(/)

A Guide to MongoDB with Java

Last modified: February 12, 2020

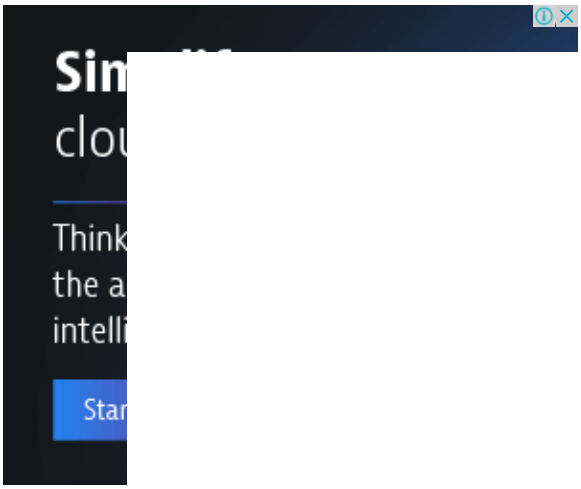
by Shubham Aggarwal (https://www.baeldung.com/author/shubham-aggarwal/)

Persistence (https://www.baeldung.com/category/persistence/)

MongoDB (https://www.baeldung.com/tag/mongodb/)

1. Overview

In this article, we'll have a look at integrating MongoDB (https://www.mongodb.com/), a very popular NoSQL open source database with a standalone Java client.



MongoDB is written in C++ and has quite a number of solid features such as map-reduce, auto-sharding, replication, high availability etc.

2. MongoDB

Let's start with a few key points about MongoDB itself:

- stores data in **JSON (https://www.w3schools.com/js/js_json_intro.asp)**-like documents that can have various structures
- uses dynamic schemas, which means that we can create records without predefining anything
- the structure of a record can be changed simply by adding new fields or deleting existing ones

The above-mentioned data model gives us the ability to represent hierarchical relationships, to store arrays and other more complex structures easily.

3. Terminologies

Understanding concepts in MongoDB becomes easier if we can compare them to relational database structures.

Let's see the analogies between Mongo and a traditional MySQL system:

- *Table* in MySQL becomes a *Collection* in Mongo
- *Row* becomes a *Document*
- *Column* becomes a *Field*
- *Joins* are defined as *linking* and *embedded* documents

This is a simplistic way to look at the MongoDB core concepts of course, but nevertheless useful.

Now, let's dive into implementation to understand this powerful database.

4. Maven Dependencies

We need to start by defining the dependency of a Java Driver for MongoDB:

```
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongo-java-driver</artifactId>
    <version>3.4.1</version>
</dependency>
```

To check if any new version of the library has been released – track the releases here (https://search.maven.org/classic/#search%7Cgav%7C1%7Cg%3A%22org.mongodb%22%20AND%20a%3A%22mongo-java-driver%22).

5. Using *MongoDB*

Now, let's start implementing Mongo queries with Java. We will follow with the basic CRUD operations as they are the best to start with.

5.1. Make a Connection With *MongoClient*

First, let's make a connection to a MongoDB server. With version >= 2.10.0, we'll use the *MongoClient*:

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

And for older versions use *Mongo* class:

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

5.2. Connecting to a Database

Now, let's connect to our database. It is interesting to note that we don't need to create one. When Mongo sees that database doesn't exist, it will create it for us:

```
DB database = mongoClient.getDB("myMongoDb");
```

Sometimes, by default, MongoDB runs in authenticated mode. In that case, we need to authenticate while connecting to a database.

We can do it as presented below:

```
MongoClient mongoClient = new MongoClient();
DB database = mongoClient.getDB("myMongoDb");
boolean auth = database.authenticate("username", "pwd".toCharArray());
```

5.3. Show Existing Databases

Let's display all existing databases. When we want to use the command line, the syntax to show databases is similar to MySQL:

```
show databases;
```

In Java, we display databases using snippet below:

```
mongoClient.getDatabaseNames().forEach(System.out::println);
```

The output will be:

```
local      0.000GB
myMongoDb  0.000GB
```

Above, *local* is the default Mongo database.

5.4. Create a *Collection*

Let's start by creating a *Collection* (table equivalent for MongoDB) for our database. Once we have connected to our database, we can make a *Collection* as:

```
database.createCollection("customers", null);
```

Now, let's display all existing collections for current database:

```
database.getCollectionNames().forEach(System.out::println);
```

The output will be:

```
customers
```

5.5. *Save – Insert*

The *save* operation has save-or-update semantics: if an *id* is present, it performs an *update*, if not – it does an *insert*.

When we *save* a new customer:

```
DBCollection collection = database.getCollection("customers");
BasicDBObject document = new BasicDBObject();
document.put("name", "Shubham");
document.put("company", "Baeldung");
collection.insert(document);
```

The entity will be inserted into a database:

```
{
    "_id" : ObjectId("33a52bb7830b8c9b233b4fe6"),
    "name" : "Shubham",
    "company" : "Baeldung"
}
```

Next, we'll look at the same operation – *save* – with *update* semantics.

5.6. *Save – Update*

Let's now look at *save* with *update* semantics, operating on an existing customer:

```
{
    "_id" : ObjectId("33a52bb7830b8c9b233b4fe6"),
    "name" : "Shubham",
    "company" : "Baeldung"
}
```

Now, when we *save* the existing customer – we will update it:

```
BasicDBObject query = new BasicDBObject();
query.put("name", "Shubham");

BasicDBObject newDocument = new BasicDBObject();
newDocument.put("name", "John");

BasicDBObject updateObject = new BasicDBObject();
updateObject.put("$set", newDocument);

collection.update(query, updateObject);
```

The database will look like this:

```
{
    "_id" : ObjectId("33a52bb7830b8c9b233b4fe6"),
    "name" : "John",
    "company" : "Baeldung"
}
```

As you can see, in this particular example, *save* uses the semantics of *update*, because we use object with given *_id*.

5.7. Read a *Document* From a *Collection*

Let's search for a *Document* in a *Collection* by making a query:

```
BasicDBObject searchQuery = new BasicDBObject();
searchQuery.put("name", "John");
DBCursor cursor = collection.find(searchQuery);

while (cursor.hasNext()) {
    System.out.println(cursor.next());
}
```

It will show the only *Document* we have by now in our *Collection*:

```
[
    {
      "_id" : ObjectId("33a52bb7830b8c9b233b4fe6"),
      "name" : "John",
      "company" : "Baeldung"
    }
]
```

5.8. *Delete* a *Document*

Let's move forward to our last CRUD operation, deletion:

```
BasicDBObject searchQuery = new BasicDBObject();
searchQuery.put("name", "John");

collection.remove(searchQuery);
```

With above command executed, our only *Document* will be removed from the *Collection*.

6. Conclusion

This article was a quick introduction to using MongoDB from Java.

The implementation of all these examples and code snippets can be found over on GitHub (https://github.com/eugenp/tutorials/tree/master/persistence-modules/java-mongodb) – this is a Maven based project, so it should be easy to import and run as it is.

Comments are closed on this article!

CATEGORIES

SPRING (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
REST (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SECURITY (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
HTTP CLIENT-SIDE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE COURSES (HTTPS://COURSES.BAELDUNG.COM)
JOBS (/TAG/ACTIVE-JOB/)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORS (/EDITORS)
OUR PARTNERS (/PARTNERS)
ADVERTISE ON BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)
PRIVACY POLICY (/PRIVACY-POLICY)
COMPANY INFO (/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)