**[Java Mongodb Mapreduce Example](http://www.pretechsol.com/2012/09/java-mongodb-mapreduce-example.html)**

23:14

[VINOD KUMARAN](https://plus.google.com/107730448860014080523)

**Map Reduce**

MapReduce is a software framework introduced by Google in 2004 to support distributed computing on large data sets on clusters of computers, [more about Map reduce](http://docs.mongodb.org/manual/core/map-reduce/).

Here is one simple example of data aggregation using via a java program.

**Goals**

1. Insert Customer records in to Mongodb

2. Categorize Customer as Major or Minor based on age and count the number of customers

**Steps**

Create a Maven project and add below dependency in pom.xml

<dependency>

<groupId>org.mongodb</groupId>

<artifactId>mongo-java-driver</artifactId>

<version>2.5.2</version>

</dependency>

Create a Java class to insert Customer details and test MapRedeuce. You can download java mongodb driver from <https://github.com/mongodb/mongo-java-driver/downloads>

package com.pretech;

import com.mongodb.BasicDBObject;

import com.mongodb.DB;

import com.mongodb.DBCollection;

import com.mongodb.DBCursor;

import com.mongodb.DBObject;

import com.mongodb.MapReduceCommand;

import com.mongodb.MapReduceOutput;

import com.mongodb.Mongo;

public class PretechMapReduceExample {

public static void main(String[] args) {

Mongo mongo;

try {

mongo = new Mongo("localhost", 27017);

DB db = mongo.getDB("library");

DBCollection pretechCustomer = db.getCollection("CustomerDetails");

BasicDBObject customer = new BasicDBObject();

customer.put("name", "RAM");

customer.put("age", 20);

pretechCustomer.insert(customer);

customer = new BasicDBObject();

customer.put("name", "CHANDRAN");

customer.put("age", 26);

pretechCustomer.insert(customer);

customer = new BasicDBObject();

customer.put("name", "VINOD");

customer.put("age", 24);

pretechCustomer.insert(customer);

customer = new BasicDBObject();

customer.put("name", "SRIKUMAR");

customer.put("age", 30);

pretechCustomer.insert(customer);

customer = new BasicDBObject();

customer.put("name", "SURAJ");

customer.put("age", 12);

pretechCustomer.insert(customer);

customer.put("name", "REEJU");

customer.put("age", 18);

pretechCustomer.insert(customer);

customer = new BasicDBObject();

customer.put("name", "VINITHA");

customer.put("age", 32);

pretechCustomer.insert(customer);

customer = new BasicDBObject();

customer.put("name", "NIRMAL");

customer.put("age", 23);

pretechCustomer.insert(customer);

customer = new BasicDBObject();

customer.put("name", "RAGHAV");

customer.put("age", 10);

pretechCustomer.insert(customer);

customer = new BasicDBObject();

customer.put("name", "SAYOOJ");

customer.put("age", 33);

pretechCustomer.insert(customer);

System.out.println("Total records in Customer collections");

// Retrieving collection details

DBCursor cursorDoc = pretechCustomer.find();

while (cursorDoc.hasNext()) {

System.out.println("Customer details " + cursorDoc.next());

}

String map = "function() { " + "var category; "

+ "if ( this.age >= 21 ) " + "category = 'Major'; "

+ "else " + "category = 'Minor'; "

+ "emit(category, {name: this.name});}";

String reduce = "function(key, values) { " + "var sum = 0; "

+ "values.forEach(function(doc) { " + "sum += 1; " + "}); "

+ "return {pretechCustomer: sum};} ";

MapReduceCommand cmd = new MapReduceCommand(pretechCustomer, map,

reduce, null, MapReduceCommand.OutputType.INLINE, null);

MapReduceOutput out = pretechCustomer.mapReduce(cmd);

System.out.println("Mapreduce results");

for (DBObject o : out.results()) {

System.out.println(o.toString());

}

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

Start mongodb and run the program (See this example [How to start Mongodb](http://www.pretechsol.com/2012/09/how-to-set-up-mongodb-in-windows.html#.UgR4J40juDR))

Output

Total records in Customer collections  
Customer details { "\_id" : { "$oid" : "520462aa9709075fca9bc655"} , "name" : "RAM" , "age" : 20}  
Customer details { "\_id" : { "$oid" : "520462aa9709075fca9bc656"} , "name" : "CHANDRAN" , "age" : 26}  
Customer details { "\_id" : { "$oid" : "520462aa9709075fca9bc657"} , "name" : "VINOD" , "age" : 24}  
Customer details { "\_id" : { "$oid" : "520462aa9709075fca9bc658"} , "name" : "SRIKUMAR" , "age" : 30}  
Customer details { "\_id" : { "$oid" : "520462aa9709075fca9bc659"} , "name" : "SURAJ" , "age" : 12}  
Customer details { "\_id" : { "$oid" : "520462aa9709075fca9bc65a"} , "name" : "VINITHA" , "age" : 32}  
Customer details { "\_id" : { "$oid" : "520462aa9709075fca9bc65b"} , "name" : "NIRMAL" , "age" : 23}  
Customer details { "\_id" : { "$oid" : "520462aa9709075fca9bc65c"} , "name" : "RAGHAV" , "age" : 10}  
Customer details { "\_id" : { "$oid" : "520462aa9709075fca9bc65d"} , "name" : "SAYOOJ" , "age" : 33}  
Mapreduce results  
{ "\_id" : "Major" , "value" : { "pretechCustomer" : 6.0}}  
{ "\_id" : "Minor" , "value" : { "pretechCustomer" : 3.0}}

**5. Inserting data to MongoDB.**

Let’s first create two books with the following commands.

|  |  |
| --- | --- |
| 1 | > book1 = {name : "Understanding JAVA", pages : 100} |
| 2 | > book2 = {name : "Understanding JSON", pages : 200} |

Now, let’s insert these two books in to a collection called books.

|  |  |
| --- | --- |
| 1 | > db.books.save(book1) |
| 2 | > db.books.save(book2) |

The above two statements will create a collection called books under the database library. Following statement will list out the two books which we just saved.

|  |  |  |
| --- | --- | --- |
| 1 | > db.books.find(); | |
| 2 |  |

|  |  |
| --- | --- |
| 3 | { "\_id" : ObjectId("4f365b1ed6d9d6de7c7ae4b1"), "name" : "Understanding JAVA", "pages" : 100 } |
| 4 | { "\_id" : ObjectId("4f365b28d6d9d6de7c7ae4b2"), "name" : "Understanding JSON", "pages" : 200 } |

Let’s add few more records.

|  |  |  |
| --- | --- | --- |
| 1 | > book = {name : "Understanding XML", pages : 300} | |
| 2 | > db.books.save(book) |

|  |  |  |
| --- | --- | --- |
| 3 | > book = {name : "Understanding Web Services", pages : 400} | |
| 4 | > db.books.save(book) |

|  |  |  |
| --- | --- | --- |
| 5 | > book = {name : "Understanding Axis2", pages : 150} | |
| 6 | > db.books.save(book) |

**6. Writing the Map function**

Let’s process this library collection in a way that, we need to find the number of books having pages less 250 pages and greater than that.

|  |  |  |
| --- | --- | --- |
| 1 | > var map = function() { | |
| 2 | var category; |

|  |  |
| --- | --- |
| 3 | if ( this.pages >= 250 ) |
| 4 | category = 'Big Books'; |

|  |  |
| --- | --- |
| 5 | else |
| 6 | category = "Small Books"; | |

|  |  |  |
| --- | --- | --- |
| 7 | emit(category, {name: this.name}); | |
| 8 | }; |

Here, the collection produced by the Map function will have a collection of following members.

|  |  |
| --- | --- |
| 1 | {"Big Books",[{name: "Understanding XML"}, {name : "Understanding Web Services"}]); |
| 2 | {"Small Books",[{name: "Understanding JAVA"}, {name : "Understanding JSON"},{name: "Understanding Axis2"}]); |

**7. Writing the Reduce function.**

|  |  |  |
| --- | --- | --- |
| 1 | > var reduce = function(key, values) { | |
| 2 | var sum = 0; |

|  |  |  |
| --- | --- | --- |
| 3 | values.forEach(function(doc) { | |
| 4 | sum += 1; |

|  |  |
| --- | --- |
| 5 | }); |
| 6 | return {books: sum}; | |

|  |  |
| --- | --- |
| 7 | }; |

**8. Running MapReduce against the books collection.**

|  |  |  |
| --- | --- | --- |
| 1 | > var count  = db.books.mapReduce(map, reduce, {out: "book\_results"}); | |
| 2 | > db[count.result].find() |

|  |  |
| --- | --- |
| 3 |  |
| 4 | { "\_id" : "Big Books", "value" : { "books" : 2 } } | |

|  |  |
| --- | --- |
| 5 | { "\_id" : "Small Books", "value" : { "books" : 3 } } |

The above says, we have 2 Big Books and 3 Small Books.

Everything done above using the MongoDB shell, can be done with Java too. Following is the Java client for it. You can download the required dependent jar from [here](http://people.wso2.com/%7Eprabath/facilelogin/mongo.jar).

|  |  |  |
| --- | --- | --- |
| 01 | import com.mongodb.BasicDBObject; | |
| 02 | import com.mongodb.DB; |

|  |  |  |
| --- | --- | --- |
| 03 | import com.mongodb.DBCollection; | |
| 04 | import com.mongodb.DBObject; |

|  |  |  |
| --- | --- | --- |
| 05 | import com.mongodb.MapReduceCommand; | |
| 06 | import com.mongodb.MapReduceOutput; |

|  |  |  |
| --- | --- | --- |
| 07 | import com.mongodb.Mongo; | |
| 08 |  |

|  |  |  |
| --- | --- | --- |
| 09 | public class MongoClient { | |
| 10 |  |

|  |  |
| --- | --- |
| 11 | /\*\* |
| 12 | \* @param args | |

|  |  |
| --- | --- |
| 13 | \*/ |
| 14 | public static void main(String[] args) { | |

|  |  |
| --- | --- |
| 15 |  |
| 16 | Mongo mongo; | |

|  |  |
| --- | --- |
| 17 |  |
| 18 | try { | |

|  |  |  |
| --- | --- | --- |
| 19 | mongo = new Mongo("localhost", 27017); | |
| 20 | DB db = mongo.getDB("library"); |

|  |  |
| --- | --- |
| 21 |  |
| 22 | DBCollection books = db.getCollection("books"); | |

|  |  |
| --- | --- |
| 23 |  |
| 24 | BasicDBObject book = new BasicDBObject(); | |

|  |  |  |
| --- | --- | --- |
| 25 | book.put("name", "Understanding JAVA"); | |
| 26 | book.put("pages", 100); |

|  |  |  |
| --- | --- | --- |
| 27 | books.insert(book); | |
| 28 |  |

|  |  |
| --- | --- |
| 29 | book = new BasicDBObject(); |
| 30 | book.put("name", "Understanding JSON"); | |

|  |  |  |
| --- | --- | --- |
| 31 | book.put("pages", 200); | |
| 32 | books.insert(book); |

|  |  |
| --- | --- |
| 33 |  |
| 34 | book = new BasicDBObject(); | |

|  |  |  |
| --- | --- | --- |
| 35 | book.put("name", "Understanding XML"); | |
| 36 | book.put("pages", 300); |

|  |  |  |
| --- | --- | --- |
| 37 | books.insert(book); | |
| 38 |  |

|  |  |
| --- | --- |
| 39 | book = new BasicDBObject(); |
| 40 | book.put("name", "Understanding Web Services"); | |

|  |  |  |
| --- | --- | --- |
| 41 | book.put("pages", 400); | |
| 42 | books.insert(book); |

|  |  |
| --- | --- |
| 43 |  |
| 44 | book = new BasicDBObject(); | |

|  |  |  |
| --- | --- | --- |
| 45 | book.put("name", "Understanding Axis2"); | |
| 46 | book.put("pages", 150); |

|  |  |  |
| --- | --- | --- |
| 47 | books.insert(book); | |
| 48 |  |

|  |  |  |
| --- | --- | --- |
| 49 | String map = "function() { "+ | |
| 50 | "var category; " + |

|  |  |  |
| --- | --- | --- |
| 51 | "if ( this.pages >= 250 ) "+ | |
| 52 | "category = 'Big Books'; " + |

|  |  |
| --- | --- |
| 53 | "else " + |
| 54 | "category = 'Small Books'; "+ | |

|  |  |  |
| --- | --- | --- |
| 55 | "emit(category, {name: this.name});}"; | |
| 56 |  |

|  |  |  |
| --- | --- | --- |
| 57 | String reduce = "function(key, values) { " + | |
| 58 | "var sum = 0; " + |

|  |  |  |
| --- | --- | --- |
| 59 | "values.forEach(function(doc) { " + | |
| 60 | "sum += 1; "+ |

|  |  |
| --- | --- |
| 61 | "}); " + |
| 62 | "return {books: sum};} "; | |

|  |  |
| --- | --- |
| 63 |  |
| 64 | MapReduceCommand cmd = new MapReduceCommand(books, map, reduce, | |

|  |  |  |
| --- | --- | --- |
| 65 | null, MapReduceCommand.OutputType.INLINE, null); | |
| 66 |  |

|  |  |  |
| --- | --- | --- |
| 67 | MapReduceOutput out = books.mapReduce(cmd); | |
| 68 |  |

|  |  |
| --- | --- |
| 69 | for (DBObject o : out.results()) { |
| 70 | System.out.println(o.toString()); | |

|  |  |
| --- | --- |
| 71 | } |
| 72 | } catch (Exception e) { | |

|  |  |  |
| --- | --- | --- |
| 73 | // TODO Auto-generated catch block | |
| 74 | e.printStackTrace(); |

|  |  |  |
| --- | --- | --- |
| 75 | } | |
| 76 | } |

|  |  |
| --- | --- |
| 77 | } |

***Reference:*** [MapReduce with MongoDB](http://blog.facilelogin.com/2012/02/mapreduce-with-mongodb.html) from our [JCG partner](http://www.javacodegeeks.com/p/jcg.html) Prabath Siriwardena at the [Facile Login](http://blog.facilelogin.com/) blog.

import com.mongodb.BasicDBObject;

import com.mongodb.DBObject;

import com.mongodb.DB;

import com.mongodb.DBCollection;

import com.mongodb.MongoClient;

import java.io.IOException;

import java.io.InputStream;

public class MongoSampleClient {

    public static void main(String[] args) {

        try {

            // create a MongoClient by connecting to the MongoDB instance in localhost

            MongoClient mongoClient = new MongoClient("localhost", 27017);

            // drop database if it already exists

            mongoClient.dropDatabase("sample");

            // creating a db named "sample" and a collection named "book"

            DB db = mongoClient.getDB("sample");

            DBCollection bookCollection = db.getCollection("book");

            // insert the 3 pages of the book into the collection

            for (int i = 1; i < 4; i++) {

                BasicDBObject doc = new BasicDBObject("pageId", "page" + i).

                        append("content", readFile("page" + i + ".txt"));

                bookCollection.insert(doc);

            }

            // read the first doc to make sure that we've inserted correctly

            DBObject firstDoc = bookCollection.findOne();

            System.out.println(firstDoc);

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

    /\*\*

     \* Reads the specified file from classpath

     \*/

    private static String readFile(String fileName) throws IOException {

        // get the input stream

        InputStream fileStream = MongoSampleClient.class.getResourceAsStream("/" + fileName);

        // create a buffer with some default size

        byte[] buffer = new byte[8192 \* 2];

        // read the stream into the buffer

        int size = fileStream.read(buffer);

        // create a string for the needed size and return

        return new String(buffer, 0, size);

    }

}

**Step 4 :** Finally you can see the content you inserted above through the MongoDB console by using following commands.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | isuru@isuru-w520:~$ mongo    MongoDB shell version: 2.0.4  connecting to: test  >  > use sample  switched to db sample  >  > db.book.find()  { "\_id" : ObjectId("519f6c1f44ae9aea2881672a"), "pageId" : "page1", "content" : "your page1 content" }  { "\_id" : ObjectId("519f6c1f44ae9aea2881672b"), "pageId" : "page2", "content" : "your page2 content" }  { "\_id" : ObjectId("519f6c1f44ae9aea2881672c"), "pageId" : "page3", "content" : "your page3 content" }  > |

That’s it. In the next post on MonogoDB we’ll be looking at how to use MongoDB Map-Reduce functionality on top of the “book” collection we created above.

|  |
| --- |
| package sample.mongo;    import com.mongodb.DB;  import com.mongodb.DBCollection;  import com.mongodb.MapReduceCommand;  import com.mongodb.MongoClient;    import java.io.IOException;  import java.io.InputStream;    public class WordCount {        public static void main(String[] args) {          try {              // create a MongoClient by connecting to the MongoDB instance in localhost              MongoClient mongoClient = new MongoClient("localhost", 27017);              // access the db named "sample"              DB db = mongoClient.getDB("sample");              // access the input collection              DBCollection collection = db.getCollection("book");              // read Map file              String map = readFile("wc\_map.js");              // read Reduce file              String reduce = readFile("wc\_reduce.js");              // execute MapReduce on the input collection and direct the result to "wordcounts" collection              collection.mapReduce(map, reduce, "wordcounts", MapReduceCommand.OutputType.REPLACE, null);          } catch (Exception e) {              e.printStackTrace();          }      }        /\*\*       \* Reads the specified file from classpath       \*/      private static String readFile(String fileName) throws IOException {          // get the input stream          InputStream fileStream = WordCount.class.getResourceAsStream("/" + fileName);          // create a buffer with some default size          byte[] buffer = new byte[8192];          // read the stream into the buffer          int size = fileStream.read(buffer);          // create a string for the needed size and return          return new String(buffer, 0, size);      }  } |

In order the execute the above code, make sure you have “wc\_map.js” and “wc\_reduce.js” files on your project classpath containing above map and reduce functions. In the above Java code, first we connect to our MongoDB database and get a reference to the “book” collection. Then we read our map and reduce functions as a String from the classpath. Finally we execute the “mapReduce()” method on our input collection. This will apply our map and reduce functions on the “book” collection and store the output in a new collection called “wordcounts”. If there’s an already existing “wordcounts” collection, it will be replaced by the new one. If you need more details on “mapReduce()” method, please have a look at the [documentation](http://docs.mongodb.org/manual/reference/method/db.collection.mapReduce/#db.collection.mapReduce) and [java doc](http://api.mongodb.org/java/2.6/com/mongodb/DBCollection.html#mapReduce%28java.lang.String,%20java.lang.String,%20java.lang.String,%20com.mongodb.MapReduceCommand.OutputType,%20com.mongodb.DBObject%29).

Finally let’s log into our MongoDB console and output collection “wordcounts”.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | isuru@isuru-w520:~$ mongo  MongoDB shell version: 2.0.4  connecting to: test  > use sample  switched to db sample  >  >  > db.wordcounts.find()  { "\_id" : { "word" : "1930s" }, "value" : { "count" : 1 } }  { "\_id" : { "word" : "A" }, "value" : { "count" : 5 } }  { "\_id" : { "word" : "After" }, "value" : { "count" : 3 } }  ... |

In the MangoDB shell type the following...

[view plainprint?](http://blog.facilelogin.com/2012/02/mapreduce-with-mongodb.html)

1. > use library

The above is supposed to create a database called 'library'.  
  
Now to see whether your database been created, just type the following - which is supposed to list all the databases.

[view plainprint?](http://blog.facilelogin.com/2012/02/mapreduce-with-mongodb.html)

1. > show dbs;

You will notice that the database that you just created is not listed there. The reason is, MongoDB creates databases on-demand. It will get created only when we add something to it.  
  
5. Inserting data to MongoDB.  
  
Let's first create two books with the following commands.

[view plainprint?](http://blog.facilelogin.com/2012/02/mapreduce-with-mongodb.html)

1. > book1 = {name : "Understanding JAVA", pages : 100}
2. > book2 = {name : "Understanding JSON", pages : 200}

Now, let's insert these two books in to a collection called books.

[view plainprint?](http://blog.facilelogin.com/2012/02/mapreduce-with-mongodb.html)

1. > db.books.save(book1)
2. > db.books.save(book2)

The above two statements will create a collection called books under the database library. Following statement will list out the two books which we just saved.

[view plainprint?](http://blog.facilelogin.com/2012/02/mapreduce-with-mongodb.html)

1. > db.books.find();
3. { "\_id" : ObjectId("4f365b1ed6d9d6de7c7ae4b1"), "name" : "Understanding JAVA", "pages" : 100 }
4. { "\_id" : ObjectId("4f365b28d6d9d6de7c7ae4b2"), "name" : "Understanding JSON", "pages" : 200 }

Let's add few more records.

[view plainprint?](http://blog.facilelogin.com/2012/02/mapreduce-with-mongodb.html)

1. > book = {name : "Understanding XML", pages : 300}
2. > db.books.save(book)
3. > book = {name : "Understanding Web Services", pages : 400}
4. > db.books.save(book)
5. > book = {name : "Understanding Axis2", pages : 150}
6. > db.books.save(book)

6. Writing the Map function  
  
Let's process this library collection in a way that, we need to find the number of books having pages less 250 pages and greater than that.

[view plainprint?](http://blog.facilelogin.com/2012/02/mapreduce-with-mongodb.html)

1. > var map = function() {
2. var category;
3. if ( this.pages >= 250 )
4. category = 'Big Books';
5. else
6. category = "Small Books";
7. emit(category, {name: this.name});
8. };

Here, the collection produced by the Map function will have a collection of following members.

[view plainprint?](http://blog.facilelogin.com/2012/02/mapreduce-with-mongodb.html)

1. {"Big Books",[{name: "Understanding XML"}, {name : "Understanding Web Services"}]);
2. {"Small Books",[{name: "Understanding JAVA"}, {name : "Understanding JSON"},{name: "Understanding Axis2"}]);

7. Writing the Reduce function.

[view plainprint?](http://blog.facilelogin.com/2012/02/mapreduce-with-mongodb.html)

1. > var reduce = function(key, values) {
2. var sum = 0;
3. values.forEach(function(doc) {
4. sum += 1;
5. });
6. return {books: sum};
7. };

8. Running MapReduce against the books collection.

[view plainprint?](http://blog.facilelogin.com/2012/02/mapreduce-with-mongodb.html)

1. > var count  = db.books.mapReduce(map, reduce, {out: "book\_results"});
2. > db[count.result].find()
4. { "\_id" : "Big Books", "value" : { "books" : 2 } }
5. { "\_id" : "Small Books", "value" : { "books" : 3 } }

The above says, we have 2 Big Books and 3 Small Books.  
  
Everything done above using the MongoDB shell, can be done with Java t