
mongojs

A node.js module for mongodb, that emulates the official mongodb API as much as possible. It wraps mongodb-native and is available through npm

```
1 npm install mongojs
```

build unknown code style standard

Usage

mongojs is easy to use:

```
1 const mongojs = require('mongojs')
2 const db = mongojs(connectionString, [collections])
```

The connection string should follow the format described in the mongo connection string docs. Some examples of this could be:

```
1 // simple usage for a local db
2 const db = mongojs('mydb', ['mycollection'])
3
4 // the db is on a remote server (the port default to mongo)
5 const db = mongojs('example.com/mydb', ['mycollection'])
6
7 // we can also provide some credentials
8 const db = mongojs('username:password@example.com/mydb', ['mycollection'])
9
10 // connect using SCRAM-SHA-1 mechanism
11 const db = mongojs('username:password@example.com/mydb?authMechanism=SCRAM-SHA-1', ['mycollection'])
12
13 // connect using a different auth source
14 const db = mongojs('username:password@example.com/mydb?authSource=authdb', ['mycollection'])
15
16 // connect with options
17 const db = mongojs('username:password@example.com/mydb', ['mycollection'], { ssl: true })
18
19 // connect now, and worry about collections later
20 const db = mongojs('mydb')
21 const mycollection = db.collection('mycollection')
```

More connection string examples

After we connected we can query or update the database just how we would using the mongo API with the exception that we use a callback. The format for callbacks is always `callback(error, value)` where error is null if no exception has occurred. The update methods `save`, `remove`, `update` and `findAndModify` also pass the `lastErrorObject` as the last argument to the callback function.

```
1 // find everything
2 db.mycollection.find(function (err, docs) {
3   // docs is an array of all the documents in mycollection
4 })
5
6 // find everything, but sort by name
7 db.mycollection.find().sort({name: 1}, function (err, docs) {
8   // docs is now a sorted array
9 })
10
11 // iterate over all whose level is greater than 90.
12 db.mycollection.find({level: {$gt: 90}}).forEach(function (err, doc) {
13   if (!doc) {
14     // we visited all docs in the collection
15     return
16   }
17   // doc is a document in the collection
18 })
19
20 // find a document using a native ObjectId
21 db.mycollection.findOne({
22   _id: mongoose.ObjectId('523209c4561c640000000001')
23 }, function(err, doc) {
24   // doc._id.toString() === '523209c4561c640000000001'
25 })
26
27 // find all named 'mathias' and increment their level
28 db.mycollection.update({name: 'mathias'}, {$inc: {level: 1}}, {multi:
29   true}, function () {
30   // the update is complete
31 })
32
33 // find one named 'mathias', tag him as a contributor and return the
34 // modified doc
35 db.mycollection.findAndModify({
36   query: { name: 'mathias' },
37   update: { $set: { tag: 'maintainer' } },
38   new: true
39 }, function (err, doc, lastErrorObject) {
40   // doc.tag === 'maintainer'
41 })
42
43 // use the save function to just save a document (callback is optional
```

```
    for all writes)
43 db.mycollection.save({created: 'just now'})
```

If you provide a callback to `find` or any cursor config operation mongojs will call `toArray` for you

```
1 db.mycollection.find({}, function (err, docs) { ... })
2
3 db.mycollection.find({}).limit(2).skip(1, function (err, docs) { ... })
```

is the same as

```
1 db.mycollection.find({}).toArray(function (err, docs) { ... })
2
3 db.mycollection.find({}).limit(2).skip(1).toArray(function (err, docs)
  { ... })
```

For more detailed information about the different usages of update and querying see the mongo docs

Events

```
1 const db = mongojs('mydb', ['mycollection'])
2
3 db.on('error', function (err) {
4   console.log('database error', err)
5 })
6
7 db.on('connect', function () {
8   console.log('database connected')
9 })
```

Streaming cursors

As of 0.7.0 all cursors are a readable stream of objects.

```
1 const JSONStream = require('JSONStream')
2
3 // pipe all documents in mycollection to stdout
4 db.mycollection.find({}).pipe(JSONStream.stringify()).pipe(process.
  stdout)
```

Notice that you should pipe the cursor through a stringifier (like `JSONStream`) if you want to pipe it to a serial stream like a http response.

Tailable cursors

If you are using a capped collection you can create a tailable cursor to that collection by adding `tailable:true` to the find options

```
1 const cursor = db.mycollection.find({}, {}, {tailable: true, timeout: false})
2
3 // since all cursors are streams we can just listen for data
4 cursor.on('data', function (doc) {
5     console.log('new document', doc)
6 })
```

Note that you need to explicitly set the selection parameter in the `find` call.

Database commands

With mongojs you can run database commands just like with the mongo shell using `db.runCommand()`

```
1 db.runCommand({ping: 1}, function (err, res) {
2     if(!err && res.ok) console.log('we\'re up')
3 })
```

or `db.collection.runCommand()`

```
1 db.things.runCommand('count', function (err, res) {
2     console.log(res)
3 })
```

Bulk updates

As of 0.15 mongojs supports the Bulk updates introduced in mongodb 2.6. Here's an example of the usage

```
1 const bulk = db.a.initializeOrderedBulkOp()
2 bulk.find({type: 'water'}).update({$set: {level: 1}})
3 bulk.find({type: 'water'}).update({$inc: {level: 2}})
4 bulk.insert({name: 'Spearow', type: 'flying'})
5 bulk.insert({name: 'Pidgeotto', type: 'flying'})
6 bulk.insert({name: 'Charmeleon', type: 'fire'})
7 bulk.find({type: 'flying'}).removeOne()
8 bulk.find({type: 'fire'}).remove()
9 bulk.find({type: 'water'}).updateOne({$set: {hp: 100}})
10
11 bulk.execute(function (err, res) {
```

```
12 console.log('Done!')
13 }
```

Replication Sets

Mongojs can also connect to a mongo replication set by providing a connection string with multiple hosts

```
1 const db = mongojs('rs-1.com,rs-2.com,rs-3.com/mydb?slaveOk=true', ['mycollection'])
```

For more detailed information about replica sets see the [mongo replication docs](#)

Using harmony features

If you run node.js with the `--harmony` option, then you can omit the collection names array, and you can do stuff like.

```
1 const mongojs = require('mongojs')
2 const db = require('mydb')
3
4 db.hackers.insert({name: 'Ed'})
```

In the above example the `hackers` collection is enabled automatically (similar to the shell) using the `Proxy` feature in harmony

Passing a DB to the constructor

If you have an instance of mongojs, you can pass this to the constructor and mongojs will use the existing connection of that instance instead of creating a new one.

```
1 const mongodb = require('mongodb')
2 const mongojs = require('mongojs')
3
4 mongodb.Db.connect('mongodb://localhost/test', function (err, theDb) {
5   const db = mongojs(theDb, ['myCollection'])
6 })
```

Features not supported for MongoDB 2.4 or older (on mongojs version 1.0+).

- Index creation and deletion

-
- Aggregation cursors.

These features are relatively easy to add, but would make the code unnecessarily more complex. If you are using mongodb 2.4 or older and would like to use mongojs 1.0 with the above mentioned features, feel free to make a pull request or open an issue..

Upgrading from 0.x.x to 1.2.x

Version > 1.0.x is a major rewrite of mongojs. So expect some things not to work the same as in mongojs 0.x.x versions. Breaking changes include:

- **Removed** `mongojs.connect` use `mongojs()` directly instead

API

This API documentation is a work in progress.

Collection

`db.collection.aggregate([pipeline], [options], [callback])` <https://docs.mongodb.org/manual/reference/method/db.collection.aggregate/>

`db.collection.aggregate([pipelineStep], [pipelineStep], [pipelineStep], ..., [callback])`

`db.collection.count([query], callback)`

`db.collection.createIndex(keys, options, [callback])`

`db.collection.distinct(field, query, callback)`

`db.collection.drop([callback])`

`db.collection.dropIndex(index, [callback])`

db.collection.dropIndexes([callback])

db.collection.ensureIndex(keys, options, [callback])

db.collection.find([criteria], [projection], [callback]) This function applies a query to a collection. You can get the return value, which is a cursor, or pass a callback as the last parameter. Said callback receives ([err](#), [documents](#))

db.collection.findOne([criteria], [projection], callback) Apply a query and get one single document passed as a callback. The callback receives ([err](#), [document](#))

db.collection.findAndModify(document, callback)

db.collection.getIndexes(callback)

db.collection.group(document, callback)

db.collection.insert(docOrDocs, [callback])

db.collection.isCapped(callback)

db.collection.mapReduce(map, reduce, options, [callback])

db.collection.reIndex([callback])

db.collection.remove(query, [justOne], [callback])

db.collection.remove(query, [options], [callback])

db.collection.runCommand(command, [callback])

db.collection.save(doc, [options], [callback])

db.collection.stats(callback)

db.collection.update(query, update, [options], [callback])

db.collection.toString() Get the name of the collection.

Cursor

cursor.batchSize(size, [callback])

cursor.count(callback)

cursor.explain(callback)

cursor.forEach(function)

cursor.limit(n, [callback])

cursor.map(function, [callback])

cursor.next(callback)

cursor.skip(n, [callback])

cursor.sort(sortOptions, [callback])

cursor.toArray(callback)

cursor.rewind()

cursor.destroy()

Database

db.addUser(document)

db.createCollection(name, options, [callback])

db.dropDatabase([callback])

db.eval(code, [params], [options], [callback])

db.getCollectionNames(callback)

db.getLastError(callback)

db.getLastErrorObj(callback)

db.removeUser(username, [callback])

db.runCommand(command, [callback])

db.stats([callback])

db.close()

Bulk

bulk.execute()

bulk.find(query)

bulk.find.remove()

bulk.find.removeOne()

bulk.find.replaceOne(document)

bulk.find.update(updaterParam)

bulk.find.updateOne(updaterParam)

bulk.find.upsert(upsertParam)

bulk.insert(document)

bulk.toString()

bulk.tojson()