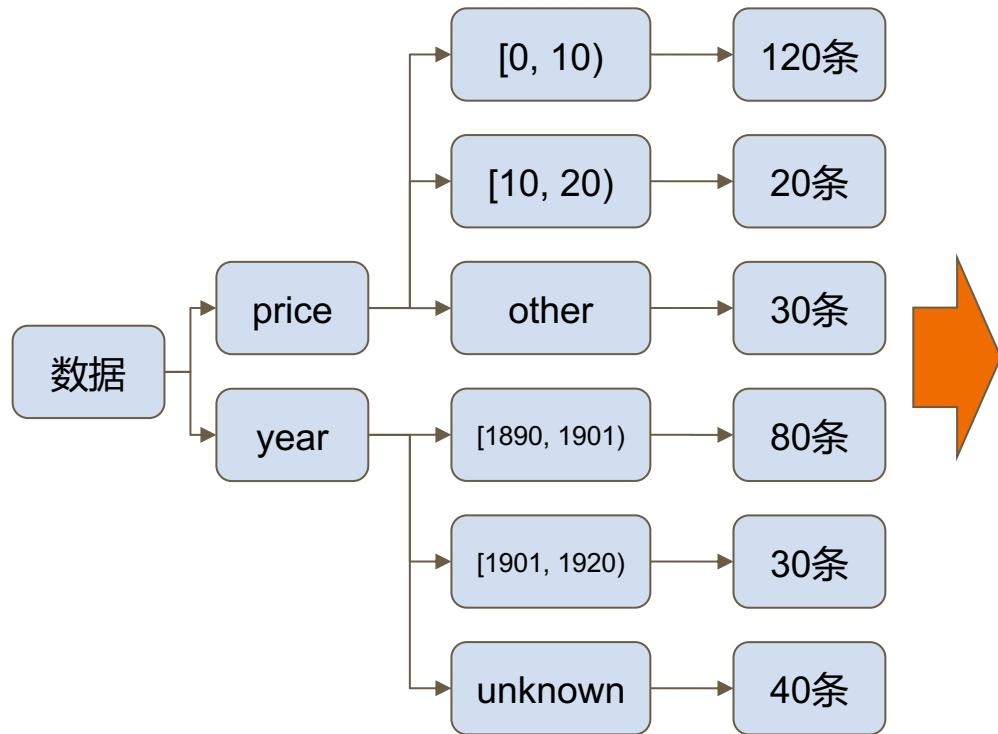
```
> db.students.aggregate([{$unwind: '$score'}])  
[  
    {name: '张三', score: {subject: '语文', score: 84}},  
    {name: '张三', score: {subject: '数学', score: 90}},  
    {name: '张三', score: {subject: '外语', score: 69}}  
]
```

MQL 特有步骤 \$bucket



```
db.products.aggregate([  
  $bucket:{  
    groupBy: "$price",  
    boundaries: [0,10,20,30,40],  
    default: "Other",  
    output:{"count":{$sum:1}}  
  }])
```

MQL 特有步骤 \$facet



```
db.products.aggregate([  
  $facet:{  
    price:{  
      $bucket:{...}  
    },  
    year:{  
      $bucket:{...}  
    }  
  }])
```

第1.9讲 聚合查询实验

目标

- 本实验中，我们使用了模拟数据 orders 来演示一些常规的聚合操作。通过这些聚合操作实验，大家可以了解到：
 - 聚合操作的基本格式；
 - 条件过滤步骤 \$match；
 - 投影步骤 \$project；
 - 展开数据步骤 \$unwind；
 - 分组聚合步骤 \$group。

聚合实验数据模型

```
> _id: ObjectId("5dbe7a545368f69de2b4d36e")
  street: "493 Hilll Curve"
  city: "Champlinberg"
  state: "Texas"
  country: "Malaysia"
  zip: "24344-1715"
  phone: "425.956.7743 x4621"
  name: "Destinee Schneider"
  userId: 3573
  orderDate: 2019-03-26 03:20:08.805
  status: "created"
  shippingFee: 8.00
  < orderLines: Array
    < 0: Object
      product: "Refined Fresh Tuna"
      sku: "2057"
      qty: 25
      price: 56.00
      cost: 46.48
    > 1: Object
    > 2: Object
    > 3: Object
    > 4: Object
    > 5: Object
  total: 407
```

聚合实验一：总销量

- 计算到目前为止的所有订单的总销售额

```
db.orders.aggregate([
  { $group:
    {
      _id: null,
      total: { $sum: "$total" }
    }
  }
])  
// 结果: // { "_id" : null, "total" : NumberDecimal("44019609") }
```

聚合实验二：订单金额汇总

- 查询2019年第一季度（1月1日~3月31日）已完成订单（completed）的订单总金额和订单总数

```
db.orders.aggregate([
    // 步骤1：匹配条件
    { $match: { status: "completed", orderDate: {
        $gte: ISODate("2019-01-01"),
        $lt: ISODate("2019-04-01") } } },
    // 步骤二：聚合订单总金额、总运费、总数量
    { $group: {
        _id: null,
        total: { $sum: "$total" },
        shippingFee: { $sum: "$shippingFee" },
        count: { $sum: 1 } } },
    { $project: {
        // 计算总金额
        grandTotal: { $add: ["$total", "$shippingFee"] },
        count: 1,
        _id: 0 } }
])
// 结果:
// { "count" : 5875, "grandTotal" : NumberDecimal("2636376.00") }
```

总结

- 聚合框架可以帮助我们完成常见的分析应用类需求
- MongoDB 聚合框架支持丰富的步骤运算符
- 通过 Compass 可以更加直观的来创建复杂的聚合计算管道

第1.10讲 复制集机制及原理

本章学习内容及目标

内容大纲	学习目标
复制集的作用	了解复制集工作原理
复制集的典型架构	了解复制集典型架构
复制集复制数据原理	了解常用选项及其使用场景
复制集故障恢复原理	
常见复制集配置选项	
使用注意事项	

复制集的作用

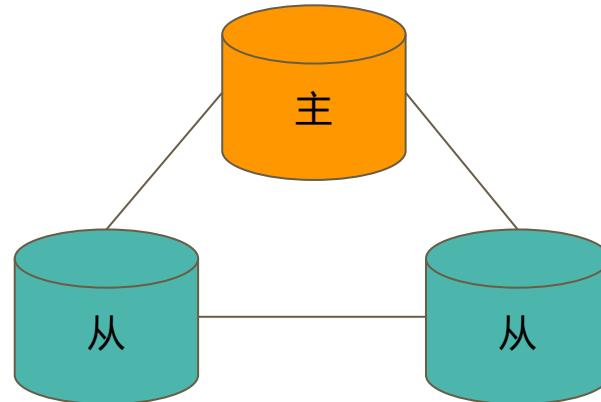
- MongoDB 复制集的主要意义在于实现服务高可用
- 它的现实依赖于两个方面的功能：
 - 数据写入时将数据迅速复制到另一个独立节点上
 - 在接受写入的节点发生故障时自动选举出一个新的替代节点

复制集的作用

- 在实现高可用的同时，复制集实现了其他几个附加作用：
 - 数据分发：将数据从一个区域复制到另一个区域，减少另一个区域的读延迟
 - 读写分离：不同类型的压力分别在不同的节点上执行
 - 异地容灾：在数据中心故障时候快速切换到异地

典型复制集结构

- 一个典型的复制集由3个以上具有投票权的节点组成，包括：
 - 一个主节点（PRIMARY）：接受写入操作和选举时投票
 - 两个（或多个）从节点（SECONDARY）：复制主节点上的新数据和选举时投票
 - 不推荐使用 Arbiter（投票节点）



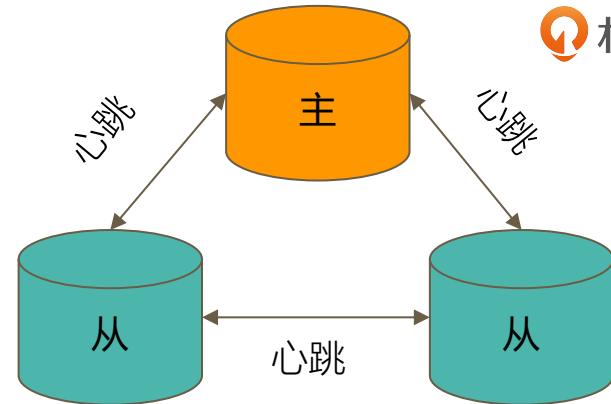
数据是如何复制的？

- 当一个修改操作，无论是插入、更新或删除，到达主节点时，它对数据的操作将被记录下来（经过一些必要的转换），这些记录称为 oplog。
- 从节点通过在主节点上打开一个 tailable 游标不断获取新进入主节点的 oplog，并在自己的数据上回放，以此保持跟主节点的数据一致。



通过选举完成故障恢复

- 具有投票权的节点之间两两互相发送心跳；
- 当5次心跳未收到时判断为节点失联；
- 如果失联的是主节点，从节点会发起选举，选出新的主节点；
- 如果失联的是从节点则不会产生新的选举；
- 选举基于 [RAFT一致性算法](#) 实现，选举成功的必要条件是大多数投票节点存活；
- 复制集中最多可以有50个节点，但具有投票权的节点最多7个。



影响选举的因素

- 整个集群必须有大多数节点存活；
- 被选举为主节点的节点必须：
 - 能够与多数节点建立连接
 - 具有较新的 oplog
 - 具有较高的优先级（如果有配置）

常见选项

- 复制集节点有以下常见的选配项：

- 是否具有投票权（`v` 参数）：有则参与投票；
- 优先级（`priority` 参数）：优先级越高的节点越优先成为主节点。优先级为0的节点无法成为主节点；
- 隐藏（`hidden` 参数）：复制数据，但对应用不可见。隐藏节点可以具有投票权，但优先级必须为0；
- 延迟（`slaveDelay` 参数）：复制 `n` 秒之前的数据，保持与主节点的时间差。



复制集注意事项

- 关于硬件：
 - 因为正常的复制集节点都有可能成为主节点，它们的地位是一样的，因此硬件配置上必须一致；
 - 为了保证节点不会同时宕机，各节点使用的硬件必须具有独立性。
- 关于软件：
 - 复制集各节点软件版本必须一致，以避免出现不可预知的问题。
- 增加节点不会增加系统写性能！

第1.11讲 实验：搭建 MongoDB 复制集

目标

- 本实验中，我们通过在一台机器上运行3个实例来搭建一个最简单的复制集。通过实验，我们将学会：
 - 如何启动一个 MongoDB 实例
 - 如何将3个 MongoDB 实例搭建成一个复制集
 - 如何对复制集运行参数做一些常规调整

1. 准备

- 安装最新的 MongoDB 版本
- Windows 系统请事先配置好 MongoDB 可执行文件的环境变量
- Linux 和 Mac 系统请配置 PATH 变量
- 确保有 10GB 以上的硬盘空间

2. 创建数据目录

- MongoDB 启动时将使用一个数据目录存放所有数据文件。我们将为3个复制集节点创建各自的数据目录。

- Linux/MacOS:

```
mkdir -p /data/db{1,2,3}
```

- Windows:

```
md c:\data\db1  
md c:\data\db2  
md c:\data\db3
```

3. 准备配置文件

复制集的每个mongod进程应该位于不同的服务器。我们现在在一台机器上运行3个进程，因此要为它们各自配置：

- 不同的端口。示例中将使用28017/28018/28019
- 不同的数据目录。示例中将使用：

/data/db1或c:\data\db1

/data/db2或c:\data\db2

/data/db3或c:\data\db3

- 不同日志文件路径。示例中将使用：

/data/db1/mongod.log或c:\data\db1\mongod.log

/data/db2/mongod.log或c:\data\db2\mongod.log

/data/db3/mongod.log或c:\data\db3\mongod.log

3. 准备配置文件（续）

Linux/MacOS

```
# /data/db1/mongod.conf
systemLog:
  destination: file
  path: /data/db1/mongod.log    # log path
  logAppend: true
storage:
  dbPath: /data/db1      # data directory
net:
  bindIp: 0.0.0.0
  port: 28017    # port
replication:
  replSetName: rs0
processManagement:
  fork: true
```

Windows

```
# c:\data\db1\mongod.conf
systemLog:
  destination: file
  path: c:\data1\mongod.log    # 日志文件路径
  logAppend: true
storage:
  dbPath: c:\data1      # 数据目录
net:
  bindIp: 0.0.0.0
  port: 28017    # 端口
replication:
  replSetName: rs0
```

4. 启动 MongoDB 进程

- Linux/Mac:

```
mongod -f /data/db1/mongod.conf  
mongod -f /data/db2/mongod.conf  
mongod -f /data/db3/mongod.conf
```

注意：如果启用了 SELinux，可能阻止上述进程启动。简单起见请关闭 SELinux。

- Windows:

```
mongod -f c:\data1\mongod.conf  
mongod -f c:\data2\mongod.conf  
mongod -f c:\data3\mongod.conf
```

因为 Windows 不支持 fork，以上命令需要在3个不同的窗口执行，执行后不可关闭窗口否则进程将直接结束。

5. 配置复制集

方法1

```
# mongo --port 28017
> rs.initiate()
> rs.add("HOSTNAME:28018")
> rs.add("HOSTNAME:28019")
```

注意：此方式hostname 需要能被解析

方法2

```
# mongo --port 28017
> rs.initiate({
    _id: "rs0",
    members: [ {
        _id: 0,
        host: "localhost:28017"
    }, {
        _id: 1,
        host: "localhost:28018"
    }, {
        _id: 2,
        host: "localhost:28019"
    } ]
})
```

6. 验证

MongoDB 主节点进行写入

```
# mongo localhost:28017  
> db.test.insert({ a:1 })  
  
> db.test.insert({ a:2 });
```

MongoDB 从节点进行读

```
# mongo localhost:28018  
> rs.slaveOk()  
> db.test.find()  
  
> db.test.find()  
{ "_id" : ObjectId("5dc4c63c672479f4de27d028"), "a" : 1 }  
  
> db.test.find()  
> db.test.find()  
{ "_id" : ObjectId("5dc4c63c672479f4de27d028"), "a" : 1 }  
{ "_id" : ObjectId("5dc4c65d672479f4de27d029"), "a" : 2 }
```

总结

- MongoDB 的生产环境要求搭建复制集
- MongoDB 复制集的搭建非常简单
- 实际环境要分三个物理（虚拟机）节点（或以上）

第1.12讲 MongoDB 全家桶

MongoDB 系列软件及工具

MongoDB 全家桶

软件模块	描述
mongod	MongoDB 数据库软件
mongo	MongoDB 命令行工具，管理 MongoDB 数据库
mongos	MongoDB 路由进程，分片环境下使用
mongodump / mongorestore	命令行数据库备份与恢复工具
mongoexport / mongoimport	CSV/JSON 导入与导出，主要用于不同系统间数据迁移
Compass	MongoDB GUI 管理工具
Ops Manager(企业版)	MongoDB 集群管理软件
BI Connector(企业版)	SQL 解释器 / BI 套接件
MongoDB Charts(企业版)	MongoDB 可视化软件
Atlas (付费及免费)	MongoDB 云托管服务，包括永久免费云数据库

mongodump / mongorestore

- 类似于 MySQL 的 dump/restore 工具
- 可以完成全库 dump：不加条件
- 也可以根据条件 dump 部分数据：-q 参数
- Dump 的同时跟踪数据就更：--oplog
- Restore 是反操作，把 mongodump 的输出导入到 mongodb

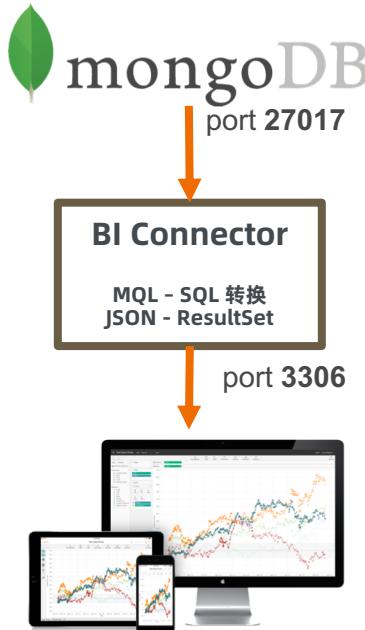
```
mongodump -h 127.0.0.1:27017 -d test -c test
```

```
mongorestore -h 127.0.0.1:27017 -d test -c test xxx.bson
```

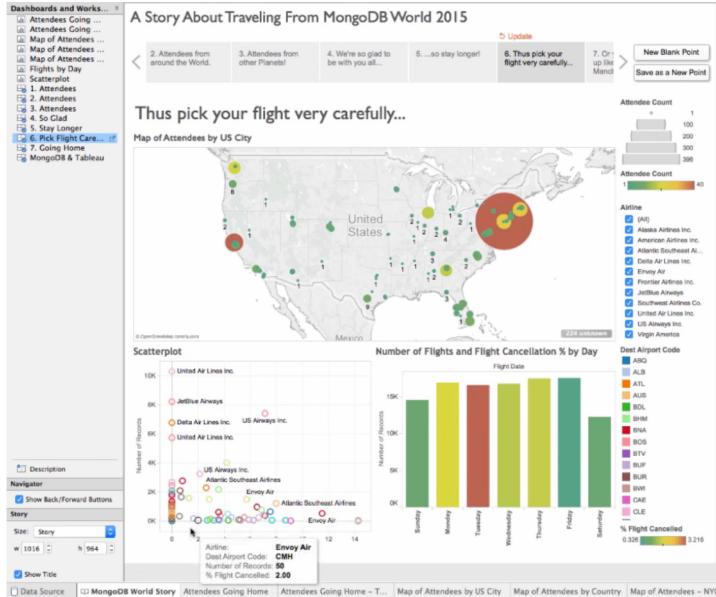
Atlas - MongoDB 公有云托管服务

<p>自助服务及弹性</p> <p>采用最好的操作自动化，按需要进行部署、修改和升级</p> <p>数据库自动维护</p> <p>数据库及基础框架资源如代码</p> <p>只需要点击几次或 API 调用，即可向上、向外或向下扩展。</p>	<p>全局及云诊断</p> <p>在 AWS、Azure、GCP 的60多个区域可用</p> <p>适用于在任何地方读/写部署和多区域故障容差的全局集群</p> <p>云服务提供商拥有一致性经验，方便数据转移</p>	<p>企业级安全及SLA</p> <p>网络隔离、VPC 同级化、端对端加密和基于角色的访问控制</p> <p>加密密钥管理、LDPA 集成、精细数据库审计</p> <p>SOC 2 /隐私盾 / HIPAA</p> <p>保证 SLA 的稳定性</p>
<p>综合监控</p> <p>积极警报，可见度超过100个关键绩效指标（KPI）</p> <p>实时性能追踪和性能顾问</p> <p>API 与监控仪表盘集成</p>	<p>受管制备份</p> <p>任意时间点数据恢复</p> <p>可查询备份快照</p> <p>分片部署一致性快照</p> <p>云数据移动性</p>	<p>Stitch：无服务器平台服务</p> <p>后端逻辑、服务集成和 API 的简单、无服务器功能</p> <p>在简单的字段级访问规则保护下，可从前端设备访问数据库</p> <p>数据库和验证触发，实时对变化做出反应</p>

MongoDB BI Connector



提供MySQL语法兼容的SQL解释器



MongoDB Ops Manager - 集群管理平台

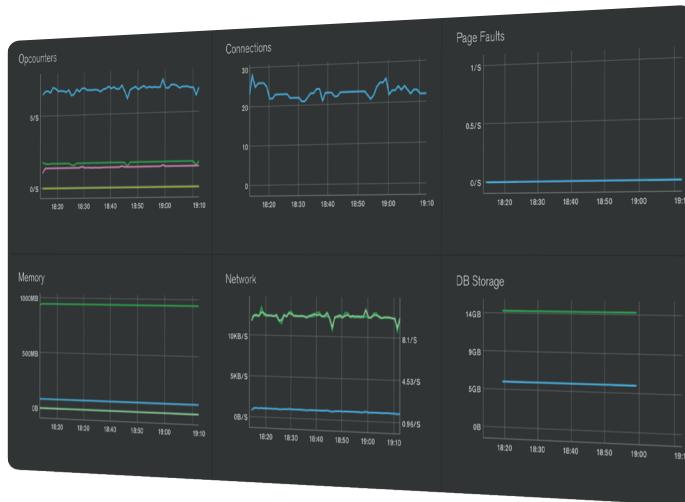
数据中心



私有云



公有云



主要功能：

- 自动化管理：补丁、升级、在线扩容
- 监控及报警
- 持续备份及任意时间点恢复
- Kubernetes 集成

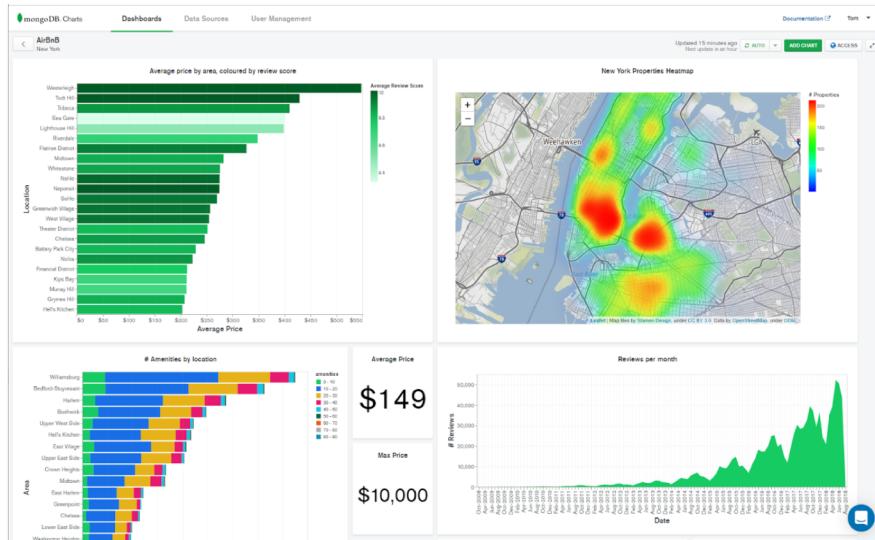


kubernetes



Pivotal Cloud Foundry

MongoDB Charts



- 托拉拽创建 MongoDB 图表
- 创建、分享和嵌入 MongoDB 数据可视化的最快、最便捷方式
- 专为 MongoDB 文档模型设计
- 一行代码在你网页应用程序中嵌入图表

本章小结

- MongoDB 的历史及发展
- MongoDB 的特色
- MongoDB 文档模型
- 版本选择，安装
- 命令行与图形工具
- 基础操作
- 聚合框架
- 复制集
- MongoDB 相关软件



扫码试看/订阅

《MongoDB 高手课》视频课程