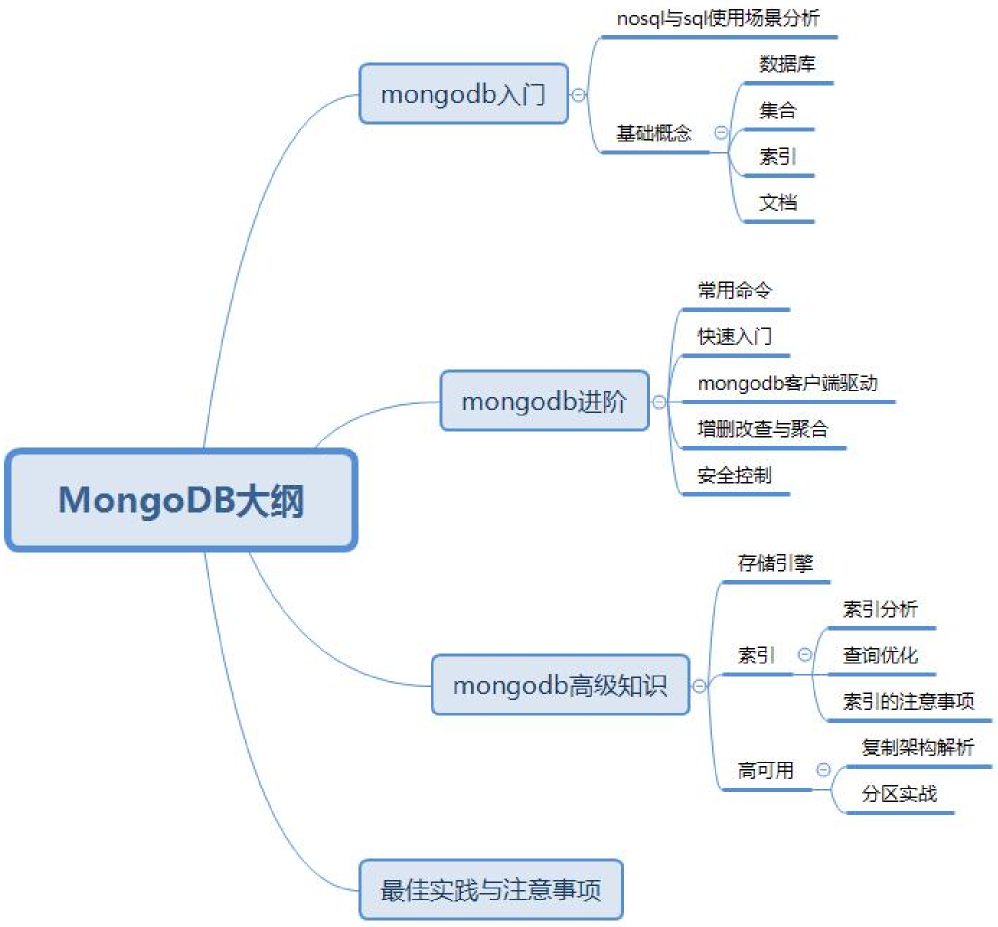
1. MongoDb 综述

1.1. 课程概述



1.2. 什么是 Nosql

NoSQL：Not Only SQL ,本质也是一种数据库的技术，相对于传统数据库技术，它不会遵循一些约束，比如：sql 标准、ACID 属性，表结构等。

Nosql 优点

- 满足对数据库的高并发读写
- 对海量数据的高效存储和访问
- 对数据库高扩展性和高可用性

● 灵活的数据结构，满足数据结构不固定的场景

Nosql 缺点

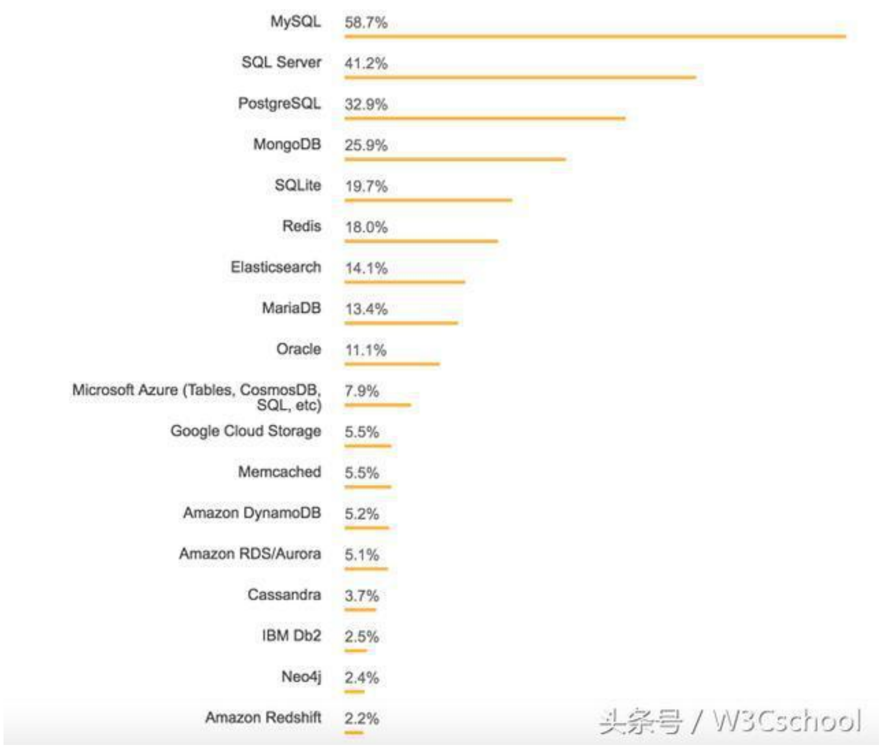
● 一般不支持事务
● 实现复杂 SQL 查询比较复杂
● 运维人员数据维护门槛较高
● 目前不是主流的数据库技术

1.2.1. NoSql 分类

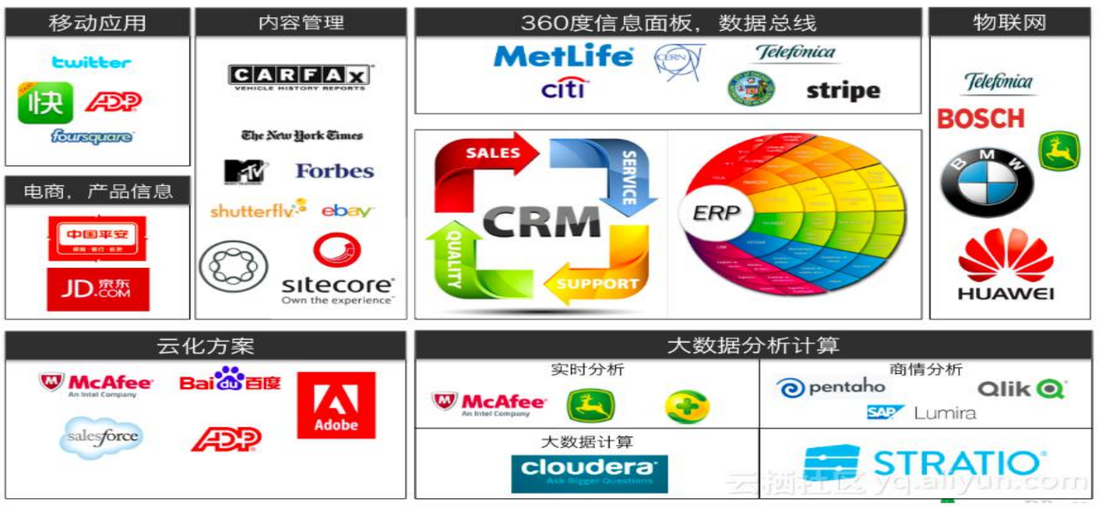
序号	类型	应用场景	典型产品
1	Key-value存储	缓存，处理高并发数据访问	Redis memcached
2	列式数据库	分布式文件系统	Cassandra Hbase
3	文档型数据库	Web应用，并发能力较强，表结构可变	mongoDB
4	图结构数据库	社交网络，推荐系统，关注构建图谱	infoGrid Neo4J

1.2.2. 数据库流行程度排行

https://db-engines.com/en/ranking



1.2.3. 谁在使用 MongoDB



1.3. MongoDb 概念入门

1.3.1.　什么是 MongoDB

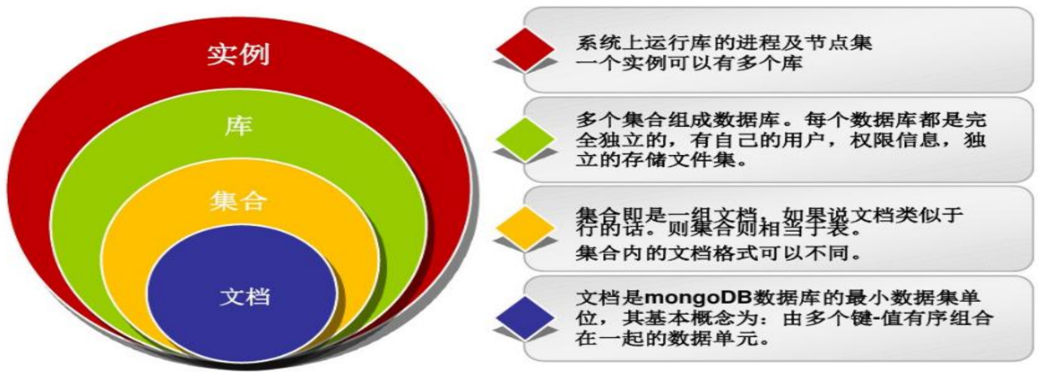
MongoDB：是一个数据库 ,高性能、无模式、文档性，目前 nosql 中最热门的数据库，开源产品，基于 c++开发。是 nosql 数据库中功能最丰富，最像关系数据库的。

特性

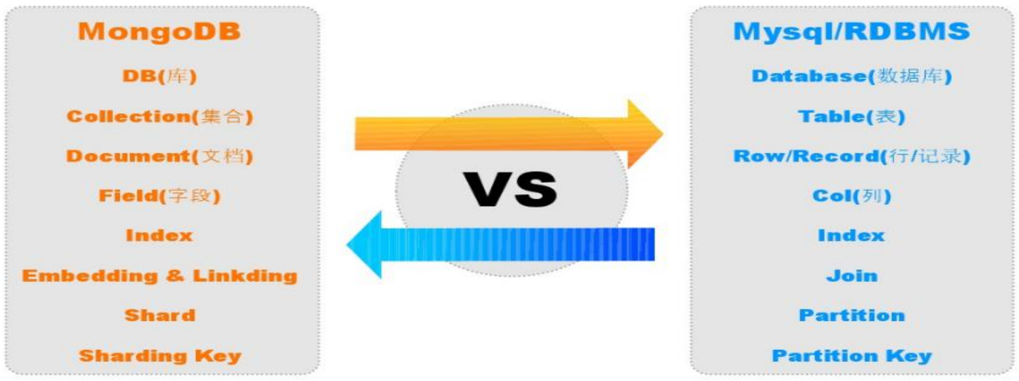
- 面向集合文档的存储：适合存储 Bson（json 的扩展）形式的数据；
- 格式自由，数据格式不固定，生产环境下修改结构都可以不影响程序运行；
- 强大的查询语句，面向对象的查询语言，基本覆盖 sql 语言所有能力；
- 完整的索引支持，支持查询计划；
- 支持复制和自动故障转移；
- 支持二进制数据及大型对象（文件）的高效存储；

- 使用分片集群提升系统扩展性；
- 使用内存映射存储引擎，把磁盘的 IO 操作转换成为内存的操作；

1.3.2. MongoDB 基本概念


实例
库
集合
文档
系统上运行库的进程及节点集
一个实例可以有多个库
多个集合组成数据库。每个数据库都是完全独立的，有自己的用户，权限信息，独立的存储文件集。
集合即是一组文档，如果说文档类似于行的话。则集合则相当于表。
集合内的文档格式可以不同。
文档是mongoDB数据库的最小数据集单位，其基本概念为：由多个键-值有序组合在一起的数据单元。

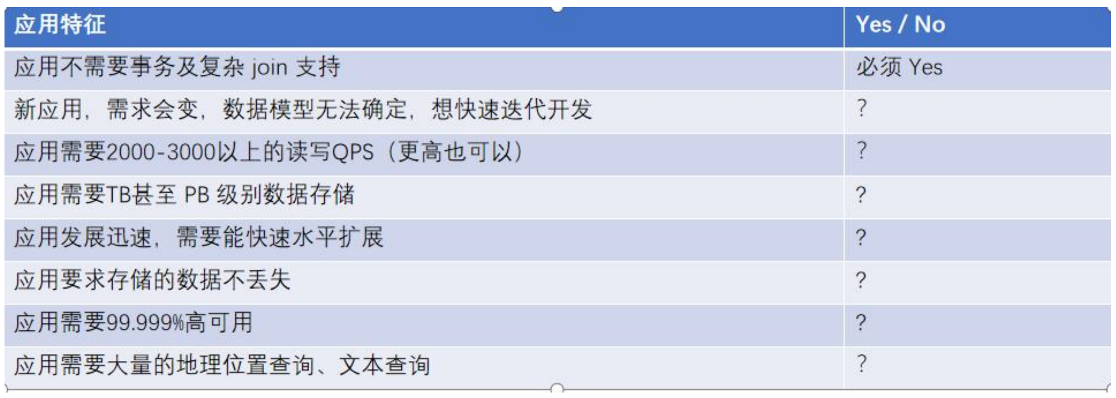
1.3.3. MongoDB 概念与 RDMS 概念对比



MongoDB	Mysql/RDBMS
DB(库)	Database(数据库)
Collection(集合)	Table(表)
Document(文档)	Row/Record(行/记录)
Field(字段)	Col(列)
Index	Index
Embedding & Linkding	Join
Shard	Partition
Sharding Key	Partition Key

1.3.4. 应不应该用 MongoDB？

并没有某个业务场景必须要使用 MongoDB 才能解决，但使用 MongoDB 通常能让你以更低的成本解决问题（包括学习、开发、运维等成本）



应用特征	Yes / No
应用不需要事务及复杂 join 支持	必须 Yes
新应用，需求会变，数据模型无法确定，想快速迭代开发	?
应用需要2000-3000以上的读写QPS（更高也可以）	?
应用需要TB甚至 PB 级别数据存储	?
应用发展迅速，需要能快速水平扩展	?
应用要求存储的数据不丢失	?
应用需要99.999%高可用	?
应用需要大量的地理位置查询、文本查询	?

如果上述有 1 个 Yes，可以考虑 MongoDB，2 个及以上的 Yes，选择 MongoDB 绝不会后悔！

1.3.5. MongoDB 使用场景

MongoDB 的应用已经渗透到各个领域，比如游戏、物流、电商、内容管理、社交、物联网、视频直播等，以下是几个实际的应用案例：

- 游戏场景，使用 MongoDB 存储游戏用户信息，用户的装备、积分等直接以内嵌文档的形式存储，方便查询、更新
- 物流场景，使用 MongoDB 存储订单信息，订单状态在运送过程中会不断更新，以 MongoDB 内嵌数组的形式来存储，一次查询就能将订单所有的变更读取出来。
- 社交场景，使用 MongoDB 存储存储用户信息，以及用户发表的朋友圈信息，通过地理位置索引实现附近的人、地点等功能
- 物联网场景，使用 MongoDB 存储所有接入的智能设备信息，以及设备汇报的日志信息，并对这些信息进行多维度的分析
- 视频直播，使用 MongoDB 存储用户信息、礼物信息等
- ……

1.3.6. 不使用 MongoDB 的场景

- 高度事务性系统：例如银行、财务等系统。MongoDB 对事物的支持较弱；
- 传统的商业智能应用：特定问题的数据分析，多数据实体关联，涉及到复杂的、高度优化的查询方式；
- 使用 sql 方便的时候；数据结构相对固定，使用 sql 进行查询统计更加便利的时候；

2. MongoDB 应用与开发

2.1. MongoDB 安装

● 官网下载安装介质：https://www.mongodb.com/download-center，选择适当的版本，这里以 linux 版本 mongodb-linux-x86_64-4.0.4 为例；

https://www.mongodb.org/dl/linux/x86_64

```
tar zxvf mongodb-linux-x86_64-4.0.4.tgz
mv mongodb-linux-x86_64-4.0.4 mongodb
mkdir -p mongodb/{data/db,log,conf}
vi mongodb/conf/mgdb.conf
```

https://docs.mongodb.com/v2.4/reference/configuration-options/

```
dbpath=/soft/mongodb/data/db   #数据文件存放目录
logpath=/soft/mongodb/log/mongodb.log   #日志文件存放目录
port=27017   #端口，默认 27017，可以自定义
logappend=true   #开启日志追加添加日志
fork=true   #以守护程序的方式启用，即在后台运行
bind_ip=0.0.0.0   #本地监听 IP，0.0.0.0 表示本地所有 IP
auth=false   #是否需要验证权限登录(用户名和密码)
```

修改环境变量

```
vi /etc/profile
export MONGODB_HOME=/soft/mongodb
export PATH=$PATH:$MONGODB_HOME/bin
source /etc/profile
```

配置开机启动

```
vi /usr/lib/systemd/system/mongodb.service
```

```
[Unit]
Description=mongodb
After=network.target remote-fs.target nss-lookup.target

[Service]
Type=forking
RuntimeDirectory=mongodb
PIDFile=/soft/mongodb/data/db/mongod.lock
ExecStart=/soft/mongodb/bin/mongod --config /soft/mongodb/conf/mgdb.conf
```

```
ExecStop=/soft/mongodb/bin/mongod --shutdown --config /soft/mongodb/conf/mgdb.conf
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

systemctl daemon-reload
systemctl start mongodb
systemctl enable mongodb

启动 mongodb
service mongodb stop
service mongodb start

https://docs.mongodb.com/v4.0/reference/configuration-options/#storage.dbPath

```
storage:
    dbPath: "/soft/mongodb/data/db"
systemLog:
    destination: file
    path: "/soft/mongodb/log/mongodb.log"
net:
    bindIp: 0.0.0.0
    port: 27017
processManagement:
    fork: true
setParameter:
    enableLocalhostAuthBypass: false
```

2.2. 快速入门

2.2.1. 目标

- 直观感受 mongoDB 的魅力
- mongo 开发入门（原生、spring）
- 开发框架版本选择
- mongoDB 数据类型全解析
- 对 nosql 的理念有初步的认识

执行命令

mongo

2.2.2. 数据结构介绍

```
{
            "_id" : ObjectId("59f938235d93fc4af8a37114"),
            "username" : "lison",
            "country" : "in11digo",
            "address" : {
                        "aCode" : "邮编",
                        "add" : "d11pff"
            },
            "favorites" : {
                        "movies" : ["杀破狼 2","1dushe","雷神 1"],
                        "cites" : ["1sh","1cs","1zz"]
             },
          "age" : 18，
          "salary"：NumberDecimal("2.099"),
          "lenght" ： 1.79
}
```

2.2.3. 需求描述

- **新增 5 人**

- **查询**

查询喜欢的城市包含东莞和东京的 user

    select * from users   where favorites.cites has "东莞"、"东京"

查询国籍为英国或者美国，名字中包含 s 的 user

    select * from users   where username like '%s%' and (country= English or country= USA)

- **修改**

把 lison 的年龄修改为 6 岁

    update   users   set age=6 where username = lison'

喜欢的城市包含东莞的人，给他喜欢的电影加入"小电影 2""小电影 3"

    update users   set favorites.movies add "小电影 2 ", "小电影 3" where favorites.cites   has "东莞"

- 删除

删除名字为 lison 的 user

    delete from users where username = ‘lison’

删除年龄大于 8 小于 25 的 user

```
    delete from users where age >8 and age <25
```

- 事务操作

Lison 和 james 要完成一次事务操作，james 转账 0.5 给 lison

```
 update    users    set lenght= lenght-0.5    where username = ‘james’
 update    users    set lenght= lenght+0.5    where username = ‘lison’
```

2.2.4. 使用 MongoDB 脚本实现

2.2.4.1. 新增 5 人

```
db.users.drop();
var user1 = {
            "username" : "lison",
            "country" : "china",
            "address" : {
                        "aCode" : "411000",
                        "add" : "长沙"
            },
            "favorites" : {
                        "movies" : ["杀破狼 2","战狼","雷神 1"],
                        "cites" : ["长沙","深圳","上海"]
            },
            "age" : 18,
           "salary":NumberDecimal("18889.09"),
           "lenght" :1.79

};
var user2 = {
            "username" : "james",
            "country" : "English",
            "address" : {
                        "aCode" : "311000",
                        "add" : "地址"
            },
            "favorites" : {
```

```
                        "movies" : ["复仇者联盟","战狼","雷神 1"],
                        "cites" : ["西安","东京","上海"]
            },
            "age" : 24,
          "salary":NumberDecimal("7889.09"),
          "lenght" :1.35
};
var user3 ={
            "username" : "deer",
            "country" : "japan",
            "address" : {
                        "aCode" : "411000",
                        "add" : "长沙"
            },
            "favorites" : {
                        "movies" : ["肉蒲团","一路向西","倩女幽魂"],
                        "cites" : ["东莞","深圳","东京"]
            },
            "age" : 22,
          "salary":NumberDecimal("6666.66"),
          "lenght" :1.85
};
var user4 =
{
            "username" : "mark",
            "country" : "USA",
            "address" : {
                        "aCode" : "411000",
                        "add" : "长沙"
            },
            "favorites" : {
                        "movies" : ["蜘蛛侠","钢铁侠","蝙蝠侠"],
                        "cites" : ["青岛","东莞","上海"]
            },
            "age" : 20,
          "salary":NumberDecimal("6398.22"),
          "lenght" :1.77
};

var user5 =
{
            "username" : "peter",
            "country" : "UK",
            "address" : {
                    "aCode" : "411000",
                    "add" : "TEST"
          },
          "favorites" : {
                    "movies" : ["蜘蛛侠","钢铁侠","蝙蝠侠"],
                    "cites" : ["青岛","东莞","上海"]
          },
        "salary":NumberDecimal("1969.88")
};

db.users.insert(user1);
db.users.insert(user2);
db.users.insert(user3);
db.users.insert(user4);
db.users.insert(user5);
```

2.2.4.2. 查询

查询喜欢的城市包含东莞和东京的 user

```
    select * from users   where favorites.cites has "东莞"、"东京"
    db.users.find({ "favorites.cites" : { "$all" : [ "东莞" , "东京"]}}).pretty()
```

查询国籍为英国或者美国，名字中包含 s 的 user

```
    select * from users   where username like '%s%' and (country= English or country= USA)
   db.users.find({ "$and" : [ { "username" : { "$regex" : ".*s.*"}} , { "$or" : [ { "country" : "English"} , { "country" : "USA"}]}]}).pretty()
```

//思考 查询姓名是 deer 或者 james 的文档

2.2.4.3. 修改

把 lison 的年龄修改为 6 岁

```
    update   users   set age=6 where username = lison'
    db.users.updateMany({ "username" : "lison"},{ "$set" : { "age" : 6}})
```

//思考，又过了一年，lison 年龄又涨了一岁

喜欢的城市包含东莞的人，给他喜欢的电影加入"小电影 2""小电影 3"

```
    update users   set favorites.movies add "小电影 2 ", "小电影 3" where favorites.cites   has "东莞"
   db.users.updateMany({ "favorites.cites" : " 东 莞 "}, { "$addToSet" : { "favorites.movies" : { "$each" : [ "小电影 2 " , "小电影 3"]}}},true)
```

2.2.4.4. 删除

删除名字为 lison 的 user

```
    delete from users where username = 'lison'
  db.users.deleteMany({ "username" : "lison"} )
```

删除年龄大于 8 小于 25 的 user

```
    delete from users where age >8 and age <25
    db.users.deleteMany({"$and" : [ {"age" : {"$gt": 8}} , {"age" : {"$lt" : 25}}]})
```

2.2.4.5. 事务操作

- 事务操作

Lison 和 james 要完成一次事务操作，james 转账 1 给 lison

```
begin
  update   users   set lenght= lenght-1   where username = 'james'
  update   users   set lenght= lenght+1   where username = 'lison'
commit
```

```
db.users.find({"username": {"$in":["lison", "james"]}}).pretty();
```

```
s = db.getMongo().startSession()
s.startTransaction()

 db.users.update({"username" : "james"},{"$inc":{"lenght":-1}})
 db.users.update({"username" : "lison"},{"$inc":{"lenght":1}})

s.commitTransaction()
s.abortTransaction()
```

注：以上操作是错误的方式，事务操作一定要在集群的环境下才可以，方式如下

```
usersCollection .find({"username": {"$in":["lison", "james"]}}).pretty();
```

```
s = db.getMongo().startSession();

s.startTransaction()
usersCollection = s.getDatabase("lison").users
```

```
usersCollection.update({"username" : "james"},{"$inc":{"lenght":-1}})
usersCollection.update({"username" : "lison"},{"$inc":{"lenght":1}})

s.commitTransaction()
s.abortTransaction()
```

2.2.5. Java 客户端

2.2.5.1. 原始客户端

2.2.5.1.1. 引入 pom 文件

```
    <dependencies>
        <dependency>
            <groupId>org.mongodb</groupId>
            <artifactId>mongo-java-driver</artifactId>
            <version>3.11.2</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>
    </dependencies>
```

2.2.5.1.2. Document 方式

```
package cn.enjoy.mg;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.function.Consumer;

import org.bson.Document;
import org.bson.conversions.Bson;
import org.junit.Before;
import org.junit.Test;


import com.mongodb.MongoClient;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.result.DeleteResult;
import com.mongodb.client.result.UpdateResult;

import static com.mongodb.client.model.Updates.*;
import static com.mongodb.client.model.Filters.*;


//原生 java 驱动  document 的操作方式
public class QuickStartJavaDocTest {



    //数据库
    private MongoDatabase db;

    //文档集合
    private MongoCollection<Document> doc;

    //连接客户端（内置连接池）
    private MongoClient client;


    @Before
    public void init() {
        client = new MongoClient("192.168.244.123", 27017);
        db = client.getDatabase("lison");
        doc = db.getCollection("users");
    }

    @Test
    public void insertDemo() {
        Document doc1 = new Document();
        doc1.append("username", "cang");
        doc1.append("country", "USA");
        doc1.append("age", 20);
        doc1.append("lenght", 1.77f);
        doc1.append("salary", new BigDecimal("6565.22"));//存金额，使用 bigdecimal 这个数
据类型

        //添加“address”子文档
        Map<String, String> address1 = new HashMap<String, String>();
        address1.put("aCode", "0000");
        address1.put("add", "xxx000");
        doc1.append("address", address1);

        //添加“favorites”子文档，其中两个属性是数组
        Map<String, Object> favorites1 = new HashMap<String, Object>();
        favorites1.put("movies", Arrays.asList("aa", "bb"));
        favorites1.put("cites", Arrays.asList("东莞", "东京"));
        doc1.append("favorites", favorites1);

        Document doc2 = new Document();
        doc2.append("username", "Chen");
        doc2.append("country", "China");
        doc2.append("age", 30);
        doc2.append("lenght", 1.77f);
        doc2.append("salary", new BigDecimal("8888.22"));
        Map<String, String> address2 = new HashMap<>();
        address2.put("aCode", "411000");
        address2.put("add", "我的地址 2");
        doc2.append("address", address2);
        Map<String, Object> favorites2 = new HashMap<>();
        favorites2.put("movies", Arrays.asList("东游记", "一路向东"));
        favorites2.put("cites", Arrays.asList("珠海", "东京"));
        doc2.append("favorites", favorites2);

        //使用 insertMany 插入多条数据
        doc.insertMany(Arrays.asList(doc1, doc2));

    }

    @Test
    public void testFind() {
        final List<Document> ret = new ArrayList<>();
        //block 接口专门用于处理查询出来的数据
        Consumer<Document> printDocument = new Consumer<Document>() {
            @Override
            public void accept(Document document) {
                System.out.println(document);
```

```
                    ret.add(document);
                }
            };
            //select * from users   where favorites.cites has "东莞"、"东京"
            //db.users.find({ "favorites.cites" : { "$all" : [ "东莞" , "东京"]}})
            Bson all = all("favorites.cites", Arrays.asList("东莞", "东京"));//定义数据过滤器，喜欢
的城市中要包含"东莞"、"东京"
            FindIterable<Document> find = doc.find(all);

            find.forEach(printDocument);


            System.out.println("------------------>" + String.valueOf(ret.size()));
            ret.removeAll(ret);


            //select * from users   where username like '%s%' and (contry= English or contry =
USA)
            // db.users.find({ "$and" : [ { "username" : { "$regex" : ".*c.*"}} , { "$or" : [ { "country" :
"English"} , { "country" : "USA"}]}]})

            String regexStr = ".*c.*";
            Bson regex = regex("username", regexStr);//定义数据过滤器，username like '%s%'
            Bson or = or(eq("country", "English"), eq("country", "USA"));//定义数据过滤器，
(contry= English or contry = USA)
            Bson and = and(regex, or);
            FindIterable<Document> find2 = doc.find(and);
            find2.forEach(printDocument);
            System.out.println("------------------>" + String.valueOf(ret.size()));

        }

        @Test
        public void testUpdate() {
            //update   users   set age=6 where username = 'lison'
//          db.users.updateMany({ "username" : "lison"},{ "$set" : { "age" : 6}},true)

            Bson eq = eq("username", "cang");//定义数据过滤器，username = 'cang'
            Bson set = set("age", 8);//更新的字段.来自于 Updates 包的静态导入
            UpdateResult updateMany = doc.updateMany(eq, set);
            System.out.println("------------------>"                                      +
String.valueOf(updateMany.getModifiedCount()));//打印受影响的行数

            //update users   set favorites.movies add "小电影 2 ", "小电影 3" where favorites.cites
```

```
has "东莞"
		//db.users.updateMany({	"favorites.cites"	:	"	东	莞	"},	{	"$addToSet"	:
{ "favorites.movies" : { "$each" : [ "小电影 2 " , "小电影 3"]}}},true)

		Bson eq2 = eq("favorites.cites", "东莞");//定义数据过滤器，favorites.cites	has "东莞
"
		Bson addEachToSet = addEachToSet("favorites.movies", Arrays.asList("小电影 2 ", "小
电影 3"));//更新的字段.来自于 Updates 包的静态导入
		UpdateResult updateMany2 = doc.updateMany(eq2, addEachToSet);
		System.out.println("------------------>"											+
String.valueOf(updateMany2.getModifiedCount()));
	}

	@Test
	public void testDelete() {

		//delete from users where username = 'lison'
		//db.users.deleteMany({ "username" : "lison"} )
		Bson eq = eq("username", "lison");//定义数据过滤器，username='lison'
		DeleteResult deleteMany = doc.deleteMany(eq);
		System.out.println("------------------>"											+
String.valueOf(deleteMany.getDeletedCount()));//打印受影响的行数

		//delete from users where age >8 and age <25
		//db.users.deleteMany({"$and" : [ {"age" : {"$gt": 8}} , {"age" : {"$lt" : 25}}]})

		Bson gt = gt("age", 8);//定义数据过滤器，age > 8，所有过滤器的定义来自于 Filter
这个包的静态方法，需要频繁使用所以静态导入
//		Bson gt = Filter.gt("age",8);

		Bson lt = lt("age", 25);//定义数据过滤器，age < 25
		Bson and = and(gt, lt);//定义数据过滤器，将条件用 and 拼接
		DeleteResult deleteMany2 = doc.deleteMany(and);
		System.out.println("------------------>"											+
String.valueOf(deleteMany2.getDeletedCount()));//打印受影响的行数
	}


@Test
	public void testTransaction() {
//		begin
//		update	users	set lenght= lenght-1	where username = 'james'
//		update	users	set lenght= lenght+1	where username = 'lison'
//		commit
```

```
            ClientSession clientSession = client.startSession();
            clientSession.startTransaction();
            Bson eq = eq("username", "james");
            Bson inc = inc("lenght", -1);
            doc.updateOne(clientSession,eq,inc);

            Bson eq2 = eq("username", "lison");
            Bson inc2 = inc("lenght", 1);

            doc.updateOne(clientSession,eq2,inc2);

            clientSession.commitTransaction();
          // clientSession.abortTransaction();

      }

}
```

2.2.5.1.3. POJO 方式

新增 Favorites

```
package cn.enjoy.entity;

import java.util.List;

public class Favorites {
      private List<String> movies;
      private List<String> cites;
      public List<String> getMovies() {
            return movies;
      }
      public void setMovies(List<String> movies) {
            this.movies = movies;
      }
      public List<String> getCites() {
            return cites;
      }
      public void setCites(List<String> cites) {
            this.cites = cites;
      }
```

```
    @Override
    public String toString() {
        return "Favorites [movies=" + movies + ", cites=" + cites + "]";
    }

}
```

新增 Address

```
package cn.enjoy.entity;

public class Address {

  private String aCode;
  private String add;
  public String getaCode() {
        return aCode;
  }
  public void setaCode(String aCode) {
        this.aCode = aCode;
  }
  public String getAdd() {
        return add;
  }
  public void setAdd(String add) {
        this.add = add;
  }
  @Override
  public String toString() {
        return "Address [aCode=" + aCode + ", add=" + add + "]";
  }
}
```

新增 User

```
package cn.enjoy.entity;

import java.math.BigDecimal;

import org.bson.types.ObjectId;

public class User {

    private ObjectId id;
    private String username;

    private String country;

    private Address address;

    private Favorites favorites;

    private int age;

    private BigDecimal salary;

    private float lenght;


    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getCountry() {
        return country;
    }
    public void setCountry(String country) {
        this.country = country;
    }
    public Address getAddress() {
        return address;
    }
    public void setAddress(Address address) {
        this.address = address;
    }
    public Favorites getFavorites() {
        return favorites;
    }
    public void setFavorites(Favorites favorites) {
        this.favorites = favorites;
    }
    public ObjectId getId() {
        return id;
    }
    public void setId(ObjectId id) {
```

```
			this.id = id;
		}
		public int getAge() {
			return age;
		}
		public void setAge(int age) {
			this.age = age;
		}
		public BigDecimal getSalary() {
			return salary;
		}
		public void setSalary(BigDecimal salary) {
			this.salary = salary;
		}
		public float getLenght() {
			return lenght;
		}
		public void setLenght(float lenght) {
			this.lenght = lenght;
		}


		@Override
		public String toString() {
			return "User [id=" + id + ", username=" + username + ", country="
					+ country + ", address=" + address + ", favorites=" + favorites
					+ ", age=" + age + ", salary=" + salary + ", lenght=" + lenght +"]";
		}

	}
```

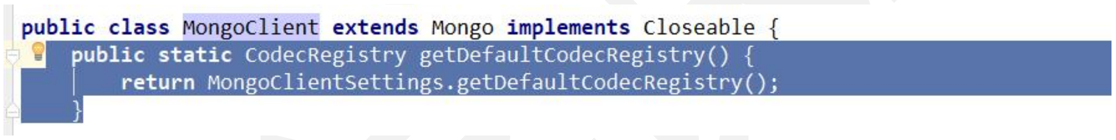
```
package cn.enjoy.mg;

import static com.mongodb.client.model.Updates.*;
import static com.mongodb.client.model.Filters.*;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.function.Consumer;
import org.bson.Document;
import org.bson.codecs.configuration.CodecRegistries;
import org.bson.codecs.configuration.CodecRegistry;
import org.bson.codecs.pojo.PojoCodecProvider;
import org.bson.conversions.Bson;
import org.junit.Before;
import org.junit.Test;


import cn.enjoy.entity.Address;
import cn.enjoy.entity.Favorites;
import cn.enjoy.entity.User;
import com.mongodb.MongoClient;
import com.mongodb.MongoClientOptions;
import com.mongodb.ServerAddress;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.Updates;
import com.mongodb.client.result.DeleteResult;
import com.mongodb.client.result.UpdateResult;

//原生 java 驱动  Pojo 的操作方式
public class QuickStartJavaPojoTest {



	private MongoDatabase db;

	private MongoCollection<User> doc;

	private MongoClient client;


	@Before
	public void init(){
		//编解码器的 list
		List<CodecRegistry> codecResgistes = new ArrayList<>();
		//list 加入默认的编解码器集合
		codecResgistes.add(MongoClient.getDefaultCodecRegistry());
		//生成一个 pojo 的编解码器
		CodecRegistry pojoCodecRegistry = CodecRegistries.
				fromProviders(PojoCodecProvider.builder().automatic(true).build());
```

```
        //list 加入 pojo 的编解码器
        codecResgistes.add(pojoCodecRegistry);
        //通过编解码器的 list 生成编解码器注册中心
        CodecRegistry registry = CodecRegistries.fromRegistries(codecResgistes);

        //把编解码器注册中心放入 MongoClientOptions
        //MongoClientOptions 相当于连接池的配置信息
        MongoClientOptions build = MongoClientOptions.builder().
                codecRegistry(registry).build();

        ServerAddress serverAddress = new ServerAddress("192.168.244.123", 27017);

        client = new MongoClient(serverAddress, build);
        db =client.getDatabase("lison");
        doc = db.getCollection("users",User.class);
}



@Test
public void insertDemo(){
        User user = new User();
        user.setUsername("cang");
        user.setCountry("USA");
        user.setAge(20);
        user.setLenght(1.77f);
        user.setSalary(new BigDecimal("6265.22"));

        //添加“address”子文档
        Address address1 = new Address();
        address1.setaCode("411222");
        address1.setAdd("sdfsdf");
        user.setAddress(address1);

        //添加“favorites”子文档，其中两个属性是数组
        Favorites favorites1 = new Favorites();
        favorites1.setCites(Arrays.asList("东莞","东京"));
        favorites1.setMovies(Arrays.asList("西游记","一路向西"));
        user.setFavorites(favorites1);



        User user1 = new User();
        user1.setUsername("chen");
        user1.setCountry("China");
        user1.setAge(30);
        user1.setLenght(1.77f);
        user1.setSalary(new BigDecimal("6885.22"));
        Address address2 = new Address();
        address2.setaCode("411000");
        address2.setAdd("我的地址 2");
        user1.setAddress(address2);
        Favorites favorites2 = new Favorites();
        favorites2.setCites(Arrays.asList("珠海","东京"));
        favorites2.setMovies(Arrays.asList("东游记","一路向东"));
        user1.setFavorites(favorites2);


        //使用 insertMany 插入多条数据
        doc.insertMany(Arrays.asList(user,user1));

    }


    @Test
    public void testFind(){

        final List<User> ret = new ArrayList<>();
        Consumer<User> printDocument = new Consumer<User>() {
            @Override
            public void accept(User t) {
                System.out.println(t.toString());
                ret.add(t);
            }

        };

        //select * from users    where favorites.cites has "东莞"、"东京"
        //db.users.find({ "favorites.cites" : { "$all" : [ "东莞" , "东京"]}})
        Bson all = all("favorites.cites", Arrays.asList("东莞","东京"));//定义数据过滤器，喜欢
的城市中要包含"东莞"、"东京"
        FindIterable<User> find = doc.find(all);
        find.forEach(printDocument);
        System.out.println("------------------>"+String.valueOf(ret.size()));
        ret.removeAll(ret);

        //select * from users    where username like '%s%' and (contry= English or contry =
USA)
        // db.users.find({ "$and" : [ { "username" : { "$regex" : ".*c.*"}} , { "$or" : [ { "country" :
```

```
"English"} , { "country" : "USA"}]}]})
            String regexStr = ".*c.*";
            Bson regex = regex("username", regexStr);//定义数据过滤器，username like '%s%'
            Bson or = or(eq("country","English"),eq("country","USA"));//定义数据过滤器，(contry=
English or contry = USA)
            FindIterable<User> find2 = doc.find(and(regex,or));
            find2.forEach(printDocument);
            System.out.println("------------------>"+String.valueOf(ret.size()));

      }


      @Test
      public void testUpdate(){
            //update    users    set age=6 where username = 'lison'
          //db.users.updateMany({ "username" : "lison"},{ "$set" : { "age" : 6}},true)
            Bson eq = eq("username", "lison");//定义数据过滤器，username = 'lison'
            Bson set = set("age", 8);//更新的字段.来自于 Updates 包的静态导入
            UpdateResult updateMany = doc.updateMany(eq, set);

      System.out.println("------------------>"+String.valueOf(updateMany.getModifiedCount()));//打
印受影响的行数

            //update users    set favorites.movies add "小电影 2 ", "小电影 3" where favorites.cites
has "东莞"
            //db.users.updateMany({ "favorites.cites" : "东莞"}, { "$addToSet" :
{ "favorites.movies" : { "$each" : [ "小电影 2 " , "小电影 3"]}}},true)
            Bson eq2 = eq("favorites.cites", "东莞");//定义数据过滤器，favorites.cites    has "东莞
"
            Bson addEachToSet = addEachToSet("favorites.movies", Arrays.asList( "小电影 2 ", "小
电影 3"));//更新的字段.来自于 Updates 包的静态导入
            UpdateResult updateMany2 = doc.updateMany(eq2, addEachToSet);

      System.out.println("------------------>"+String.valueOf(updateMany2.getModifiedCount()));
      }

      @Test
      public void testDelete(){

            //delete from users where username = ‘lison’
            //db.users.deleteMany({ "username" : "lison"} )
            Bson eq = eq("username", "lison");//定义数据过滤器，username='lison'
            DeleteResult deleteMany = doc.deleteMany(eq);
            System.out.println("------------------>"+String.valueOf(deleteMany.getDeletedCount()));//
```

```
打印受影响的行数

            //delete from users where age >8 and age <25
            //db.users.deleteMany({"$and" : [ {"age" : {"$gt": 8}} , {"age" : {"$lt" : 25}}]})
            Bson gt = gt("age",8);//定义数据过滤器，age > 8，所有过滤器的定义来自于 Filter
这个包的静态方法，需要频繁使用所以静态导入

            Bson lt = lt("age",25);//定义数据过滤器，age < 25
            Bson and = and(gt,lt);//定义数据过滤器，将条件用 and 拼接
            DeleteResult deleteMany2 = doc.deleteMany(and);

      System.out.println("------------------>"+String.valueOf(deleteMany2.getDeletedCount()));//打
印受影响的行数
      }



}
```

com.mongodb.MongoClient



```
public class MongoClient extends Mongo implements Closeable {
    public static CodecRegistry getDefaultCodecRegistry() {
        return MongoClientSettings.getDefaultCodecRegistry();
    }
```