Learn about Java Driver in MongoDB

In the following article, we will introduce you some basic features of MongoDV Java Driver as well as how to deploy and apply in practice.

TipsMake.com - In the following article, we will introduce you some basic features of MongoDV Java Driver as well as how to deploy and apply in practice . To learn more about Java API functions, please visit here.

In fact, using Java drivers is quite simple. First, you need to make sure that the **mongo.jar** jar driver is included in the main **classpath** . The following code is taken from the example **/ QuickTour.java** example in the driver.

Create connection:

To create a connection to **MongoDB** , the minimum condition to meet is the name of the database. Technically, this database does not need to exist, if not, **MongoDB** will create one copy for the user. Besides, users can clearly specify the address and port of the server when connecting. The following example shows three basic ways to connect to a database named mydb on the local computer:

```
import com.mongodb.Mongo;  import com.mongodb.DB;  import com.mongodb.DBCollec
```

```
Mongo m = new Mongo ();
// or
Mongo m = new Mongo ("localhost");
// or
Mongo m = new Mongo ("localhost", 27017);

DB db = m.getDB ("mydb");
```

At this point, the db object becomes the primary connection to the MongoDB server for a fixed database. And with this component we will be able to perform many more tasks. However, you should note that the Mongo object will become a pool of the database connection process, and we only need one object of the Mongo class with many different threads (refer here).). In essence, Mongo class is designed specifically to be a secure thread and easily shared among many different threads. Usually, you only need to create an example for cluster DB and use it throughout the application. If you encounter a special case where we are forced to create many variables, notice:

1. Limitations on resource use (maximum number of connections .) are applied on a variable.
2. To close a variable, use the **mongo.close ()** function to wipe the data about the resource used.

Confirmation process (optional):

MongoDB can operate in Secure mode, where all access to the database is controlled, monitored by name and password. And when doing so, any client application must provide the name and password before performing the next steps. In Java Driver, we only need to use the following command syntax when connecting to a mongo object:

```
boolean auth = db.authenticate (myUserName, myPassword);
```

If the name and password are correct with credentials in the **database** , **auth** will become **true** , otherwise it will become **false** (check MongoDB log file). In fact, most people use MongoDB without having to confirm it in a safe and secure environment.

Get Collection list:

In essence, each database has 0 or more Collection, and we can get a list of those components from db:

```
Set colls = db.getCollectionNames ();
```

for (String s: colls) {
System.out.println (s);
}

and assume that there are two collections, names and passwords inside the database. Results returned from the system will include:

```
name address
```

Get Collection:

If you want to retrieve any one collection to use, you only need to explicitly specify that collection's name to getCollection (String collectionName):

```
DBCollection coll = db.getCollection ("testCollection")
```

Once we have this collection object, we can easily perform other operations, such as insert, data query .

Insert Document:

After obtaining the collection object, you can insert the document - Document into that collection. For example, try to create a small document in JSON as follows:

```
{"name": "MongoDB", "type": "database", "count": 1, "info": {x: 203, y: 102}}
```

When doing so, it means that the system has an embedded inner document inside. To do so, use the BasicDBObject class to create the text (including the inner document), and then insert it into the Collection with the insert syntax ()

```
BasicDBObject doc = new BasicDBObject ();
```

doc.put ("name", "MongoDB");
doc.put ("type", "database");
doc.put ("count", 1);

BasicDBObject info = new BasicDBObject ();

info.put ("x", 203);
info.put ("y", 102);

doc.put ("info", info);

coll.insert (doc);

Find the first Document in the Collection using findOne ()

To list and display the text inserted in the previous step, we just use the *findOne* operator to get the first document in the collection list. And this method will return a single text (rather than the *DBCurso* r form found and returned by the *find ()* operator, so it is suitable for cases where there is only 1 text, and they are We don't have to 'face' cursor:

```
DBObject myDoc = coll.findOne ();  System.out.println (myDoc);
```

The returned result will look like this:

```
{"_id": "49902cde5162504500b45c2c", "name": "MongoDB", "type": "database", "cou
```

Note that the _id component will be automatically attached to the document by MongoDB. Remember that MongoDB will also automatically store components with names starting with "**_**" / "**$**" for use in internal systems later.

Assign more documents:

The document section assigned to the collection will look like this:

```
{"i": value}
```

And to do this, we only need to apply the effective loop:

```
for (int i = 0; i 100; i ++) {coll.insert (new BasicDBObject (). append ("i", i
```

Note that you can insert different types of documents into the same collection, which also means that MongoDB is a schema-free form.

Count Document in Collection:

Now, when we have inserted 101 documents (including 100 in the loop and 1 element from the beginning), it is possible to check whether those components use the getCount () function:

```
System.out.println (coll.getCount ());
```

The system will return the result 101 as desired.

Use Cursor to retrieve all Documents:

To do this, we can use the find () syntax to return the DBCursor object to allow the user to repeat the text that matches the query search query. Therefore, use the following command to query all Document sections and list them:

```
DBCursor cur = coll.find ();
```

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

The system will display the full 101 documents in the collection.

Get a single document with the query statement:

Technically, users can create a query statement to pass the find () syntax to the collection of text collections in the collection. For example, if you want to find any text that has the value of field i is 71, follow this syntax:

```
BasicDBObject query = new BasicDBObject ();
```

```
query.put ("i", 71);

cur = coll.find (query);

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

and the system will return exactly 1 single document:

```
{"_id": "49903677516250c1008d624e", "i": 71}
```

Besides, you often see many examples and documentation in MongoDB or use the $ operator, such as:

```
db.things.find ({j: {$ ne: 3}, k: {$ gt: 10}});
```

They represent **String** keys in **Java drivers** , using embedded **DBObjects** objects:

```
BasicDBObject query = new BasicDBObject ();
```

query.put ("j", new BasicDBObject ("$ ne", 3));
query.put ("k", new BasicDBObject ("$ gt", 10));

cur = coll.find (query);

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

Get lots of text with a query statement:

In fact, we can retrieve multiple documents from the collection with only one query statement. For example, if you want to list the entire text with the condition "i"> 50, then apply the following command:

```
query = new BasicDBObject ();
```

query.put ("i", new BasicDBObject ("$ gt", 50)); // eg find all where i> 50

cur = coll.find (query);

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

or with condition 20

```
query = new BasicDBObject ();
```

query.put ("i", new BasicDBObject ("$ gt", 20) .append ("$ lte", 30)); // ie 20

cur = coll.find (query);

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

Create Index:

MongoDB supports index, and they are easy to assign to the collection. To create an index, we just need to specify the data field to be indexed, and the sort order is **ascending - ascending (1)** or **descending - descending (-1).** The following statement will generate index entries in the ascending order of the field i:

```
coll.createIndex (new BasicDBObject ("i", 1));  // create index on "i", ascend
```

Get the list of index items in the collection:

The general command syntax is as follows:

```
List list = coll.getIndexInfo ();
```

for (DBObject o: list) {
System.out.println (o);
}

And the system will display the return list as follows:

```
{"name": "i_1", "ns": "mydb.testCollection", "key": {"i": 1}}
```

Refer to administrative features:

First, get the list of the database with the command:

```
Mongo m = new Mongo ();
```

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

If you want to delete the database by name, use the variable according to Mongo's corresponding object type:

```
m.dropDatabase ("my_new_db");
```

Hopefully, the above information can help you understand some of the basic features of MongoDV Java Driver as well as how to deploy and apply in practice. Good luck!