npm is now a part of GitHub

Products Pricing Documentation Community

npm Sign Up Sign In

Search packages Search

Miss any of our Open RFC calls? Watch the recordings here! »

mongoose

5.9.28 • Public • Published 5 days ago

Readme

Explore BETA

11 Dependencies

9,032 Dependents

602 Versions

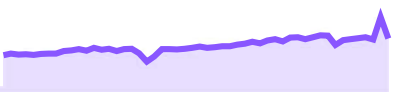
Install

```
> npm i mongoose
```

Fund this package

Weekly Downloads

955,680



Version
5.9.28

License
MIT

Unpacked Size
2.01 MB

Total Files
209

Issues
326

Pull Requests
5

Homepage
mongoosejs.com

Repository
github.com/Automattic/mongoose

Last publish
5 days ago

Collaborators




Try on RunKit

Report a vulnerability

Mongoose

Mongoose is a MongoDB object modeling tool designed to work in an asynchronous environment. Mongoose supports both promises and callbacks.

Documentation

The official documentation website is **mongoosejs.com**.

Mongoose 5.0.0 was released on January 17, 2018. You can find more details on **backwards breaking changes in 5.0.0 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:

```
$ npm install mongoose
```

Importing

```
// Using Node.js `require()`
const mongoose = require('mongoose');

// Using ES6 imports
import mongoose from 'mongoose';
```

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`.

```
await mongoose.connect('mongodb://localhost/my_database', {
  useNewUrlParser: true,
  useUnifiedTopology: true
});
```

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.

```
const Schema = mongoose.Schema;
const ObjectId = Schema.ObjectId;

const BlogPost = new Schema({
  author: ObjectId,
  title: String,
  body: String,
  date: Date
});
```

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:

```
const Comment = new Schema({
```

```
  name: { type: String, default: 'hahaha' },
  age: { type: Number, min: 18, index: true },
  bio: { type: String, match: /[a-z]/ },
  date: { type: Date, default: Date.now },
  buff: Buffer
});

// a setter
Comment.path('name').set(function (v) {
  return capitalize(v);
});

// middleware
Comment.pre('save', function (next) {
  notify(this.get('email'));
  next();
});
```

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

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

Or just do it all at once

```
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

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

Then Mongoose will create the model for your **tickets** collection, not your **ticket** collection.

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

```
const instance = new MyModel();
instance.my.key = 'hello';
instance.save(function (err) {
  //
});
```

Or we can find documents from the same collection

```
MyModel.find({}, function (err, docs) {
  // docs.forEach
});
```

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

```
const instance = await MyModel.findOne({ ... });
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:

```
const conn = mongoose.createConnection('your connection string');
const MyModel = conn.model('ModelName', schema);
const m = new MyModel;
m.save(); // works
```

vs

```
const conn = mongoose.createConnection('your connection string');
const MyModel = mongoose.model('ModelName', schema);
const m = new MyModel;
m.save(); // does not work b/c the default connection object was ne
```

Embedded Documents

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

```
comments: [Comment]
```

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

```
// retrieve my model
const BlogPost = mongoose.model('BlogPost');

// create a blog post
const post = new BlogPost();

// create a comment
post.comments.push({ title: 'My comment' });

post.save(function (err) {
  if (!err) console.log('Success!');
});
```

The same goes for removing them:

```
BlogPost.findById(myId, function (err, post) {
  if (!err) {
    post.comments[0].remove();
    post.save(function (err) {
      // do something
    });
  }
```

```
});
```

Embedded documents enjoy all the same features as your models. Defaults, validators, middleware. Whenever an error occurs, it's bubbled to the `save()` error callback, so error handling is a snap!

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 `set`s a path in your Document to a new value:

```
schema.pre('set', function (next, path, val, typel) {
  // `this` is the current Document
  this.emit('set', path, val);

  // Pass control to the next pre
  next();
});
```

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`:

```
.pre(method, function firstPre (next, methodArg1, methodArg2) {
  // Mutate methodArg1
  next("altered-" + methodArg1.toString(), methodArg2);
});

// pre declaration is chainable
.pre(method, function secondPre (next, methodArg1, methodArg2) {
  console.log(methodArg1);
  // => 'altered-originalValOfMethodArg1'
```

```
    console.log(methodArg2);
    // => 'originalValOfMethodArg2'

    // Passing no arguments to `next` automatically passes along the
    // i.e., the following `next()` is equivalent to `next(methodArg1
    // and also equivalent to, with the example method arg
    // values, `next('altered-originalValOfMethodArg1', 'originalValO
    next();
  });
```

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:

```
  new Schema({
    broken: { type: Boolean },
    asset: {
      name: String,
      type: String // uh oh, it broke. asset will be interpreted as S
    }
  });

  new Schema({
    works: { type: Boolean },
    asset: {
      name: String,
      type: { type: String } // works. asset is an object with a type
    }
  });
```

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.

Keywords

mongodb document model schema database odm data datastore query nosql orm db

Support

Help

Community

Advisories

Status

Contact npm

Company

About

Blog

Press

Terms & Policies

Policies

Terms of Use

Code of Conduct

Privacy