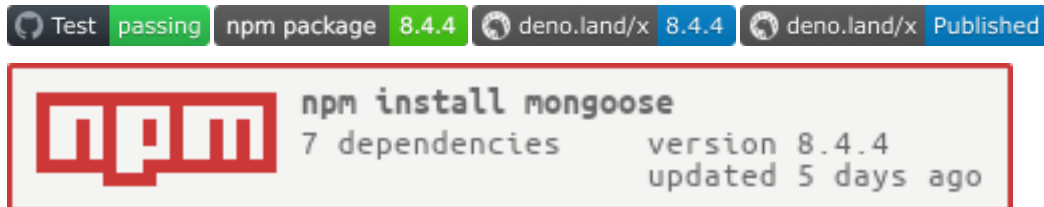


---

## Mongoose

Mongoose is a MongoDB object modeling tool designed to work in an asynchronous environment. Mongoose supports Node.js and Deno (alpha).



## Documentation

The official documentation website is [mongoosejs.com](https://mongoosejs.com).

Mongoose 8.0.0 was released on October 31, 2023. You can find more details on backwards breaking changes in 8.0.0 on our docs site.

## Support

- [Stack Overflow](#)
- [Bug Reports](#)
- [Mongoose Slack Channel](#)
- [Help Forum](#)
- [MongoDB Support](#)

## Plugins

Check out the [plugins search site](#) to see hundreds of related modules from the community. Next, learn how to write your own plugin from the docs or this [blog post](#).

## Contributors

Pull requests are always welcome! Please base pull requests against the [master](#) branch and follow the contributing guide.

If your pull requests makes documentation changes, please do **not** modify any `.html` files. The `.html` files are compiled code, so please make your changes in `docs/*.pug`, `lib/*.js`, or `test/docs/*.js`.

---

View all 400+ contributors.

## Installation

First install Node.js and MongoDB. Then:

```
1 npm install mongoose
```

Mongoose 6.8.0 also includes alpha support for Deno.

## Importing

```
1 // Using Node.js `require()`  
2 const mongoose = require('mongoose');  
3  
4 // Using ES6 imports  
5 import mongoose from 'mongoose';
```

Or, using Deno's `createRequire()` for CommonJS support as follows.

```
1 import { createRequire } from 'https://deno.land/std@0.177.0/node/  
  module.ts';  
2 const require = createRequire(import.meta.url);  
3  
4 const mongoose = require('mongoose');  
5  
6 mongoose.connect('mongodb://127.0.0.1:27017/test')  
7   .then(() => console.log('Connected!'));
```

You can then run the above script using the following.

```
1 deno run --allow-net --allow-read --allow-sys --allow-env mongoose-test  
  .js
```

## Mongoose for Enterprise

Available as part of the Tidelift Subscription

The maintainers of mongoose and thousands of other packages are working with Tidelift to deliver commercial support and maintenance for the open source dependencies you use to build your applications. Save time, reduce risk, and improve code health, while paying the maintainers of the exact dependencies you use. Learn more.

---

## Overview

### Connecting to MongoDB

First, we need to define a connection. If your app uses only one database, you should use `mongoose.connect`. If you need to create additional connections, use `mongoose.createConnection`.

Both `connect` and `createConnection` take a `mongodb://` URI, or the parameters `host`, `database`, `port`, `options`.

```
1 await mongoose.connect('mongodb://127.0.0.1/my_database');
```

Once connected, the `open` event is fired on the `Connection` instance. If you're using `mongoose.connect`, the `Connection` is `mongoose.connection`. Otherwise, `mongoose.createConnection` return value is a `Connection`.

**Note:** If the local connection fails then try using `127.0.0.1` instead of `localhost`. Sometimes issues may arise when the local hostname has been changed.

**Important!** Mongoose buffers all the commands until it's connected to the database. This means that you don't have to wait until it connects to MongoDB in order to define models, run queries, etc.

### Defining a Model

Models are defined through the `Schema` interface.

```
1 const Schema = mongoose.Schema;
2 const ObjectId = Schema.ObjectId;
3
4 const BlogPost = new Schema({
5   author: ObjectId,
6   title: String,
7   body: String,
8   date: Date
9 });
```

Aside from defining the structure of your documents and the types of data you're storing, a `Schema` handles the definition of:

- Validators (async and sync)
- Defaults
- Getters
- Setters
- Indexes

- 
- Middleware
  - Methods definition
  - Statics definition
  - Plugins
  - pseudo-JOINs

The following example shows some of these features:

```
1 const Comment = new Schema({
2   name: { type: String, default: 'hahaha' },
3   age: { type: Number, min: 18, index: true },
4   bio: { type: String, match: /[a-z]/ },
5   date: { type: Date, default: Date.now },
6   buff: Buffer
7 });
8
9 // a setter
10 Comment.path('name').set(function(v) {
11   return capitalize(v);
12 });
13
14 // middleware
15 Comment.pre('save', function(next) {
16   notify(this.get('email'));
17   next();
18 });
```

Take a look at the example in `examples/schema/schema.js` for an end-to-end example of a typical setup.

## Accessing a Model

Once we define a model through `mongoose.model('ModelName', mySchema)`, we can access it through the same function

```
1 const MyModel = mongoose.model('ModelName');
```

Or just do it all at once

```
1 const MyModel = mongoose.model('ModelName', mySchema);
```

The first argument is the *singular* name of the collection your model is for. **Mongoose automatically looks for the plural version of your model name.** For example, if you use

```
1 const MyModel = mongoose.model('Ticket', mySchema);
```

---

Then `MyModel` will use the **tickets** collection, not the **ticket** collection. For more details read the model docs.

Once we have our model, we can then instantiate it, and save it:

```
1 const instance = new MyModel();
2 instance.my.key = 'hello';
3 await instance.save();
```

Or we can find documents from the same collection

```
1 await MyModel.find({});
```

You can also `findOne`, `findById`, `update`, etc.

```
1 const instance = await MyModel.findOne({ /* ... */ });
2 console.log(instance.my.key); // 'hello'
```

For more details check out the docs.

**Important!** If you opened a separate connection using `mongoose.createConnection()` but attempt to access the model through `mongoose.model('modelName')` it will not work as expected since it is not hooked up to an active db connection. In this case access your model through the connection you created:

```
1 const conn = mongoose.createConnection('your connection string');
2 const MyModel = conn.model('modelName', schema);
3 const m = new MyModel();
4 await m.save(); // works
```

vs

```
1 const conn = mongoose.createConnection('your connection string');
2 const MyModel = mongoose.model('modelName', schema);
3 const m = new MyModel();
4 await m.save(); // does not work b/c the default connection object was
  never connected
```

## Embedded Documents

In the first example snippet, we defined a key in the Schema that looks like:

```
1 comments: [Comment]
```

Where `Comment` is a `Schema` we created. This means that creating embedded documents is as simple as:

---

```
1 // retrieve my model
2 const BlogPost = mongoose.model('BlogPost');
3
4 // create a blog post
5 const post = new BlogPost();
6
7 // create a comment
8 post.comments.push({ title: 'My comment' });
9
10 await post.save();
```

The same goes for removing them:

```
1 const post = await BlogPost.findById(myId);
2 post.comments[0].deleteOne();
3 await post.save();
```

Embedded documents enjoy all the same features as your models. Defaults, validators, middleware.

## Middleware

See the docs page.

**Intercepting and mutating method arguments** You can intercept method arguments via middleware.

For example, this would allow you to broadcast changes about your Documents every time someone [sets](#) a path in your Document to a new value:

```
1 schema.pre('set', function(next, path, val, type) {
2   // `this` is the current Document
3   this.emit('set', path, val);
4
5   // Pass control to the next pre
6   next();
7 });
```

Moreover, you can mutate the incoming [method](#) arguments so that subsequent middleware see different values for those arguments. To do so, just pass the new values to [next](#):

```
1 schema.pre(method, function firstPre(next, methodArg1, methodArg2) {
2   // Mutate methodArg1
3   next('altered-' + methodArg1.toString(), methodArg2);
4 });
5
```

---

```

6 // pre declaration is chainable
7 schema.pre(method, function secondPre(next, methodArg1, methodArg2) {
8   console.log(methodArg1);
9   // => 'altered-originalValOfMethodArg1'
10
11   console.log(methodArg2);
12   // => 'originalValOfMethodArg2'
13
14   // Passing no arguments to `next` automatically passes along the
15   // current argument values
16   // i.e., the following `next()` is equivalent to `next(methodArg1,
17   // methodArg2)`
18   // and also equivalent to, with the example method arg
19   // values, `next('altered-originalValOfMethodArg1', '
20   // originalValOfMethodArg2')`
21   next();
22 });

```

**Schema gotcha** `type`, when used in a schema has special meaning within Mongoose. If your schema requires using `type` as a nested property you must use object notation:

```

1 new Schema({
2   broken: { type: Boolean },
3   asset: {
4     name: String,
5     type: String // uh oh, it broke. asset will be interpreted as
6                   String
7   }
8 });
9
10 new Schema({
11   works: { type: Boolean },
12   asset: {
13     name: String,
14     type: { type: String } // works. asset is an object with a type
15                       property
16   }
17 });

```

## Driver Access

Mongoose is built on top of the official MongoDB Node.js driver. Each mongoose model keeps a reference to a native MongoDB driver collection. The collection object can be accessed using `YourModel.collection`. However, using the collection object directly bypasses all mongoose features, including hooks, validation, etc. The one notable exception that `YourModel.collection` still buffers commands. As such, `YourModel.collection.find()` will **not** return a cursor.

---

## API Docs

Find the API docs here, generated using dox and acquit.

## Related Projects

### MongoDB Runners

- run-rs
- mongodb-memory-server
- mongodb-topology-manager

### Unofficial CLIs

- mongoosejs-cli

### Data Seeding

- dookie
- seedgoose
- mongoose-data-seed

### Express Session Stores

- connect-mongodb-session
- connect-mongo

## License

Copyright (c) 2010 LearnBoost <dev@learnboost.com>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the 'Software'), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:



---

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED 'AS IS', WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.