# Достъп до MongoDB с Java – Map-Reduce

## I. Въведение

Map-reduce е парадигма за обработка на данни, която позволява да се обработват бързо големи масиви от данни и да се получават обобщени резултати. Map-reduce прилага фаза Map към всеки входен документ, по-точно към документите в колекцията, които отговарят на условието на заявката. Това условие се задава чрез фаза “query”. Функцията Map работи с двойки „ключ-стойност”. За ключовете, които имат няколко стойности, MongoDB прилага фазата за редуциране, която използва израз за обработка, който води до желаните обобщени данни, например получаване на сума или средна стойност. След това MongoDB съхранява резултатите в колекция. По желание изходът на функцията за редуциране може да премине през функция за финализиране, чрез която може да получим допълнително обобщаване на резултатите.

За извършване на операции Map-reduce MongoDB предоставя метод **mapReduce**. Синтаксисът на този метод е следния:

db.collection.mapReduce(map, reduce, options)

където:

- **map** е низ, който описва анонимна JavaScript функция от тялото на която се вика метод **emit** чрез който се задава двойката „ключ-стойност” с която ще се работи, например:

  "function() { emit(KEY, VALUE) }"

  След фаза Map, всички документи с еднаква стойност на ключа KEY се обединяват, а стойностите на поле “VALUE” от всички документи се записват в масив. Следователно тази фаза реализира операция групиране.

- **reduce** е низ, който описва анонимна JavaScript функция която като аргументи получава ключа и стойността. От тялото на функцията се реализира редуцирането на документите до стойност, например:

  "function(KEY, VALUE) { return Array.avg(VALUE); }"

  При конкретния пример функцията връща средната аритметичната стойност изчислена за всички полета с име “VALUE”. За целта е използвана статична функция avg.

- **options** е JSON обект чрез който можем да зададем първоначално филтриране на документите чрез поле “query”, както и името на новата колекция чрез поле “out”, например:

```
{
    query: {status: “ok”},
    out: “new_collection”
}
```

В този случай ще се обработват само документите, за които поле “status” има стойност “ok”. Изходната колекция се преименува до “new_collection”.

Нека за пример да вземем база данни “my-sensors”. В колекция “data” има 5 документа, които съдържат информация, получена от сензори за температура и влажност на въздуха в определен момент от време (поле “timestamp”). Типът на сензора се задава чрез поле “type”, а стойността на измерената величина чрез поле “value”. Трябва да намерим средната стойност за температурата и влажността на въздуха, получени от всички сензори преди дата 20.06.2021 г.. Необходимо е да групираме документите по техния тип. Следователно, за стойност на ключа задаваме KEY=”type”. Търсим средна стойност на показанията на сензорите. Следователно, за стойност на VALUE задаваме ”value”. Зададено е ограничение над входните за Map-Reduce операции документи. Това налага да използваме поле “query” чрез което ще зададем желания филтър. Филтрира се стойността на поле “timestamp” което е от тип ISODate. На Фиг. 1 е показан резултата, който се получава на всяка фаза от реализация на операции Map-Reduce. На фаза “query” се изключва документа с “timestamp” поле от дата 26.06.2021 г., тъй като не отговаря на зададения филтър. На фаза “map” остават само два документа. Единият описва показанията всички сензори за температура, а другият – сензорите за влажност на въздуха. Стойностите на сензорите са обединени в масив. При последната фаза “reduce” се реализира операция получаване на средна стойност (avg) за стойностите от масивите.

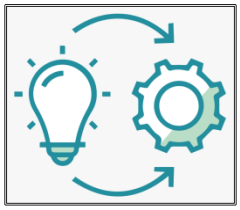

Фиг.1 Реализация на операция Map-Reduce за база данни my-sensors

Основното предимство на **mapReduce** е възможността да персонализираме както map функцията, така и reduce функцията. Това се реализира чрез вмъкване на необходимия JavaScript код в тялото на функциите. Можете да използвате пълните възможности на езика JavaScript и така да създавате сложни както map, така и reduce функции.

Основният недостатък на метод mapReduce е, че ако използвате map и reduce функциите в базовия им вариант няма да получите по-високо бързодействие при сравнение с използване на операторите за агрегиране на съдържание. Конвейерът за агрегиране на MongoDB осигурява по-добра производителност от Map-Reduce.

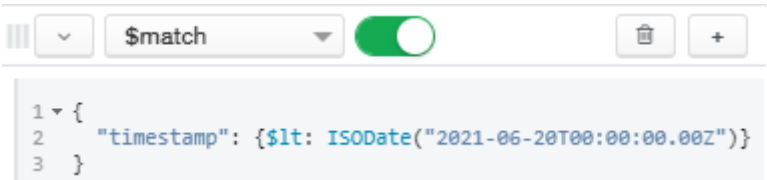
Всички операции Map-Reduce могат да бъдат пренаписани, като се използват операторите на конвейера за агрегиране, например $group и $merge. За операциите Map-Reduce, които изискват персонализирана функционалност, MongoDB от версия 4.4 предоставя операторите за агрегиране $accumulator и $function. Може да използвате тези оператори, за да дефинирате потребителски изрази за агрегиране на съдържание. Оператор **$function** дефинира потребителска функция за агрегиране или израз чрез JavaScript. Можете да използвате оператор $function, за да дефинирате потребителски функции за агрегиране, което не се поддържа от езика за заявки на MongoDB. Акумулаторите са оператори, които поддържат своето състояние (например общи суми, максимални стойности, минимални стойности и свързани с тях данни), докато документите преминават през конвейера. Използвайте оператора **$accumulator**, за да изпълнявате свои собствени функции на JavaScript, за да реализирате поведение, което не се поддържа от езика за заявки на MongoDB. Трябва да се има предвид, че изпълнението на JavaScript в израз за агрегиране може да намали производителността. Използвайте оператора $function само ако предоставените конвейерни оператори не могат да изпълнят нуждите на вашето приложение.

**Задачи за изпълнение**



**Задача 1**: *Използвайте приложението MongoDB Compass, за да получите чрез агрегиране средната стойност за показанията на всички сензори, групирани по типа им. Работете с база данни "my-sensors", колекция "data". Задайте филтриране по дата на получаване на показанията на сензорите: да се обработват данните получени преди дата 20.06.2021 г.*

Задачата можем да решим като използваме две фази на агрегиране. При първата фаза ще филтрираме документите по стойността на поле "timestamp". За целта ще използваме оператор $match:




След тази фаза ще отпадне един от документите. Следващата фаза трябва да групира документите по типа на сензорите. Ще използваме оператор $group, като за стойност за поле "_id" ще зададем "$value". Ще създадем и ново поле с име "avg". Стойността на това поле трябва да е средната стойност на показанията на сензорите. За целта ще използваме оператор $avg:

**Задача 2**: *Реализирайте условието на Задача 1 като Java конзолно приложение. Закръглете резултата (средна стойност на показанията на сензорите) до втория знак. Включете в резултата и мерните единици: ℃ за температурата и % за относителната влажност на въздуха.*

Използвайте възможността за генериране на Java код чрез приложението Compass. Създайте Java конзолно приложение с име на проекта MONGODB_EX_2. Декларирайте в тялото на метод main всички необходими константи:

```
final String DATABASE_NAME = "my-sensors";
final String COLLECTION_NAME = "data";
final String KEY = "type";
final String VALUE = "value";
```

Формирайте стойностите на аргументите на метод mapReduce – map функцията и reduce функцията:

```
String mapFunction =
        String.format("function() {emit(this.%s, this.%s)}",
                KEY, VALUE);
String reduceFunction =
        String.format("function(%s, %s) {return Array.avg(%s);}",
                KEY, VALUE, VALUE);
```

Създайте Bson обект, който описва правилото за филтриране:

```
String date = "2021-06-20T00:00:00.000Z";
Instant instant = Instant.parse(date);
Date timestamp = Date.from(instant);
Bson query = lt("timestamp", timestamp);
```

Следва свързване към базата данни и изпълнение на метод mapReduce. На Фиг. 2 е показан програмния код, който реализира комуникацията със сървъра и форматира резултата, съгласно заданието. След като получим обект за достъп до желаната колекция (редове 46-47) може да извикаме метод mapReduce (ред 50). Ще използваме варианта на метода., който изисква задаване на два аргумента: map функцията като низ и reduce функцията като низ. Ще използваме метод **filter** (ред 51) от интерфейс MapReduceIterable, за да зададем необходимото първоначално филтриране на документите по дата. Като аргумент на този метод подаваме обект query, който описва правилото за филтриране.

След изпълнение на метод mapReduce се получава обект “result”, който е от тип MapReduceIterable (ред 49). Създаваме обект-итератор, за да обработим последователно всички документи в обект “result” (ред 53). За целта използваме метод **hasNext**. За да разпечатаме коректно мерната единица проверяваме дали данните са от сензор за температура или сензор за относителна влажност (ред 59). За да визуализираме °C използваме уникод “\u00B0C”.

След стартиране на приложението трябва да получите следния резултат:

```
Average humidity: 59.67%
Average temperature: 26.00°C
```

```
40          try {
41              // ------------------------------------------------ Connect to server
42              MongoClient mongoClient = new MongoClient("localhost", 27017);
43              //------------------------------------------------- Create database
44              MongoDatabase db = mongoClient.getDatabase(DATABASE_NAME);
45              //------------------------------------------------- Create collection
46              MongoCollection<Document> collection
47                      = db.getCollection(COLLECTION_NAME);
48              // ------------------------------------------------------ Map-Reduce
49              MapReduceIterable<Document> result = collection
50                      .mapReduce(mapFunction, reduceFunction)
51                      .filter(query);
52              // ------------------------------------------------------ Show result
53              MongoCursor<Document> iterator = result.iterator();
54              while (iterator.hasNext()) {
55                  Document document = iterator.next();
56                  String json = document.toJson();
57                  JSONObject obj = new JSONObject(json);
58                  String sensorType = obj.getString("_id");
59                  String unit = sensorType.equals("temperature") ? "\u00B0C" : "%";
60                  double val = obj.getDouble("value");
61                  System.out.printf("Average %s: %.2f%s\n", sensorType, val, unit);
62              }
63              // ---------------------------------------------------------------
64
65          } catch (JSONException e) {
66              System.out.println(e);
67          }
```

Фиг. 2 Програмен код - Задача 2

**Задача 3:** *Напишете Java конзолно приложение, което чрез метод mapReduce намира средната стойност за показанията на сензорите за температура само ако температурата е по-висока от 20°C. Реализирайте задачата като дефинирате собствени map и reduce JavaScript функции. Работете с база данни "my-sensors", колекция "data".*

Задачата има за цел да демонстрира възможностите, които метод mapReduce предоставя при използване на собствени map и reduce функции. Ще реализираме необходимото филтриране в тялото на map функцията, а редуцирането (получаване на средна стойност) – в тялото на reduce функцията. На Фиг. 3 е показан програмния код, който дефинира необходимите функции. От тялото на map функцията (редове 22-26) емитираме типа на сензора като ключ и показанието на сензора като стойност, само ако типа на сензора е SENSOR_TYPE, а праговата стойност е по-голяма от VALUE_TH. Изчисляването на средна стойност се реализира чрез програмния код от тялото на функция reduce (редове 31-36). За целта използваме цикъл, който сумира стойностите на сензорите от всички входни документи. Крайният резултат получаваме като тази сума разделим на броя на показанията.

```
        String mapFunction =
                        "function(){ "
                            + "if (this.type === \""+ SENSOR_TYPE +"\") {"
                                + "if (this.value > "+ VALUE_TH +") {"
                                    + "emit(this.type, this.value)"
                                + "}"
                            + "}"
                        + "}";

        String reduceFunction =
                        "function(type, value){"
                            + "var n = value.length;"
                            + "var sum = 0;"
                            + "for (var i=0; i<n;i++) {"
                                + "sum += value[i];"
                            + "}"
                            + "return sum/n;"
                        + "}";
```

Фиг. 3 Дефиниране на функции map и reduce, Задача 5

Резултатът, който трябва да получите, е следния:

```
Average temperature: 26.00°C
----------------------------
```

**Задача 4:** *Напишете Java конзолно приложение, което чрез метод mapReduce намира средната стойност за показанията на сензорите за относителна влажност на въздуха за избран месец от избрана година. Работете с база данни "my-sensors", колекция "data".*

Ще използваме следните константи, свързани с базата данни:

```
final String DATABASE_NAME = "my-sensors";
final String COLLECTION_NAME = "data";
final String SENSOR_TYPE = "humidity";
final String KEY = "type";
final String VALUE = "value";
```

Трябва програмно да намерим броя на дните в зададените месец и година. За целта ще използваме клас **YearMonth** и метод **lengthOfMonth**:

```
final int YEAR = 2021;
final int MONTH = 10;

int daysInMonth;
try {
    YearMonth yearMonthObject = YearMonth.of(YEAR, MONTH);
    daysInMonth = yearMonthObject.lengthOfMonth();
}
catch(DateTimeException dte) {
    System.out.println("Задали сте невалидна стойност за месеца!");
    return;
}
```

Функции map и reduce приемат следния вид:

```
String mapFunction =
    "function(){ "
        + "if (this.type === \"" + SENSOR_TYPE + "\") {"
            + "emit(this.type, this.value)"
        + "}"
    + "}";
String reduceFunction
        = String.format("function(%s, %s) {return Array.avg(%s);}",
                KEY, VALUE, VALUE);
```

Следва формиране на обект query чрез който ще зададем филтриране на документите по дата:

```
String startDate =
        String.format("%04d-%02d-01T00:00:00.000Z", YEAR, MONTH);
String endDate =
        String.format("%04d-%02d-%02dT00:00:00.000Z",
                YEAR, MONTH, daysInMonth);
Instant instantStartDate = Instant.parse(startDate);
Instant instantEndDate = Instant.parse(endDate);
Date timestampStartDate = Date.from(instantStartDate);
Date timestampEndDate = Date.from(instantEndDate);
Bson query = and(
        gte("timestamp", timestampStartDate),
        lte("timestamp", timestampEndDate)
);
```

Изпълнение на функция mapReduce:

```
MapReduceIterable<Document> result = collection
        .mapReduce(mapFunction, reduceFunction)
        .filter(query);
```

Разпечатване на резултата:

```
MongoCursor<Document> iterator = result.iterator();
while (iterator.hasNext()) {
    Document document = iterator.next();
    String json = document.toJson();
    JSONObject obj = new JSONObject(json);
    String sensorType = obj.getString("_id");
    String unit = sensorType.equals("temperature") ? "\u00B0C" : "%";
    double val = obj.getDouble("value");
    System.out.printf("Average %s: %.2f%s\n", sensorType, val, unit);
}
```