**MongoDB架构及基本操作说明**

当前架构 双服务器架构

Secondary1

Arbiter1

Primary2

ConfigSvr

Mongos

Primary1

Secondary2

Arbiter2

ConfigSvr

Mongos

Primary

Secondary

Arbiter

ConfigSvr

Mongos

当前架构为单shard+replica Set模式，双服务器为双Shard+Replica Set模式。同一个Shard中的primary和Secondary存储内容一致。而双Shard则是两个Shard分布式存储不同数据，备份由shard内部进行。

双服务器中的两个Shard各含一个primary ，一个secondary，和一个arbiter（arbiter的唯一作用是在primary 宕机后选举新的primary时拥有投票权，用以使存活节点数大于50%，不包括50%，否则系统将整个down掉，以及在票数相同的情况下用以打破选举的平衡，并不存储和读取数据）。

因为同一个shard中，只有primary可以用以写，secondary只是用于对primary节点的备份并用于读操作，然后再primary宕机的情况下接管它的工作。所以，双shard模式下，两个服务器分别包含一个primary，而且同一个shard的arbiter必须和secondary在一个服务器上。这样子既保证了两个服务器都可以进行读、写操作，而且在primary down的时候也能够继续使得选取成功secondary。

后续扩展时，可以再在集群中添加新的shard，然后与老的shard进行balance均衡操作。

当前架构启动顺序代码：

1. 先启动primary，secondary，arbiter这三个节点，linux中命令分别为：

./mongod --replSet SetName --dbpath DbDir --logpath LogDir/mongod.log --port PortNum

这里的SetName是你为建立的Replica Set所取的名字，因为三个节点处于同一个set，所以SetName是一致的。dbpath是该节点上用于存储mongoDB数据的目录。Logpath为存储日志的地址，此处需要是一个.log的文档文件，而不是一个文件夹目录。Port是为这个mongod连接打开的端口号。

1. 启动mongo连接其中一个节点，进行Set的初始化操作

./mongo localhost:PortNum/admin

db.runCommand({“replSetInitiate”:{“\_id”:”SetName”,”members”:[{“\_id”:1,”host”:”localhost:PortNum1”},{“\_id”:2,”host”:”localhost:PortNum2”},{“\_id”:3,”host”:”localhost:PortNum3”,”arbiterOnly”:true}]}})

1. 启动ConfigSvr

./mongod --dbpath DbDir --logpath LogDir --port PortNum

1. 启动mongos

./mongos --port PortNum --configdb localhost:ConfigSvrPort

1. 启动客户端进行整个架构的初始化操作

./mongo SvrIP:mongosPort/admin

db.runCommand({addshard:”SetName/localhost:PortNum1,localhost:PortNum2”,name:”ShardName”})

(ShardName是此时才定义的，此处addshard操作后面的节点只需要Set中的Standard节点就行，不需要写入arbiter)

1. 开始存储操作，在新加入一个数据库时，需要在admin库下进行操作，才能使得它为分布式存储

use admin

db.runCommand({enablesharding:”newDB”})

db.runCOmmand({shardcollection:’DB.Collection’,key:{\_id:1}})

第一个指令为将newDB进行分布式，第二个指令为分别指定DB中的各个Collection的分布式存储的依据，默认为根据\_id来进行

当前的所有节点都位于服务器192.168.1.113上，其中mongos端口为27050，primary为27051，Secondary为27052，Arbiter为27053，ConfigSvr为27054。目前只开放了27050端口，其他端口都只能从服务器上localhost访问。

因为目前的1.18.1的MongoDB并不支持Shard+Replica Set模式的用户安全设置功能，所以此处并无设置。预计于下个月发布的1.19.1版本可能会加入这一功能，但要具体完善，初步估计需要到1.20版。

**MongoDB操作**

1. 自增长ID：

在原有Collection之外，新建一个Collection，用以存储自增长的ID，只需要一条Document就行。每次在lky\_form插入新的Document时，从这个Collection中FindAndModify一下他原有的数据，这是一个原子操作，便避免了并发性问题的可能。

具体代码如下：  
命令行版：

userId = db.counter.findAndModify({update:{$inc:{‘count’:1},query:{name:”counter”},new:true}})

a={“\_id”:userId.count,…}

db.CollectionName.insert(a)

java版：

DBObject modifier =new BasicDBObject("counter",1);DBObject incQuery = new BasicDBObject("$inc",modifier);DBObject user=number.findAndModify(num0, incQuery); //此处的num0为我设置的存入新的表单的那个Document的名字DBObject newuser = new BasicDBObject();newuser.put("\_id", user.get("counter"));

1. 模糊查询

使用正则表达式进行模糊查询

命令行版：

db.CollectionName.find({“tag”:/good/})

//此命令行表示查询tag中包含了good字符的Document

或者也可以写为：  
db.CollectionName.find({“tag”:{$regex:”good”}})

Java版：

import java.util.regex.Pattern;

DBCursor cur ;

cur = coll.find(new BasicDBObject().append("tag",Pattern.*compile*("good",Pattern.*CASE\_INSENSITIVE*)));

**附录：**

**MongoDB Java Driver 简单操作**

**一、Java驱动一致性**

 MongoDB的Java驱动是线程安全的，对于一般的应用，只要一个Mongo实例即可，Mongo有个内置的连接池（池大小默认为10个）。

 对于有大量写和读的环境中，为了确保在一个Session中使用同一个DB时，我们可以用以下方式保证一致性：

DB mdb = mongo.getDB('dbname');

 mdb.requestStart();  
 //  
 // 业务代码  
 //  
 mdb.requestDone();

 DB和DBCollection是绝对线程安全的，它们被缓存起来了，所以在应用中取到的可能是同一个对象。

**二、保存/查找对象(DBObject)**

 Java驱动提供了DBObject接口，方便我们保存对象到数据库中。  
   
 定义需要保存的对象：  
   
 public class Tweet implements DBObject {  
  /\*\* ...... \*/  
 }  
   
 然后我们可以使用该对象：  
   
 Tweet tweet = new Tweet();  
 tweet.put("user", userId);  
 tweet.put("message", message);  
 tweet.put("date", new Date());  
   
 collection.insert(tweet);  
   
 当从数据库中查询时，结果会自动的转换成DBObject对象，我们可以转换成我们自己的类型：  
   
 collection.setObjectClass(Tweet);  
   
 Tweet myTweet = (Tweet)collection.findOne();

**三、创建连接**

 Mongo m = new Mongo();  
 Mongo m = new Mongo("localhost");  
 Mongo m = new Mongo("localhost", 27017);  
   
 DB db = m.getDB("mydb);  
   
 注意：事实上，Mongo实例代表了一个数据库连接池，即使在多线程的环境中，一个Mongo实例对我们来说已经足够了。

**四、认证（可选的）**

 boolean auth = db.authenticate("myUserName", "myPasswd");

   
**五、取得Collection列表**

 Set<String> colls = db.getCollectionNames();  
   
 for(String s : colls) {  
  System.out.prinln(s);  
 }

**六、获取一个Collection**

 DBCollection coll = db.getCollection("testCollection");  
   
 使用DBCollection，我们可以进行插入、查询数据等数据操作。

**七、插入文档**

 假设有个JSON文档如下所示：  
   
 {  
  "name": "MongoDB",  
  "type": "database",  
  "count": 1,  
  "info": {  
     x: 203,  
     y: 102  
    }  
 }  
   
 注意：上面的JSON文档有个内嵌文档"info"。  
   
 我们完全可以利用BasicDBObject来创建一个和上面的JSON一样的文档，并且把它保存在MongoDB中。  
   
 DBObject doc = new BasicDBObject();  
   
 doc.put("name", "MongoDB");  
 doc.put("type", "database");  
 doc.put("count", 1);  
   
 DBObject info = new BasicDBObject();  
 info.put("x", 203);  
 info.put("y", 102);  
   
 doc.put("info", info);  
   
 coll.insert(doc);

**八、查询第一个文档（findOne()）**

 为了验证在上面我们保存的类似JSON的数据，我们可以用findOne()方法取得数据。  
   
 findOne(): 返回一个文档；  
 find(): 返回一个游标（DBCursor），其中包含一组对象DBObject；  
   
 DBObject doc = coll.findOne();  
 System.out.println(doc);  
   
 我们将会看到控制台输出：  
 { "\_id" : "49902cde5162504500b45c2c" , "name" : "MongoDB" , "type" : "database" , "count" : 1 , "info" : { "x" : 203 , "y" : 102} , "\_ns" : "testCollection"}

**九、插入多个文档**

 为了在后来展示更多的查询方法，我们先插入几个文档，它们的JSON像这样：  
 {  
  "i": value  
 }  
   
 使用一个循环插入数据：  
   
 for(int i = 0; i < 100; i++) {  
  coll.insert(new BasicDBObject().append("i", i));  
 }  
   
 我们注意到，同一个coll，我们完全可以插入不同风格的数据，这就是MongoDB的重要特性“模式自由”。

**十、统计文档数**

 现在我们已经有101份文档在数据库中了，现在统计一下看是否正确。  
   
 long count = coll.getCount();  
 System.out.println(count);  
   
 控制台将会输出：101

**十一、使用游标取得所有的文档**

 DBCursor cursor = coll.find();  
   
 while(cursor.hasNext()) {  
  DBObject object = cursor.next();  
  System.out.println(object);  
 }  
   
**十二、查询单个文档**

 DBObject query = new BasicDBObject();  
   
 query.put("i", 71);  
   
 cursor = coll.find(query);  
   
 while(cur.hasNext()) {  
  DBObject object = cursor.next();  
  System.out.println(object);  
 }  
   
 控制台的输出类似如下：  
   
 { "\_id" : "49903677516250c1008d624e" , "i" : 71 , "\_ns" : "testCollection"}

**十三、查询文档集合**

 根据查询条件，我们可以通过DBCollection从数据库中取出多个对象，比如查询i>50的文档集合：  
   
 query = new BasicDBObject();  
   
 query.put("i", new BasicDBObject("$gt", 50)); // i>50  
   
 cursor = coll.find(query);  
   
 while(cursor.hasNext()) {  
  DBObject object = cursor.next();  
  System.out.println(object);  
 }  
   
 比如查询条件为 20<i<=30：  
   
 query = new BasicDBObject();  
   
 // 20<i<=30  
 query.put("i", new BasicDBObject("$gt", 20).append("$lte", 30));  
   
 cursor = coll.find(query);  
   
 while(cursor.hasNext()) {  
  DBObject object = cursor.next();  
  System.out.println(object);  
 }

**十四、创建索引**

 MongoDB支持索引，并且给一个DBCollection添加索引非常简单，你只要指明需要创建索引的字段，然后指明其是升序(1)还是降序(-1)即可，比如在"i"上创建升序索引。  
   
 coll.createIndex(new BasicDBObject("i", 1)); // 1代表升序

**十五、查询索引**

 我们可以查询到所有的索引：  
   
 List<DBObject> list = coll.getIndexInfo();  
   
 for(DBObject index : list)｛  
  System.out.println(index);  
 }  
   
 控制台的输出类似如下所示：  
   
 { "name" : "i\_1" , "ns" : "mydb.testCollection" , "key" : { "i" : 1} , "\_ns" : "system.indexes"}

   **MongoDB的管理功能**     
**一、获取所有的数据库**

 Mongo m = new Mongo();  
   
 for(String s : m.getDatabaseNames()) {  
  System.out.println(s);  
 }

**二、删除数据库**

 m.dropDatabase("my\_new\_db");  
   
   
   **MongoDB的Java类型**

**一、对象ID**

 ObjectId被用作自动生成的唯一ID.  
   
 ObjectId id = new ObjectId();  
 ObjectId copy = new ObjectId(id);  
   
**二、正则表达式**

 Pattern john = Pattern.compile("joh?n", CASE\_INSENSITIVE);  
 DBObject query = new BasicDBObject("name", john);  
   
 // 查询所有 "name" 匹配 /joh?n/i 的文档  
 DBCursor cursor = collection.find(query);

**三、日期和时间**

 Date now = new Date();  
 DBObject time = new BasicDBObject("ts", now);  
   
 collection.save(time);  
   
**四、数据库引用**

 DBRef可以用来保存数据库引用。  
   
 DBRef addressRef = new DBRef(db, "foo.bar", address\_id);  
 DBObject address = addressRef.fetch();  
   
 DBObject person = BasicDBObjectBuilder.start()  
  .add("name", "Fred")  
  .add("address", addressRef)  
  .get();  
 collection.save(person);  
   
 DBObject fred = collection.findOne();  
 DBRef addressObj = (DBRef)fred.get("address");  
 addressObj.fetch();  
   
**五、二进制数据**

 字节数组(byte[])被当作二进制数据。  
   
**六、内嵌文档**

 JSON样式的数据如下：  
 {  
  "x": {  
   "y": 3  
  }  
 }  
   
 则在MongoDB中，Java表示为：  
   
 DBObject y = new BasicDBObject("y", 3);  
 DBObject x = new BasicDBObject("x", y);  
**七、数组**

 任何继承自List的对象，在MongoDB中，都被当成是数组。  
   
 如果想表示如下JSON数据：  
   
 {  
  "x": [  
   1,  
   2,  
   {"foo": "bar"},  
   4  
  ]  
 }  
   
 则在Java中，应该为：  
   
 List<Object> x = new ArrayList<Object>();  
 x.add(1);  
 x.add(2);  
 x.add(new BasicDBObject("foo", "bar"));  
 x.add(4);  
   
 DBObject doc = new BasicDBObject("x", x);  
 System.out.println(doc);