mongoose

Fork me on GitHub

index.js

Mongoose#Aggregate()

The Mongoose Aggregate constructor

Mongoose#CastError(`type`, `value`, `path`, `[reason]`)

The Mongoose CastError constructor

Parameters:

- `type` <String> The name of the type
- `value` <Any> The value that failed to cast
- `path` <String> The path `a.b.c` in the doc where this cast error occurred
- `[reason]` <Error> The original error that was thrown

Mongoose#Collection()

The Mongoose Collection constructor

Mongoose#connect(`uri(s)`, `[options]`, `[callback]`)

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

Opens the default mongoose connection.

Parameters:

- `uri(s)` <String>
- `[options]` <Object>
- `[callback]` <Function>

Returns:

- <MongooseThenable> pseudo-promise wrapper around this

See:

- Mongoose#createConnection

If arguments are passed, they are proxied to either Connection#open or Connection#openSet appropriately.

Options passed take precedence over options included in connection strings.

Example:

```
mongoose.connect('mongodb://user:pass@localhost:port/database

// replica sets
var uri = 'mongodb://user:pass@localhost:port,anotherhost:po
mongoose.connect(uri);

// with options
mongoose.connect(uri, options);

// connecting to multiple mongos
var uri = 'mongodb://hostA:27501,hostB:27501';
var opts = { mongos: true };
mongoose.connect(uri, opts);

// optional callback that gets fired when initial connection
var uri = 'mongodb://nonexistent.domain:27000';
mongoose.connect(uri, function(error) {
```

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

```
  // if error is truthy, the initial connection failed.
})
```

show code

Mongoose#Connection()

The Mongoose Connection constructor

Mongoose#createConnection(`[uri]`, `[options]`, `[options.config]`, `[options.config.autoIndex]`)

Creates a Connection instance.

Parameters:

- `[uri]` <String> a mongodb:// URI
- `[options]` <Object> options to pass to the driver
- `[options.config]` <Object> mongoose-specific options
- `[options.config.autoIndex]` <Boolean> set to false to disable automatic index creation for all models associated with this connection.

Returns:

- <Connection> the created Connection object

See:

- Connection#open
- Connection#openSet

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

Each `connection` instance maps to a single database. This method is helpful when mangaging multiple db connections.

If arguments are passed, they are proxied to either Connection#open or Connection#openSet appropriately. This means we can pass `db`, `server`, and `replset` options to the driver. *Note that the `safe` option specified in your schema will overwrite the `safe` db option specified here unless you set your schemas `safe` option to `undefined`. See this for more information.*

Options passed take precedence over options included in connection strings.

Example:

```
// with mongodb:// URI
db = mongoose.createConnection('mongodb://user:pass@localhos

// and options
var opts = { db: { native_parser: true }}
db = mongoose.createConnection('mongodb://user:pass@localhos

// replica sets
db = mongoose.createConnection('mongodb://user:pass@localhos

// and options
var opts = { replset: { strategy: 'ping', rs_name: 'testSet'
db = mongoose.createConnection('mongodb://user:pass@localhos

// with [host, database_name[, port] signature
db = mongoose.createConnection('localhost', 'database', port

// and options
var opts = { server: { auto_reconnect: false }, user: 'usern
db = mongoose.createConnection('localhost', 'database', port

// initialize now, connect later
db = mongoose.createConnection();
db.open('localhost', 'database', port, [opts]);
```

show code

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

Mongoose#disconnect(`[fn]`)

Disconnects all connections.

Parameters:

- `[fn]` <Function> called after all connection close.

Returns:

- <MongooseThenable> pseudo-promise wrapper around this

show code

Mongoose#Document()

The Mongoose Document constructor.

Mongoose#DocumentProvider()

The Mongoose DocumentProvider constructor.

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js
querystream.js
connection.js
utils.js
browser.js
drivers/node-mongodb-native/collection.js
drivers/node-mongodb-native/connection.js
error/messages.js
error/validation.js
error.js
querycursor.js
virtualtype.js
schema.js
document.js
types/subdocument.js

Mongoose#Error()

The MongooseError constructor.

Mongoose#get(`key`)

Gets mongoose options

Parameters:

- `key` <String>

Example:

```
mongoose.get('test') // returns the 'test' value
```

Mongoose#model(`name, [schema], [collection], [skipInit]`)

Defines a model or retrieves it.

Parameters:

- `name` <String, Function> model name or class extending Model
- `[schema]` <Schema>
- `[collection]` <String> name (optional, inferred from model name)
- `[skipInit]` <Boolean> whether to skip initialization (defaults to false)

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

Models defined on the `mongoose` instance are available to all connection created by the same `mongoose` instance.

Example:

```
var mongoose = require('mongoose');

// define an Actor model with this mongoose instance
mongoose.model('Actor', new Schema({ name: String }));

// create a new connection
var conn = mongoose.createConnection(..);

// retrieve the Actor model
var Actor = conn.model('Actor');
```

When no `collection` argument is passed, Mongoose produces a collection name by passing the model `name` to the utils.toCollectionName method. This method pluralizes the name. If you don't like this behavior, either pass a collection name or set your schemas collection name option.

Example:

```
var schema = new Schema({ name: String }, { collection: 'acto

// or

schema.set('collection', 'actor');

// or

var collectionName = 'actor'
var M = mongoose.model('Actor', schema, collectionName)
```

show code

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

Mongoose#Model()

The Mongoose Model constructor.

Mongoose#modelNames()

Returns an array of model names created on this instance of Mongoose.

Returns:

- <Array>

Note:

Does not include names of models created using `connection.model()`.

show code

Mongoose()

Mongoose constructor.

The exports object of the `mongoose` module is an instance of this class. Most apps will only use this one instance.

show code

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

Mongoose#Mongoose()

The Mongoose constructor

The exports of the mongoose module is an instance of this class.

Example:

```
var mongoose = require('mongoose');
var mongoose2 = new mongoose.Mongoose();
```

Mongoose#plugin(`fn`, `[opts]`)

Declares a global plugin executed on all Schemas.

Parameters:

- `fn` <Function> plugin callback
- `[opts]` <Object> optional options

Returns:

- <Mongoose> this

See:

- plugins

Equivalent to calling `.plugin(fn)` on each Schema you create.

show code

☐ private

function Object() { [native code] }#Promise()

The Mongoose Promise constructor.

Mongoose#PromiseProvider()

Storage layer for mongoose promises

Mongoose#Query()

The Mongoose Query constructor.

Mongoose#Schema()

The Mongoose Schema constructor

Example:

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var CatSchema = new Schema(..);
```

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

Mongoose#SchemaType()

The Mongoose SchemaType constructor

Mongoose#set(`key`, `value`)

Sets mongoose options

Parameters:

- `key` <String>
- `value` <String, Function>

Example:

```
mongoose.set('test', value) // sets the 'test' option to `va
mongoose.set('debug', true) // enable logging collection meth
mongoose.set('debug', function(collectionName, methodName, ar
```

show code

()

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

Expose connection states for user-land

show code

Mongoose#VirtualType()

The Mongoose VirtualType constructor

Mongoose#connection

The default connection of the mongoose module.

Example:

```
var mongoose = require('mongoose');
mongoose.connect(...);
mongoose.connection.on('error', cb);
```

This is the connection used by default for every model created using mongoose.model.

show code

Returns:

- <Connection>

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

Mongoose#mongo

The node-mongodb-native driver Mongoose uses.

show code

Mongoose#mquery

The mquery query builder Mongoose uses.

show code

Mongoose#SchemaTypes

The various Mongoose SchemaTypes.

Note:

Alias of mongoose.Schema.Types for backwards compatibility.

show code

See:

- Schema.SchemaTypes

private

Mongoose#Types

mongoose

The various Mongoose Types.

Example:

```
var mongoose = require('mongoose');
var array = mongoose.Types.Array;
```

Types:

- ObjectId
- Buffer
- SubDocument
- Array
- DocumentArray

Using this exposed access to the `ObjectId` type, we can construct ids on demand.

```
var ObjectId = mongoose.Types.ObjectId;
var id1 = new ObjectId;
```

show code

Mongoose#version

The Mongoose version

show code

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

querystream.js

QueryStream#destroy(`[err]`)

Destroys the stream, closing the underlying cursor, which emits the close event. No more events will be emitted after the close event.

Parameters:

- `[err]` <Error>

show code

QueryStream#pause()

Pauses this stream.

show code

QueryStream#pipe()

Pipes this query stream into another stream. This method is inherited from NodeJS Streams.

See:

- NodeJS

Example:

☐ private

mongoose

```
query.stream().pipe(writeStream [, options])
```

QueryStream(query, [options])

Provides a Node.js 0.8 style ReadStream interface for Queries.

Parameters:

- `query` <Query>
- `[options]` <Object>

Inherits:

- NodeJS Stream

Events:

- `data`: emits a single Mongoose document
- `error`: emits when an error occurs during streaming. This will emit *before* the `close` event.
- `close`: emits when the stream reaches the end of the cursor or an error occurs, or the stream is manually `destroy`ed. After this event, no more events are emitted.

```
var stream = Model.find().stream();

stream.on('data', function (doc) {
  // do something with the mongoose document
}).on('error', function (err) {
  // handle the error
}).on('close', function () {
  // the stream is closed
});
```

☐ private

mongoose

The stream interface allows us to simply "plug-in" to other *Node.js 0.8* style write streams.

```
Model.where('created').gte(twoWeeksAgo).stream().pipe(writeSt
```

Valid options

- `transform`: optional function which accepts a mongoose document. The return value of the function will be emitted on `data`.

Example

```
// JSON.stringify all documents before emitting
var stream = Thing.find().stream({ transform: JSON.stringify
stream.pipe(writeStream);
```

*NOTE: plugging into an HTTP response will *not* work out of the box. Those streams expect only strings or buffers to be emitted, so first formatting our documents as strings/buffers is necessary.*

NOTE: these streams are Node.js 0.8 style read streams which differ from Node.js 0.10 style. Node.js 0.10 streams are not well tested yet and are not guaranteed to work.

show code

QueryStream#resume()

Resumes this stream.

show code

☐ private

mongoose

QueryStream#paused

Flag stating whether or not this stream is paused.

show code

QueryStream#readable

Flag stating whether or not this stream is readable.

show code

connection.js

Connection(base)

Connection constructor

Parameters:

- base <Mongoose> a mongoose instance

Inherits:

- NodeJS EventEmitter

Events:

☐ private

mongoose

- [home]
- [FAQ]
- [plugins]
- [change log]
- [support]
- [fork]
- [guide]
- [API docs]
- [quick start]
- [contributors]
- [prior releases]

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

- `connecting`: Emitted when `connection.{open,openSet}()` is executed on this connection.
- `connected`: Emitted when this connection successfully connects to the db. May be emitted *multiple* times in `reconnected` scenarios.
- `open`: Emitted after we `connected` and `onOpen` is executed on all of this connections models.
- `disconnecting`: Emitted when `connection.close()` was executed.
- `disconnected`: Emitted after getting disconnected from the db.
- `close`: Emitted after we `disconnected` and `onClose` executed on all of this connections models.
- `reconnected`: Emitted after we `connected` and subsequently `disconnected`, followed by successfully another successfull connection.
- `error`: Emitted when an error occurs on this connection.
- `fullsetup`: Emitted in a replica-set scenario, when primary and at least one seconaries specified in the connection string are connected.
- `all`: Emitted in a replica-set scenario, when all nodes specified in the connection string are connected.

For practical reasons, a Connection equals a Db.

show code

Connection#open(`connection_string`, `[database]`, `[port]`, `[options]`, `[callback]`)

Opens the connection to MongoDB.

Parameters:

☐ private

- `connection_string` <String> mongodb://uri or the host to which you are connecting
- `[database]` <String> database name
- `[port]` <Number> database port
- `[options]` <Object> options
- `[callback]` <Function>

See:

- node-mongodb-native
- http://mongodb.github.com/node-mongodb-native/api-generated/db.html#authenticate

`options` is a hash with the following possible properties:

```
config  - passed to the connection config instance
db      - passed to the connection db instance
server  - passed to the connection server instance(s)
replset - passed to the connection ReplSet instance
user    - username for authentication
pass    - password for authentication
auth    - options for authentication (see http://mongodb.gitl
```

Notes:

Mongoose forces the db option `forceServerObjectId` false and cannot be overridden.
Mongoose defaults the server `auto_reconnect` options to true which can be overridden.
See the node-mongodb-native driver instance for options that it understands.

Options passed take precedence over options included in connection strings.

show code

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

Connection#dropDatabase(`callback`)

Helper for `dropDatabase()`.

Parameters:

- `callback` <Function>

Returns:

- <Promise>

show code

Connection#openSet(`uris`, `[database]`, `[options]`, `[callback]`)

Opens the connection to a replica set.

Parameters:

- `uris` <String> MongoDB connection string
- `[database]` <String> database name if not included in `uris`
- `[options]` <Object> passed to the internal driver
- `[callback]` <Function>

See:

- node-mongodb-native
- http://mongodb.github.com/node-mongodb-native/api-generated/db.html#authenticate

Example:

```
var db = mongoose.createConnection();
db.openSet("mongodb://user:pwd@localhost:27020,localhost:270
```

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

The database name and/or auth need only be included in one URI.
The `options` is a hash which is passed to the internal driver connection object.

Valid `options`

```
db      - passed to the connection db instance
server  - passed to the connection server instance(s)
replset - passed to the connection ReplSetServer instance
user    - username for authentication
pass    - password for authentication
auth    - options for authentication (see http://mongodb.gitl
mongos  - Boolean - if true, enables High Availability suppoı
```

Options passed take precedence over options included in connection strings.

Notes:

If connecting to multiple mongos servers, set the `mongos` option to true.

```
conn.open('mongodb://mongosA:27501,mongosB:27501', { mongos:
```

Mongoose forces the db option `forceServerObjectId` false and cannot be overridden.
Mongoose defaults the server `auto_reconnect` options to true which can be overridden.
See the node-mongodb-native driver instance for options that it understands.

Options passed take precedence over options included in connection strings.

show code

☐ private

mongoose

Connection#close(`[callback]`)

Closes the connection

Parameters:

- `[callback]` <Function> optional

Returns:

- <Connection> self

show code

Connection#collection(`name`, `[options]`)

Retrieves a collection, creating it if not cached.

Parameters:

- `name` <String> of the collection
- `[options]` <Object> optional collection options

Returns:

- <Collection> collection instance

Not typically needed by applications. Just talk to your collection through your model.

show code

☐ private

Connection#model(`name`, `[schema]`, `[collection]`)

Defines or retrieves a model.

Parameters:

- `name` <String> the model name
- `[schema]` <Schema> a schema. necessary when defining a model
- `[collection]` <String> name of mongodb collection (optional) if not given it will be induced from model name

Returns:

- <Model> The compiled model

See:

- Mongoose#model

```
var mongoose = require('mongoose');
var db = mongoose.createConnection(..);
db.model('Venue', new Schema(..));
var Ticket = db.model('Ticket', new Schema(..));
var Venue = db.model('Venue');
```

When no `collection` argument is passed, Mongoose produces a collection name by passing the model `name` to the utils.toCollectionName method. This method pluralizes the name. If you don't like this behavior, either pass a collection name or set your schemas collection name option.

Example:

```
var schema = new Schema({ name: String }, { collection: 'acto

// or

schema.set('collection', 'actor');

// or
```

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

```
var collectionName = 'actor'
var M = conn.model('Actor', schema, collectionName)
```

show code

Connection#modelNames()

Returns an array of model names created on this connection.

Returns:

- <Array>

show code

Connection#config

A hash of the global options that are associated with this connection

show code

Connection#db

The mongodb.Db instance, set when the connection is opened

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

show code

Connection#collections

A hash of the collections associated with this connection

show code

Connection#readyState

Connection ready state

- 0 = disconnected
- 1 = connected
- 2 = connecting
- 3 = disconnecting

Each state change emits its associated event name.

Example

```
conn.on('connected', callback);
conn.on('disconnected', callback);
```

show code

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

utils.js

exports.pluralization

Pluralization rules.

show code

These rules are applied while processing the argument to `toCollectionName`.

exports.uncountables

Uncountable words.

show code

These words are applied while processing the argument to `toCollectionName`.

browser.js

function Object() { [native code] }#Promise()

The Mongoose Promise constructor.

☐ private

exports.Document()

The Mongoose browser Document constructor.

exports.Error()

The MongooseError constructor.

exports.PromiseProvider()

Storage layer for mongoose promises

exports.Schema()

The Mongoose Schema constructor

Example:

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var CatSchema = new Schema(..);
```

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

exports.VirtualType()

The Mongoose VirtualType constructor

exports#SchemaTypes

The various Mongoose SchemaTypes.

Note:

Alias of mongoose.Schema.Types for backwards compatibility.

show code

See:

- Schema.SchemaTypes

exports#Types

The various Mongoose Types.

Example:

```
var mongoose = require('mongoose');
var array = mongoose.Types.Array;
```

☐ private

mongoose

Types:

- ObjectId
- Buffer
- SubDocument
- Array
- DocumentArray

Using this exposed access to the `ObjectId` type, we can construct ids on demand.

```
var ObjectId = mongoose.Types.ObjectId;
var id1 = new ObjectId;
```

show code

drivers/node-mongodb-native/collection.js

function Object() { [native code] }#$format()

Formatter for debug print args

function Object() { [native code] }#$print()

Debug print helper

☐ private

NativeCollection#getIndexes(`callback`)

Retreives information about this collections indexes.

Parameters:

- `callback` <Function>

drivers/node-mongodb-native/connection.js

NativeConnection#useDb(`name`)

Switches to a different database using the same connection pool.

Parameters:

- `name` <String> The database name

Returns:

- <Connection> New Connection Object

Returns a new connection object, with the new db.

show code

NativeConnection.STATES

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

Expose the possible connection states.

show code

error/messages.js

MongooseError.messages()

The default built-in validator error messages. These may be customized.

show code

```
// customize within each schema or globally like so
var mongoose = require('mongoose');
mongoose.Error.messages.String.enum  = "Your custom message 
```

As you might have noticed, error messages support basic templating

- `{PATH}` is replaced with the invalid document path
- `{VALUE}` is replaced with the invalid value
- `{TYPE}` is replaced with the validator type such as "regexp", "min", or "user defined"
- `{MIN}` is replaced with the declared min value for the Number.min validator
- `{MAX}` is replaced with the declared max value for the Number.max validator

Click the "show code" link below to see all defaults.

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

error/validation.js

ValidationError#toString()

Console.log helper

show code

error.js

MongooseError(`msg`)

MongooseError constructor

Parameters:

- `msg` <String> Error message

Inherits:

- Error

show code

MongooseError.messages

☐ private

The default built-in validator error messages.

show code

See:

- Error.messages

querycursor.js

QueryCursor#close(`callback`)

Marks this cursor as closed. Will stop streaming and subsequent calls to `next()` will error.

Parameters:

- `callback` <Function>

Returns:

- <Promise>

See:

- MongoDB

QueryCursor#eachAsync(`fn`, `[callback]`)

Execute `fn` for every document in the cursor. If `fn` returns a promise, will wait for the promise to resolve before iterating on to the next one.

private

mongoose

Returns a promise that resolves when done.

Parameters:

- `fn` <Function>
- `[callback]` <Function> executed when all docs have been processed

Returns:

- <Promise>

QueryCursor#map(`fn`)

Registers a transform function which subsequently maps documents retrieved
via the streams interface or `.next()`

Parameters:

- `fn` <Function>

Returns:

- <QueryCursor>

Example

```
// Map documents returned by `data` events
Thing.
  find({ name: /^hello/ }).
  cursor().
  map(function (doc) {
   doc.foo = "bar";
   return doc;
  })
  on('data', function(doc) { console.log(doc.foo); });
```

☐ private

```
// Or map documents returned by `.next()`
var cursor = Thing.find({ name: /^hello/ }).
  cursor().
  map(function (doc) {
    doc.foo = "bar";
    return doc;
  });
cursor.next(function(error, doc) {
  console.log(doc.foo);
});
```

QueryCursor#next(`callback`)

Get the next document from this cursor. Will return `null` when there are no documents left.

Parameters:

- `callback` <Function>

Returns:

- <Promise>

QueryCursor(`query`, `options`)

A QueryCursor is a concurrency primitive for processing query results one document at a time. A QueryCursor fulfills the Node.js streams3 API,
in addition to several other mechanisms for loading documents from

☐ private

MongoDB
one at a time.

Parameters:

- `query` <Query>
- `options` <Object> query options passed to `.find()`

Inherits:

- Readable

Events:

- `cursor`: Emitted when the cursor is created
- `error`: Emitted when an error occurred
- `data`: Emitted when the stream is flowing and the next doc is ready
- `end`: Emitted when the stream is exhausted

Unless you're an advanced user, do **not** instantiate this class directly. Use `Query#cursor()` instead.

show code

virtualtype.js

VirtualType#applyGetters(`value`, `scope`)

Applies getters to `value` using optional `scope`.

Parameters:

- `value` <Object>
- `scope` <Object>

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

Returns:

- <T> the value after applying all getters

show code

VirtualType#applySetters(`value`, `scope`)

Applies setters to `value` using optional `scope`.

Parameters:

- `value` <Object>
- `scope` <Object>

Returns:

- <T> the value after applying all setters

show code

VirtualType#get(`fn`)

Defines a getter.

Parameters:

- `fn` <Function>

Returns:

- <VirtualType> this

☐ private

Example:

```
var virtual = schema.virtual('fullname');
virtual.get(function () {
  return this.name.first + ' ' + this.name.last;
});
```

show code

VirtualType#set(`fn`)

Defines a setter.

Parameters:

- `fn` <Function>

Returns:

- <VirtualType> this

Example:

```
var virtual = schema.virtual('fullname');
virtual.set(function (v) {
  var parts = v.split(' ');
  this.name.first = parts[0];
  this.name.last = parts[1];
});
```

show code

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

VirtualType()

VirtualType constructor

This is what mongoose uses to define virtual attributes via `Schema.prototype.virtual`.

Example:

```
var fullname = schema.virtual('fullname');
fullname instanceof mongoose.VirtualType // true
```

show code

schema.js

Schema#add(`obj`, `prefix`)

Adds key path / schema type pairs to this schema.

Parameters:

- `obj` <Object>
- `prefix` <String>

Example:

```
var ToySchema = new Schema;
ToySchema.add({ name: 'string', color: 'string', price: 'numb
```

private show code

Schema#eachPath(`fn`)

Iterates the schemas paths similar to Array#forEach.

Parameters:

- `fn` <Function> callback function

Returns:

- <Schema> this

The callback is passed the pathname and schemaType as arguments on each iteration.

show code

Schema#get(`key`)

Gets a schema option.

Parameters:

- `key` <String> option name

show code

Schema#index(`fields`, `[options]`, `[options.expires=null]`)

☐ private

Defines an index (most likely compound) for this schema.

Parameters:

- `fields` <Object>
- `[options]` <Object> Options to pass to MongoDB driver's `createIndex()` function
- `[options.expires=null]` <String> Mongoose-specific syntactic sugar, uses ms to convert `expires` option into seconds for the `expireAfterSeconds` in the above link.

Example

```
schema.index({ first: 1, last: -1 })
```

show code

Schema#indexes()

Compiles indexes from fields and schema-level indexes

show code

Schema#loadClass(`model`)

Loads an ES6 class into a schema. Maps setters + getters, static methods, and instance methods to schema virtuals, statics, and methods.

Parameters:

private

mongoose

- `model` <Function>

show code

Schema#method(`method`, `[fn]`)

Adds an instance method to documents constructed from Models compiled from this schema.

Parameters:

- `method` <String, Object> name
- `[fn]` <Function>

Example

```
var schema = kittySchema = new Schema(..);

schema.method('meow', function () {
  console.log('meeeeeoooooooooooow');
})

var Kitty = mongoose.model('Kitty', schema);

var fizz = new Kitty;
fizz.meow(); // meeeeeooooooooooooow
```

If a hash of name/fn pairs is passed as the only argument, each name/fn pair will be added as methods.

```
schema.method({
    purr: function () {}
  , scratch: function () {}
});

// later
```

private

```
fizz.purr();
fizz.scratch();
```

show code

mongoose

Schema#path(`path`, `constructor`)

Gets/sets schema paths.

Parameters:

- `path` <String>
- `constructor` <Object>

Sets a path (if arity 2)
Gets a path (if arity 1)

Example

```
schema.path('name') // returns a SchemaType
schema.path('name', Number) // changes the schemaType of `nar
```

show code

Schema#pathType(`path`)

Returns the pathType of `path` for this schema.

Parameters:

private

- `path` <String>

Returns:

- <String>

Given a path, returns whether it is a real, virtual, nested, or ad-hoc/undefined path.

show code

Schema#plugin(`plugin`, `[opts]`)

Registers a plugin for this schema.

Parameters:

- `plugin` <Function> callback
- `[opts]` <Object>

See:

- plugins

show code

Schema#post(`method`, `fn`)

Defines a post hook for the document

Parameters:

private `method` <String> name of the method to hook

- fn <Function> callback

See:

- middleware
- hooks.js
- kareem

```
var schema = new Schema(..);
schema.post('save', function (doc) {
  console.log('this fired after a document was saved');
});

shema.post('find', function(docs) {
  console.log('this fired after you run a find query');
});

var Model = mongoose.model('Model', schema);

var m = new Model(..);
m.save(function(err) {
  console.log('this fires after the `post` hook');
});

m.find(function(err, docs) {
  console.log('this fires after the post find hook');
});
```

show code

Schema#pre(`method`, `callback`)

Defines a pre hook for the document.

Parameters:

private

- method <String>

mongoose

- `callback` <Function>

See:

- hooks.js

Example

```
var toySchema = new Schema(..);

toySchema.pre('save', function (next) {
  if (!this.created) this.created = new Date;
  next();
})

toySchema.pre('validate', function (next) {
  if (this.name !== 'Woody') this.name = 'Woody';
  next();
})
```

show code

Schema#queue(`name, args`)

Adds a method call to the queue.

Parameters:

- `name` <String> name of the document method to call later
- `args` <Array> arguments to pass to the method

show code

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

Schema#remove(`path`)

Removes the given `path` (or [`paths`]).

Parameters:

- `path` <String, Array>

show code

Schema#requiredPaths(`invalidate`)

Returns an Array of path strings that are required by this schema.

Parameters:

- `invalidate` <Boolean> refresh the cache

Returns:

- <Array>

show code

Schema(`definition`, [`options`])

Schema constructor.

Parameters:

private `definition` <Object>

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

- `[options]` <Object>

Inherits:

- NodeJS EventEmitter

Events:

- `init`: Emitted after the schema is compiled into a `Model`.

Example:

```
var child = new Schema({ name: String });
var schema = new Schema({ name: String, age: Number, childrer
var Tree = mongoose.model('Tree', schema);

// setting schema options
new Schema({ name: String }, { _id: false, autoIndex: false 
```

Options:

- autoIndex: bool - defaults to null (which means use the connection's autoIndex option)
- bufferCommands: bool - defaults to true
- capped: bool - defaults to false
- collection: string - no default
- emitIndexErrors: bool - defaults to false.
- id: bool - defaults to true
- _id: bool - defaults to true
- `minimize`: bool - controls document#toObject behavior when called manually - defaults to true
- read: string
- safe: bool - defaults to true.
- shardKey: bool - defaults to `null`
- strict: bool - defaults to true
- toJSON - object - no default
- toObject - object - no default
- typeKey - string - defaults to 'type'
- useNestedStrict - boolean - defaults to false
- validateBeforeSave - bool - defaults to `true`
- versionKey: string - defaults to "__v"

☐ private

mongoose

Note:

When nesting schemas, (`children` in the example above), always declare the child schema first before passing it into its parent.

show code

Schema#set(`key`, `[value]`)

Sets/gets a schema option.

Parameters:

- `key` <String> option name
- `[value]` <Object> if not passed, the current option value is returned

See:

- Schema

Example

```
schema.set('strict'); // 'true' by default
schema.set('strict', false); // Sets 'strict' to false
schema.set('strict'); // 'false'
```

show code

Schema#static(`name`, `[fn]`)

☐ private

mongoose

Adds static "class" methods to Models compiled from this schema.

Parameters:

- `name` <String, Object>
- `[fn]` <Function>

Example

```
var schema = new Schema(..);
schema.static('findByName', function (name, callback) {
  return this.find({ name: name }, callback);
});

var Drink = mongoose.model('Drink', schema);
Drink.findByName('sanpellegrino', function (err, drinks) {
  //
});
```

If a hash of name/fn pairs is passed as the only argument, each name/fn pair will be added as statics.

show code

Schema#virtual(`name, [options]`)

Creates a virtual type with the given name.

Parameters:

- `name` <String>
- `[options]` <Object>

Returns:

- <VirtualType>

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

show code

Schema#virtualpath(`name`)

Returns the virtual type with the given `name`.

Parameters:

- `name` <String>

Returns:

- <VirtualType>

show code

Schema.indexTypes()

The allowed index types

show code

Schema.reserved

Reserved document keys.

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

Keys in this object are names that are rejected in schema declarations b/c they conflict with mongoose functionality. Using these key name will throw an error.

```
on, emit, _events, db, get, set, init, isNew, errors, schema
```

NOTE: Use of these terms as method names is permitted, but play at your own risk, as they may be existing mongoose document methods you are stomping on.

```
var schema = new Schema(..);
 schema.methods.init = function () {} // potentially breaking
```

Schema.Types

The various built-in Mongoose Schema Types.

show code

Example:

```
var mongoose = require('mongoose');
var ObjectId = mongoose.Schema.Types.ObjectId;
```

Types:

- String
- Number
- Boolean | Bool
- Array
- Buffer
- Date
- ObjectId | Oid

private

mongoose

- Mixed

Using this exposed access to the `Mixed` SchemaType, we can use them in our schema.

```
var Mixed = mongoose.Schema.Types.Mixed;
new mongoose.Schema({ _user: Mixed })
```

Schema#obj

The original object passed to the schema constructor

Example:

```
var schema = new Schema({ a: String }).add({ b: String });
schema.obj; // { a: String }
```

show code

document.js

MISSING method name

Don't run validation on this path or persist changes to this path.

Parameters:

☐ private

mongoose

- `path` <String> the path to ignore

Example:

```
doc.foo = null;
doc.$ignore('foo');
doc.save() // changes to foo will not be persisted and valid
```

show code

function Object() { [native code] }#$isDefault(`[path]`)

Checks if a path is set to its default.

Parameters:

- `[path]` <String>

Returns:

- <Boolean>

Example

```
MyModel = mongoose.model('test', { name: { type: String, def
var m = new MyModel();
m.$isDefault('name');                  // true
```

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

Document#depopulate(`path`)

Takes a populated field and returns it to its unpopulated state.

Parameters:

- `path` <String>

See:

- Document.populate

Example:

```
Model.findOne().populate('author').exec(function (err, doc) ·
  console.log(doc.author.name); // Dr.Seuss
  console.log(doc.depopulate('author'));
  console.log(doc.author); // '5144cf8050f071d979c118a7'
})
```

If the path was not populated, this is a no-op.

show code

Document#equals(`doc`)

Returns true if the Document stores the same data as doc.

Parameters:

- `doc` <Document> a document to compare

Returns:

- <Boolean>

☐ private

Documents are considered equal when they have matching `_id`s, unless neither
document has an `_id`, in which case this function falls back to using `deepEqual()`.

show code

Document#execPopulate()

Explicitly executes population and returns a promise. Useful for ES2015 integration.

Returns:

- <Promise> promise that resolves to the document when population is done

See:

- Document.populate

Example:

```
var promise = doc.
  populate('company').
  populate({
    path: 'notes',
    match: /airline/,
    select: 'text',
    model: 'modelName'
    options: opts
  }).
  execPopulate();

// summary
doc.execPopulate().then(resolve, reject);
```

☐ private

show code

Document#get(`path`, `[type]`)

Returns the value of a path.

Parameters:

- `path` <String>
- `[type]` <Schema, String, Number, Buffer, *> optionally specify a type for on-the-fly attributes

Example

```
// path
doc.get('age') // 47

// dynamic casting to a string
doc.get('age', String) // "47"
```

show code

Document#init(`doc`, `fn`)

Initializes the document without setters or marking anything modified.

Parameters:

- `doc` <Object> document returned by mongo
- `fn` <Function> callback

private

Called internally after a document is returned from mongodb.

show code

Document#inspect()

Helper for console.log

show code

Document#invalidate(`path`, `errorMsg`, `value`, `[kind]`)

Marks a path as invalid, causing validation to fail.

Parameters:

- `path` <String> the field to invalidate
- `errorMsg` <String, Error> the error which states the reason `path` was invalid
- `value` <Object, String, Number, T> optional invalid value
- `[kind]` <String> optional `kind` property for the error

Returns:

- <ValidationError> the current ValidationError, with all currently invalidated paths

The `errorMsg` argument will become the message of the `ValidationError`.

The `value` argument (if passed) will be available through the `ValidationError.value` property.

☐ private

```
doc.invalidate('size', 'must be less than 20', 14);

doc.validate(function (err) {
  console.log(err)
  // prints
  { message: 'Validation failed',
    name: 'ValidationError',
    errors:
     { size:
        { message: 'must be less than 20',
          name: 'ValidatorError',
          path: 'size',
          type: 'user defined',
          value: 14 } } }
})
```

show code

Document#isDirectModified(`path`)

Returns true if `path` was directly set and modified, else false.

Parameters:

- `path` <String>

Returns:

- <Boolean>

Example

```
doc.set('documents.0.title', 'changed');
doc.isDirectModified('documents.0.title') // true
doc.isDirectModified('documents') // false
```

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

show code

Document#isInit(`path`)

Checks if `path` was initialized.

Parameters:

- `path` <String>

Returns:

- <Boolean>

show code

Document#isModified(`[path]`)

Returns true if this document was modified, else false.

Parameters:

- `[path]` <String> optional

Returns:

- <Boolean>

If `path` is given, checks if a path or any full path containing `path` as part of its path chain has been modified.

Example

private

```
doc.set('documents.0.title', 'changed');
doc.isModified()                        // true
doc.isModified('documents')             // true
doc.isModified('documents.0.title')     // true
doc.isModified('documents otherProp') // true
doc.isDirectModified('documents')       // false
```

show code

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

Document#isSelected(`path`)

Checks if `path` was selected in the source query which initialized this document.

Parameters:

- `path` <String>

Returns:

- <Boolean>

Example

```
Thing.findOne().select('name').exec(function (err, doc) {
   doc.isSelected('name') // true
   doc.isSelected('age')  // false
})
```

show code

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

Document#markModified(`path`)

Marks the path as having pending changes to write to the db.

Parameters:

- `path` <String> the path to mark modified

Very helpful when using Mixed types.

Example:

```
doc.mixed.type = 'changed';
doc.markModified('mixed.type');
doc.save() // changes to mixed.type are now persisted
```

show code

Document#modifiedPaths()

Returns the list of paths that have been modified.

Returns:

- <Array>

show code

Document#populate(`[path], [callback]`)

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

Populates document references, executing the `callback` when complete.
If you want to use promises instead, use this function with `execPopulate()`

Parameters:

- `[path]` <String, Object> The path to populate or an options object
- `[callback]` <Function> When passed, population is invoked

Returns:

- <Document> this

See:

- Model.populate
- Document.execPopulate

Example:

```
doc
.populate('company')
.populate({
  path: 'notes',
  match: /airline/,
  select: 'text',
  model: 'modelName'
  options: opts
}, function (err, user) {
  assert(doc._id === user._id) // the document itself is pas
})

// summary
doc.populate(path)                   // not executed
doc.populate(options);               // not executed
doc.populate(path, callback)         // executed
doc.populate(options, callback);     // executed
doc.populate(callback);              // executed
doc.populate(options).execPopulate() // executed, returns pr
```

NOTE:
private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js
querystream.js
connection.js
utils.js
browser.js
drivers/node-mongodb-native/collection.js
drivers/node-mongodb-native/connection.js
error/messages.js
error/validation.js
error.js
querycursor.js
virtualtype.js
schema.js
document.js
types/subdocument.js

Population does not occur unless a `callback` is passed *or* you explicitly call `execPopulate()`.
Passing the same path a second time will overwrite the previous path options.
See Model.populate() for explaination of options.

show code

Document#populated(`path`)

Gets _id(s) used during population of the given `path`.

Parameters:

- `path` <String>

Returns:

- <Array, ObjectId, Number, Buffer, String, undefined>

Example:

```
Model.findOne().populate('author').exec(function (err, doc) 
  console.log(doc.author.name)         // Dr.Seuss
  console.log(doc.populated('author')) // '5144cf8050f071d97
})
```

If the path was not populated, undefined is returned.

show code

private

mongoose

- [home]
- [FAQ]
- [plugins]
- [change log]
- [support]
- [fork]
- [guide]
- [API docs]
- [quick start]
- [contributors]
- [prior releases]

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

Document#set(`path`, `val`, `[type]`, `[options]`)

Sets the value of a path, or many paths.

Parameters:

- `path` <String, Object> path or object of key/vals to set
- `val` <Any> the value to set
- `[type]` <Schema, String, Number, Buffer, *> optionally specify a type for "on-the-fly" attributes
- `[options]` <Object> optionally specify options that modify the behavior of the set

Example:

```
// path, value
doc.set(path, value)

// object
doc.set({
    path  : value
  , path2 : {
       path  : value
    }
})

// on-the-fly cast to number
doc.set(path, value, Number)

// on-the-fly cast to string
doc.set(path, value, String)

// changing strict mode behavior
doc.set(path, value, { strict: false });
```

show code

☐ private

mongoose

Document#toJSON(`options`)

The return value of this method is used in calls to JSON.stringify(doc).

Parameters:

- `options` <Object>

Returns:

- <Object>

See:

- Document#toObject

This method accepts the same options as Document#toObject. To apply the options to every document of your schema by default, set your schemas `toJSON` option to the same argument.

```
schema.set('toJSON', { virtuals: true })
```

See schema options for details.

show code

Document#toObject(`[options]`)

Converts this document into a plain javascript object, ready for storage in MongoDB.

Parameters:

- `[options]` <Object>

Returns:

private

- <Object> js object

See:

- mongodb.Binary

Buffers are converted to instances of mongodb.Binary for proper storage.

Options:

- `getters` apply all getters (path and virtual getters)
- `virtuals` apply virtual getters (can override `getters` option)
- `minimize` remove empty objects (defaults to true)
- `transform` a transform function to apply to the resulting document before returning
- `depopulate` depopulate any populated paths, replacing them with their original refs (defaults to false)
- `versionKey` whether to include the version key (defaults to true)
- `retainKeyOrder` keep the order of object keys. If this is set to true, `Object.keys(new Doc({ a: 1, b: 2}).toObject())` will always produce `['a', 'b']` (defaults to false)

Getters/Virtuals

Example of only applying path getters

```
doc.toObject({ getters: true, virtuals: false })
```

Example of only applying virtual getters

```
doc.toObject({ virtuals: true })
```

Example of applying both path and virtual getters

```
doc.toObject({ getters: true })
```

To apply these options to every document of your schema by default, set your schemas `toObject` option to the same argument.

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

mongoose

```
schema.set('toObject', { virtuals: true })
```

Transform

We may need to perform a transformation of the resulting object based on some criteria, say to remove some sensitive information or return a custom object. In this case we set the optional `transform` function.

Transform functions receive three arguments

```
function (doc, ret, options) {}
```

- `doc` The mongoose document which is being converted
- `ret` The plain object representation which has been converted
- `options` The options in use (either schema options or the options passed inline)

Example

```
// specify the transform schema option
if (!schema.options.toObject) schema.options.toObject = {};
schema.options.toObject.transform = function (doc, ret, optic
  // remove the _id of every document before returning the re
  delete ret._id;
  return ret;
}

// without the transformation in the schema
doc.toObject(); // { _id: 'anId', name: 'Wreck-it Ralph' }

// with the transformation
doc.toObject(); // { name: 'Wreck-it Ralph' }
```

With transformations we can do a lot more than remove properties. We can even return completely new customized objects:

```
if (!schema.options.toObject) schema.options.toObject = {};
schema.options.toObject.transform = function (doc, ret, optic
  return { movie: ret.name }
```

private

mongoose

```
}

// without the transformation in the schema
doc.toObject(); // { _id: 'anId', name: 'Wreck-it Ralph' }

// with the transformation
doc.toObject(); // { movie: 'Wreck-it Ralph' }
```

Note: if a transform function returns `undefined`, the return value will be ignored.

Transformations may also be applied inline, overridding any transform set in the options:

```
function xform (doc, ret, options) {
  return { inline: ret.name, custom: true }
}

// pass the transform as an inline option
doc.toObject({ transform: xform }); // { inline: 'Wreck-it Ra
```

*Note: if you call `toObject` and pass any options, the transform declared in your schema options will **not** be applied. To force its application pass `transform: true`*

```
if (!schema.options.toObject) schema.options.toObject = {};
schema.options.toObject.hide = '_id';
schema.options.toObject.transform = function (doc, ret, optio
  if (options.hide) {
    options.hide.split(' ').forEach(function (prop) {
      delete ret[prop];
    });
  }
  return ret;
}

var doc = new Doc({ _id: 'anId', secret: 47, name: 'Wreck-it
doc.toObject();                                        // { s
doc.toObject({ hide: 'secret _id' });                  // { _
doc.toObject({ hide: 'secret _id', transform: true }); // { n
```

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

Transforms are applied *only to the document and are not applied to sub-documents*.

Transforms, like all of these options, are also available for `toJSON`.

See schema options for some more details.

During save, no custom options are applied to the document before being sent to the database.

show code

Document#toString()

Helper for console.log

Document#unmarkModified(`path`)

Clears the modified state on the specified path.

Parameters:

- `path` <String> the path to unmark modified

Example:

```
doc.foo = 'bar';
doc.unmarkModified('foo');
doc.save() // changes to foo will not be persisted
```

☐ private

mongoose

show code

Document#update(`doc`, `options`, `callback`)

Sends an update command with this document `_id` as the query selector.

Parameters:

- `doc` <Object>
- `options` <Object>
- `callback` <Function>

Returns:

- <Query>

See:

- Model.update

Example:

```
weirdCar.update({$inc: {wheels:1}}, { w: 1 }, callback);
```

Valid options:

- same as in Model.update

show code

Document#validate(`optional`, `callback`)

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

Executes registered validation rules for this document.

Parameters:

- `optional` <Object> options internal options
- `callback` <Function> optional callback called after validation completes, passing an error if one occurred

Returns:

- <Promise> Promise

Note:

This method is called `pre` save and if a validation rule is violated, save is aborted and the error is returned to your `callback`.

Example:

```
doc.validate(function (err) {
  if (err) handleError(err);
  else // validation passed
});
```

show code

Document#validateSync(`pathsToValidate`)

Executes registered validation rules (skipping asynchronous validators) for this document.

Parameters:

- `pathsToValidate` <Array, string> only validate the given paths

Returns:

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

- <MongooseError, undefined> MongooseError if there are errors during validation, or undefined if there is no error.

Note:

This method is useful if you need synchronous validation.

Example:

```
var err = doc.validateSync();
if ( err ){
  handleError( err );
} else {
  // validation passed
}
```

show code

Document#errors

Hash containing current validation errors.

show code

Document#id

The string version of this documents _id.

Note:

☐ private

This getter exists on all documents by default. The getter can be disabled by setting the `id` option of its `Schema` to false at construction time.

```
new Schema({ name: String }, { id: false });
```

show code

See:

- Schema options

Document#isNew

Boolean flag specifying if the document is new.

show code

Document#schema

The documents schema.

show code

private types/subdocument.js

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

Subdocument#ownerDocument()

Returns the top level document of this sub-document.

Returns:

- <Document>

show code

Subdocument#remove(`[options]`, `[callback]`)

Null-out this subdoc

Parameters:

- `[options]` <Object>
- `[callback]` <Function> optional callback for compatibility with Document.prototype.remove

show code

types/array.js

MongooseArray#$shift()

Atomically shifts the array at most one time per document `save()`.

private

See:

- mongodb

NOTE:

Calling this mulitple times on an array before saving sends the same command as calling it once.
This update is implemented using the MongoDB $pop method which enforces this restriction.

```
doc.array = [1,2,3];

 var shifted = doc.array.$shift();
 console.log(shifted); // 1
 console.log(doc.array); // [2,3]

 // no affect
 shifted = doc.array.$shift();
 console.log(doc.array); // [2,3]

 doc.save(function (err) {
   if (err) return handleError(err);

   // we saved, now $shift works again
   shifted = doc.array.$shift();
   console.log(shifted ); // 2
   console.log(doc.array); // [3]
 })
```

MongooseArray#remove()

Alias of pull

See:

private MongooseArray#pull

- mongodb

mongoose

MongooseArray.$pop()

Pops the array atomically at most one time per document `save()`.

See:

- mongodb

NOTE:

Calling this mulitple times on an array before saving sends the same command as calling it once.
This update is implemented using the MongoDB $pop method which enforces this restriction.

```
doc.array = [1,2,3];

 var popped = doc.array.$pop();
 console.log(popped); // 3
 console.log(doc.array); // [1,2]

 // no affect
 popped = doc.array.$pop();
 console.log(doc.array); // [1,2]

 doc.save(function (err) {
   if (err) return handleError(err);

   // we saved, now $pop works again
   popped = doc.array.$pop();
   console.log(popped); // 2
   console.log(doc.array); // [1]
 })
```

☐ private

mongoose

- [home]
- [FAQ]
- [plugins]
- [change log]
- [support]
- [fork]
- [guide]
- [API docs]
- [quick start]
- [contributors]
- [prior releases]

MongooseArray.addToSet(`[args...]`)

Adds values to the array if not already present.

Parameters:

- `[args...]` <T>

Returns:

- <Array> the values that were added

Example:

```
console.log(doc.array) // [2,3,4]
var added = doc.array.addToSet(4,5);
console.log(doc.array) // [2,3,4,5]
console.log(added)     // [5]
```

MongooseArray.indexOf(`obj`)

Return the index of `obj` or `-1` if not found.

Parameters:

- `obj` <Object> the item to look for

Returns:

- <Number>

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

MongooseArray.inspect()

Helper for console.log

MongooseArray.nonAtomicPush(`[args...]`)

Pushes items to the array non-atomically.

Parameters:

- `[args...]` <T>

NOTE:

marks the entire array as modified, which if saved, will store it as a `$set` operation, potentially overwritting any changes that happen between when you retrieved the object and when you save it.

MongooseArray.pop()

Wraps `Array#pop` with proper change tracking.

See:

- MongooseArray#$pop

Note:

marks the entire array as modified which will pass the entire thing to $set potentially overwritting any changes that happen between when you retrieved the object and when you save it.

private

mongoose

MongooseArray.pull(`[args...]`)

Pulls items from the array atomically. Equality is determined by casting the provided value to an embedded document and comparing using the `Document.equals()` function.

Parameters:

- `[args...]` <T>

See:

- mongodb

Examples:

```
doc.array.pull(ObjectId)
doc.array.pull({ _id: 'someId' })
doc.array.pull(36)
doc.array.pull('tag 1', 'tag 2')
```

To remove a document from a subdocument array we may pass an object with a matching `_id`.

```
doc.subdocs.push({ _id: 4815162342 })
doc.subdocs.pull({ _id: 4815162342 }) // removed
```

Or we may passing the _id directly and let mongoose take care of it.

```
doc.subdocs.push({ _id: 4815162342 })
doc.subdocs.pull(4815162342); // works
```

The first pull call will result in a atomic operation on the database, if pull is called repeatedly without saving the document, a $set operation is used on the complete array instead, overwriting possible changes that happened on the database in the meantime.

private

mongoose

MongooseArray.push([args...])

Wraps `Array#push` with proper change tracking.

Parameters:

- `[args...]` <Object>

MongooseArray.set()

Sets the casted `val` at index `i` and marks the array modified.

Returns:

- <Array> this

Example:

```
// given documents based on the following
var Doc = mongoose.model('Doc', new Schema({ array: [Number]

var doc = new Doc({ array: [2,3,4] })

console.log(doc.array) // [2,3,4]

doc.array.set(1,"5");
console.log(doc.array); // [2,5,4] // properly cast to numbe
doc.save() // the change is saved

// VS not using array#set
doc.array[1] = "5";
console.log(doc.array); // [2,"5",4] // no casting
doc.save() // change is not saved
```

private

mongoose

MongooseArray.shift()

Wraps `Array#shift` with proper change tracking.

Example:

```
doc.array = [2,3];
var res = doc.array.shift();
console.log(res) // 2
console.log(doc.array) // [3]
```

Note:

marks the entire array as modified, which if saved, will store it as a `$set` operation, potentially overwritting any changes that happen between when you retrieved the object and when you save it.

MongooseArray.sort()

Wraps `Array#sort` with proper change tracking.

NOTE:

marks the entire array as modified, which if saved, will store it as a `$set` operation, potentially overwritting any changes that happen between when you retrieved the object and when you save it.

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

MongooseArray.splice()

Wraps `Array#splice` with proper change tracking and casting.

Note:

marks the entire array as modified, which if saved, will store it as a `$set` operation, potentially overwritting any changes that happen between when you retrieved the object and when you save it.

MongooseArray.toObject(`options`)

Returns a native js Array.

Parameters:

- `options` <Object>

Returns:

- <Array>

MongooseArray.unshift()

Wraps `Array#unshift` with proper change tracking.

Note:

marks the entire array as modified, which if saved, will store it as a `$set` operation, potentially overwritting any changes that happen between when you retrieved the object and when you save it.

☐ private

mongoose

types/documentarray.js

MongooseDocumentArray.create(`obj`)

Creates a subdocument casted to this schema.

Parameters:

- `obj` <Object> the value to cast to this arrays SubDocument schema

This is the same subdocument constructor used for casting.

MongooseDocumentArray.id(`id`)

Searches array items for the first document with a matching _id.

Parameters:

- `id` <ObjectId, String, Number, Buffer>

Returns:

- <EmbeddedDocument, null> the subdocument or null if not found.

Example:

```
var embeddedDoc = m.array.id(some_id);
```

☐ private

MongooseDocumentArray.inspect()

Helper for console.log

MongooseDocumentArray.toObject(`[options]`)

Returns a native js Array of plain js objects

Parameters:

- `[options]` <Object> optional options to pass to each documents <code>toObject</code> method call during conversion

Returns:

- <Array>

NOTE:

Each sub-document is converted to a plain object by calling its `#toObject` method.

types/buffer.js

MongooseBuffer.copy(`target`)

☐ private

Copies the buffer.

Parameters:

- `target` <Buffer>

Returns:

- <Number> The number of bytes copied.

Note:

`Buffer#copy` does not mark `target` as modified so you must copy from a `MongooseBuffer` for it to work as expected. This is a work around since `copy` modifies the target, not this.

MongooseBuffer.equals(`other`)

Determines if this buffer is equals to `other` buffer

Parameters:

- `other` <Buffer>

Returns:

- <Boolean>

MongooseBuffer.subtype(`subtype`)

Sets the subtype option and marks the buffer modified.

Parameters:

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

- subtype <Hex>

See:

- http://bsonspec.org/#/specification

SubTypes:

var bson = require('bson')
bson.BSON_BINARY_SUBTYPE_DEFAULT
bson.BSON_BINARY_SUBTYPE_FUNCTION
bson.BSON_BINARY_SUBTYPE_BYTE_ARRAY
bson.BSON_BINARY_SUBTYPE_UUID
bson.BSON_BINARY_SUBTYPE_MD5
bson.BSON_BINARY_SUBTYPE_USER_DEFINED

doc.buffer.subtype(bson.BSON_BINARY_SUBTYPE_UUID);

MongooseBuffer.toObject(`[subtype]`)

Converts this buffer to its Binary type representation.

Parameters:

- `[subtype]` <Hex>

Returns:

- <Binary>

See:

- http://bsonspec.org/#/specification

SubTypes:

var bson = require('bson')
bson.BSON_BINARY_SUBTYPE_DEFAULT
bson.BSON_BINARY_SUBTYPE_FUNCTION
bson.BSON_BINARY_SUBTYPE_BYTE_ARRAY

private

bson.BSON_BINARY_SUBTYPE_UUID
bson.BSON_BINARY_SUBTYPE_MD5
bson.BSON_BINARY_SUBTYPE_USER_DEFINED

doc.buffer.toObject(bson.BSON_BINARY_SUBTYPE_USER_DEFINED);

MongooseBuffer.write()

Writes the buffer.

types/objectid.js

ObjectId()

ObjectId type constructor

Example

```
var id = new mongoose.Types.ObjectId;
```

☐ private

types/embedded.js

EmbeddedDocument#inspect()

Helper for console.log

show code

EmbeddedDocument#invalidate(`path`, `err`)

Marks a path as invalid, causing validation to fail.

Parameters:

- `path` <String> the field to invalidate
- `err` <String, Error> error which states the reason `path` was invalid

Returns:

- <Boolean>

show code

EmbeddedDocument#ownerDocument()

Returns the top level document of this sub-document.

Returns:

☐ private

mongoose

- <Document>

show code

EmbeddedDocument#parent()

Returns this sub-documents parent document.

show code

EmbeddedDocument#parentArray()

Returns this sub-documents parent array.

show code

EmbeddedDocument#remove(`[options]`, `[fn]`)

Removes the subdocument from its parent array.

Parameters:

- `[options]` <Object>
- `[fn]` <Function>

☐ private

show code

EmbeddedDocument.markModified(`path`)

Marks the embedded doc modified.

show code

Parameters:

- `path` <String> the path which changed

Example:

```
var doc = blogpost.comments.id(hexstring);
doc.mixed.type = 'changed';
doc.markModified('mixed.type');
```

query.js

Query#$where(`js`)

Specifies a javascript function or expression to pass to MongoDBs query system.

Parameters:

- `js` <String, Function> javascript string or function

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js
querystream.js
connection.js
utils.js
browser.js
drivers/node-mongodb-native/collection.js
drivers/node-mongodb-native/connection.js
error/messages.js
error/validation.js
error.js
querycursor.js
virtualtype.js
schema.js
document.js
types/subdocument.js

Returns:

- <Query> this

See:

- $where

Example

```
query.$where('this.comments.length === 10 || this.name.lengt

// or

query.$where(function () {
  return this.comments.length === 10 || this.name.length ===
})
```

NOTE:

Only use `$where` when you have a condition that cannot be met using other MongoDB operators like `$lt`.
Be sure to read about all of its caveats before using.

Query#all([path], val)

Specifies an $all query condition.

Parameters:

- `[path]` <String>
- `val` <Number>

See:

- $all

private

mongoose

When called with one argument, the most recent path passed to `where()` is used.

Query#and(`array`)

Specifies arguments for a `$and` condition.

Parameters:

- `array` <Array> array of conditions

Returns:

- <Query> this

See:

- $and

Example

```
query.and([{ color: 'green' }, { status: 'ok' }])
```

Query#batchSize(`val`)

Specifies the batchSize option.

Parameters:

- `val` <Number>

☐ private

See:

- batchSize

Example

```
query.batchSize(100)
```

Note

Cannot be used with `distinct()`

Query#box(`val`, `Upper`)

Specifies a $box condition

Parameters:

- `val` <Object>
- `Upper` <[Array]> Right Coords

Returns:

- <Query> this

See:

- $box
- within() Query#within
- http://www.mongodb.org/display/DOCS/Geospatial+Indexing

Example

```
var lowerLeft = [40.73083, -73.99756]
var upperRight= [40.741404,  -73.988135]
```

☐ private

```
query.where('loc').within().box(lowerLeft, upperRight)
query.box({ ll : lowerLeft, ur : upperRight })
```

Query#cast(model, [obj])

Casts this query to the schema of `model`

Parameters:

- `model` <Model>
- `[obj]` <Object>

Returns:

- <Object>

Note

If `obj` is present, it is cast instead of this query.

show code

Query#catch([reject])

Executes the query returning a `Promise` which will be resolved with either the doc(s) or rejected with the error. Like `.then()`, but only takes a rejection handler.

Parameters:

- `[reject]` <Function>

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

Returns:

- <Promise>

show code

Query#center()

DEPRECATED Alias for circle

Deprecated. Use circle instead.

Query#centerSphere(`[path]`, `val`)

DEPRECATED Specifies a $centerSphere condition

Parameters:

- `[path]` <String>
- `val` <Object>

Returns:

- <Query> this

See:

- http://www.mongodb.org/display/DOCS/Geospatial+Indexing
- $centerSphere

Deprecated. Use circle instead.

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

Example

```
var area = { center: [50, 50], radius: 10 };
query.where('loc').within().centerSphere(area);
```

show code

Query#circle([path], area)

Specifies a $center or $centerSphere condition.

Parameters:

- [path] <String>
- area <Object>

Returns:

- <Query> this

See:

- $center
- $centerSphere
- $geoWithin
- http://www.mongodb.org/display/DOCS/Geospatial+Indexing

Example

```
var area = { center: [50, 50], radius: 10, unique: true }
query.where('loc').within().circle(area)
// alternatively
query.circle('loc', area);

// spherical calculations
var area = { center: [50, 50], radius: 10, unique: true, sphe
```

☐ private

```
query.where('loc').within().circle(area)
// alternatively
query.circle('loc', area);
```

New in 3.7.0

Query#comment(`val`)

Specifies the `comment` option.

Parameters:

- `val` <Number>

See:

- comment

Example

```
query.comment('login query')
```

Note

Cannot be used with `distinct()`

Query#count(`[criteria]`, `[callback]`)

Specifying this query as a `count` query.

private

mongoose

Parameters:

- `[criteria]` <Object> mongodb selector
- `[callback]` <Function>

Returns:

- <Query> this

See:

- count

Passing a `callback` executes the query.

Example:

```
var countQuery = model.where({ 'color': 'black' }).count();

query.count({ color: 'black' }).count(callback)

query.count({ color: 'black' }, callback)

query.where('color', 'black').count(function (err, count) {
  if (err) return handleError(err);
  console.log('there are %d kittens', count);
})
```

show code

Query#cursor(`[options]`)

Returns a wrapper around a mongodb driver cursor.
A QueryCursor exposes a Streams3-compatible interface, as well as a `.next()` function.

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

Parameters:

- `[options]` <Object>

Returns:

- <QueryCursor>

See:

- QueryCursor

Example

```
// There are 2 ways to use a cursor. First, as a stream:
Thing.
  find({ name: /^hello/ }).
  cursor().
  on('data', function(doc) { console.log(doc); }).
  on('end', function() { console.log('Done!'); });

// Or you can use `.next()` to manually get the next doc in t
// `.next()` returns a promise, so you can use promises or ca
var cursor = Thing.find({ name: /^hello/ }).cursor();
cursor.next(function(error, doc) {
  console.log(doc);
});

// Because `.next()` returns a promise, you can use co
// to easily iterate through all documents without loading th
// all into memory.
co(function*() {
  const cursor = Thing.find({ name: /^hello/ }).cursor();
  for (let doc = yield cursor.next(); doc != null; doc = yie
    console.log(doc);
  }
});
```

Valid options

- `transform`: optional function which accepts a mongoose document. The return value of the function will be emitted on `data`

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js
querystream.js
connection.js
utils.js
browser.js
drivers/node-mongodb-native/collection.js
drivers/node-mongodb-native/connection.js
error/messages.js
error/validation.js
error.js
querycursor.js
virtualtype.js
schema.js
document.js

and returned by `.next()`.

show code

Query#distinct(`[field]`, `[criteria]`, `[callback]`)

Declares or executes a distict() operation.

Parameters:

- `[field]` <String>
- `[criteria]` <Object, Query>
- `[callback]` <Function>

Returns:

- <Query> this

See:

- distinct

Passing a `callback` executes the query.

Example

```
distinct(field, conditions, callback)
distinct(field, conditions)
distinct(field, callback)
distinct(field)
distinct(callback)
distinct()
```

show code

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

Query#elemMatch(`path`, `criteria`)

Specifies an `$elemMatch` condition

Parameters:

- `path` <String, Object, Function>
- `criteria` <Object, Function>

Returns:

- <Query> this

See:

- $elemMatch

Example

```
query.elemMatch('comment', { author: 'autobot', votes: {$gte

query.where('comment').elemMatch({ author: 'autobot', votes:

query.elemMatch('comment', function (elem) {
  elem.where('author').equals('autobot');
  elem.where('votes').gte(5);
})

query.where('comment').elemMatch(function (elem) {
  elem.where({ author: 'autobot' });
  elem.where('votes').gte(5);
})
```

Query#equals(`val`)

☐ private

Specifies the complementary comparison value for paths specified with `where()`

Parameters:

- `val` <Object>

Returns:

- <Query> this

Example

```
User.where('age').equals(49);

// is the same as

User.where('age', 49);
```

Query#exec([operation], [callback])

Executes the query

Parameters:

- `[operation]` <String, Function>
- `[callback]` <Function>

Returns:

- <Promise>

Examples:

```
var promise = query.exec();
var promise = query.exec('update');
```

☐ private

```
query.exec(callback);
query.exec('find', callback);
```

show code

Query#exists([path], val)

Specifies an `$exists` condition

Parameters:

- `[path]` <String>
- `val` <Number>

Returns:

- <Query> this

See:

- $exists

Example

```
// { name: { $exists: true }}
Thing.where('name').exists()
Thing.where('name').exists(true)
Thing.find().exists('name')

// { name: { $exists: false }}
Thing.where('name').exists(false);
Thing.find().exists('name', false);
```

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

Query#find(`[criteria]`, `[callback]`)

Finds documents.

Parameters:

- `[criteria]` <Object> mongodb selector
- `[callback]` <Function>

Returns:

- <Query> this

When no `callback` is passed, the query is not executed. When the query is executed, the result will be an array of documents.

Example

```
query.find({ name: 'Los Pollos Hermanos' }).find(callback)
```

show code

Query#findOne(`[criteria]`, `[projection]`, `[callback]`)

Declares the query a findOne operation. When executed, the first found document is passed to the callback.

Parameters:

- `[criteria]` <Object, Query> mongodb selector
- `[projection]` <Object> optional fields to return
- `[callback]` <Function>

Returns:

- <Query> this

☐ private

See:

- findOne
- Query.select

Passing a `callback` executes the query. The result of the query is a single document.

- *Note:* `conditions` is optional, and if `conditions` is null or undefined, mongoose will send an empty `findOne` command to MongoDB, which will return an arbitrary document. If you're querying by `_id`, use `Model.findById()` instead.

Example

```
var query  = Kitten.where({ color: 'white' });
query.findOne(function (err, kitten) {
  if (err) return handleError(err);
  if (kitten) {
    // doc may be null if no document matched
  }
});
```

show code

Query#findOneAndRemove(`[conditions]`, `[options]`, `[callback]`)

Issues a mongodb findAndModify remove command.

Parameters:

- `[conditions]` <Object>
- `[options]` <Object>
- `[callback]` <Function>

private

mongoose

- <Query> this

See:

- mongodb

Finds a matching document, removes it, passing the found document (if any) to the callback. Executes immediately if `callback` is passed.

Available options

- `sort`: if multiple docs are found by the conditions, sets the sort order to choose which doc to update
- `maxTimeMS`: puts a time limit on the query - requires mongodb >= 2.6.0
- `passRawResult`: if true, passes the raw result from the MongoDB driver as the third callback parameter

Callback Signature

```
function(error, doc, result) {
  // error: any errors that occurred
  // doc: the document before updates are applied if `new: fa
  // result: [raw result from the MongoDB driver](http://mong
}
```

Examples

```
A.where().findOneAndRemove(conditions, options, callback) //
A.where().findOneAndRemove(conditions, options)  // return Qu
A.where().findOneAndRemove(conditions, callback) // executes
A.where().findOneAndRemove(conditions) // returns Query
A.where().findOneAndRemove(callback)   // executes
A.where().findOneAndRemove()           // returns Query
```

☐ private

mongoose

Query#findOneAndUpdate(`[query]`, `[doc]`, `[options]`, `[callback]`)

Issues a mongodb findAndModify update command.

Parameters:

- `[query]` <Object, Query>
- `[doc]` <Object>
- `[options]` <Object>
- `[callback]` <Function>

Returns:

- <Query> this

See:

- mongodb

Finds a matching document, updates it according to the `update` arg, passing any `options`, and returns the found document (if any) to the callback. The query executes immediately if `callback` is passed.

Available options

- `new`: bool - if true, return the modified document rather than the original. defaults to false (changed in 4.0)
- `upsert`: bool - creates the object if it doesn't exist. defaults to false.
- `fields`: {Object|String} - Field selection. Equivalent to `.select(fields).findOneAndUpdate()`
- `sort`: if multiple docs are found by the conditions, sets the sort order to choose which doc to update
- `maxTimeMS`: puts a time limit on the query - requires mongodb >= 2.6.0
- `runValidators`: if true, runs update validators on this command. Update validators validate the update operation against the model's schema.
- `setDefaultsOnInsert`: if this and `upsert` are true, mongoose will apply the defaults specified in the model's schema if a new document is created. This option only works on MongoDB >= 2.4 because it relies on MongoDB's `$setOnInsert` operator.

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

- `passRawResult`: if true, passes the raw result from the MongoDB driver as the third callback parameter
- `context` (string) if set to 'query' and `runValidators` is on, `this` will refer to the query in custom validator functions that update validation runs. Does nothing if `runValidators` is false.

Callback Signature

```
function(error, doc) {
  // error: any errors that occurred
  // doc: the document before updates are applied if `new: fa
}
```

Examples

```
query.findOneAndUpdate(conditions, update, options, callback)
query.findOneAndUpdate(conditions, update, options)  // retu
query.findOneAndUpdate(conditions, update, callback) // exec
query.findOneAndUpdate(conditions, update)           // retu
query.findOneAndUpdate(update, callback)             // retu
query.findOneAndUpdate(update)                       // retu
query.findOneAndUpdate(callback)                     // exec
query.findOneAndUpdate()                             // retu
```

Query#geometry(`object`)

Specifies a `$geometry` condition

Parameters:

- `object` <Object> Must contain a `type` property which is a String and a `coordinates` property which is an Array. See the examples.

☐ private

mongoose

- [home]
- [FAQ]
- [plugins]
- [change log]
- [support]
- [fork]
- [guide]
- [API docs]
- [quick start]
- [contributors]
- [prior releases]

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

Returns:

- <Query> this

See:

- $geometry
- http://docs.mongodb.org/manual/release-notes/2.4/#new-geospatial-indexes-with-geojson-and-improved-spherical-geometry
- http://www.mongodb.org/display/DOCS/Geospatial+Indexing

Example

```
var polyA = [[[ 10, 20 ], [ 10, 40 ], [ 30, 40 ], [ 30, 20 ]]
query.where('loc').within().geometry({ type: 'Polygon', coor

// or
var polyB = [[ 0, 0 ], [ 1, 1 ]]
query.where('loc').within().geometry({ type: 'LineString', c

// or
var polyC = [ 0, 0 ]
query.where('loc').within().geometry({ type: 'Point', coordi

// or
query.where('loc').intersects().geometry({ type: 'Point', co
```

The argument is assigned to the most recent path passed to `where()`.

NOTE:

`geometry()` **must** come after either `intersects()` or `within()`.

The `object` argument must contain `type` and `coordinates` properties.
- type {String}
- coordinates {Array}

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

Query#getQuery()

Returns the current query conditions as a JSON object.

Returns:

- <Object> current query conditions

Example:

```
var query = new Query();
query.find({ a: 1 }).where('b').gt(2);
query.getQuery(); // { a: 1, b: { $gt: 2 } }
```

show code

Query#getUpdate()

Returns the current update operations as a JSON object.

Returns:

- <Object> current update operations

Example:

```
var query = new Query();
query.update({}, { $set: { a: 5 } });
query.getUpdate(); // { $set: { a: 5 } }
```

show code

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

Query#gt(`[path]`, `val`)

Specifies a $gt query condition.

Parameters:

- `[path]` <String>
- `val` <Number>

See:

- $gt

When called with one argument, the most recent path passed to `where()` is used.

Example

```
Thing.find().where('age').gt(21)

// or
Thing.find().gt('age', 21)
```

Query#gte(`[path]`, `val`)

Specifies a $gte query condition.

Parameters:

- `[path]` <String>
- `val` <Number>

See:

- $gte

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

When called with one argument, the most recent path passed to `where()` is used.

Query#hint(`val`)

Sets query hints.

Parameters:

- `val` <Object> a hint object

Returns:

- <Query> this

See:

- $hint

Example

```
query.hint({ indexA: 1, indexB: -1})
```

Note

Cannot be used with `distinct()`

Query#in(`[path]`, `val`)

Specifies an $in query condition.

private

Parameters:

- `[path]` <String>
- `val` <Number>

See:

- $in

When called with one argument, the most recent path passed to `where()` is used.

Query#intersects(`[arg]`)

Declares an intersects query for `geometry()`.

Parameters:

- `[arg]` <Object>

Returns:

- <Query> this

See:

- $geometry
- geoIntersects

Example

```
query.where('path').intersects().geometry({
    type: 'LineString'
  , coordinates: [[180.0, 11.0], [180, 9.0]]
})

query.where('path').intersects({
    type: 'LineString'
```

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

```
  , coordinates: [[180.0, 11.0], [180, 9.0]]
})
```

NOTE:

MUST be used after `where()`.

NOTE:

In Mongoose 3.7, `intersects` changed from a getter to a function. If you need the old syntax, use this.

Query#lean(`bool`)

Sets the lean option.

Parameters:

- `bool` <Boolean> defaults to true

Returns:

- <Query> this

Documents returned from queries with the `lean` option enabled are plain javascript objects, not MongooseDocuments. They have no `save` method, getters/setters or other Mongoose magic applied.

Example:

```
new Query().lean() // true
new Query().lean(true)
new Query().lean(false)

Model.find().lean().exec(function (err, docs) {
  docs[0] instanceof mongoose.Document // false
});
```

☐ private

This is a great option in high-performance read-only scenarios, especially when combined with stream.

show code

Query#limit(`val`)

Specifies the maximum number of documents the query will return.

Parameters:

- `val` <Number>

Example

```
query.limit(20)
```

Note

Cannot be used with `distinct()`

Query#lt(`[path]`, `val`)

Specifies a $lt query condition.

Parameters:

- `[path]` <String>
- `val` <Number>

☐ private

See:

- $lt

When called with one argument, the most recent path passed to `where()` is used.

Query#lte(`[path]`, `val`)

Specifies a $lte query condition.

Parameters:

- `[path]` <String>
- `val` <Number>

See:

- $lte

When called with one argument, the most recent path passed to `where()` is used.

Query#maxDistance(`[path]`, `val`)

Specifies a $maxDistance query condition.

Parameters:

- `[path]` <String>
- `val` <Number>

☐ private

See:

- $maxDistance

When called with one argument, the most recent path passed to `where()` is used.

Query#maxscan()

DEPRECATED Alias of `maxScan`

See:

- maxScan

Query#maxScan(`val`)

Specifies the maxScan option.

Parameters:

- `val` <Number>

See:

- maxScan

Example

```
query.maxScan(100)
```

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

Note

Cannot be used with `distinct()`

Query#merge(`source`)

Merges another Query or conditions object into this one.

Parameters:

- `source` <Query, Object>

Returns:

- <Query> this

When a Query is passed, conditions, field selection and options are merged.

New in 3.7.0

Query#merge(`source`)

Merges another Query or conditions object into this one.

Parameters:

- `source` <Query, Object>

Returns:

- <Query> this

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

When a Query is passed, conditions, field selection and options are merged.

show code

Query#mod(`[path]`, `val`)

Specifies a `$mod` condition

Parameters:

- `[path]` <String>
- `val` <Number>

Returns:

- <Query> this

See:

- $mod

Query#ne(`[path]`, `val`)

Specifies a $ne query condition.

Parameters:

- `[path]` <String>
- `val` <Number>

See:

☐ private

- $ne

When called with one argument, the most recent path passed to `where()` is used.

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

Query#near(`[path]`, `val`)

Specifies a `$near` or `$nearSphere` condition

Parameters:

- `[path]` <String>
- `val` <Object>

Returns:

- <Query> this

See:

- $near
- $nearSphere
- $maxDistance
- http://www.mongodb.org/display/DOCS/Geospatial+Indexing

These operators return documents sorted by distance.

Example

```
query.where('loc').near({ center: [10, 10] });
query.where('loc').near({ center: [10, 10], maxDistance: 5 })
query.where('loc').near({ center: [10, 10], maxDistance: 5, s
query.near('loc', { center: [10, 10], maxDistance: 5 });
```

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

Query#nearSphere()

DEPRECATED Specifies a `$nearSphere` condition

See:

- near()
- $near
- $nearSphere
- $maxDistance

Example

```
query.where('loc').nearSphere({ center: [10, 10], maxDistance
```

Deprecated. Use `query.near()` instead with the `spherical` option set to `true`.

Example

```
query.where('loc').near({ center: [10, 10], spherical: true
```

show code

Query#nin([path], val)

Specifies an $nin query condition.

Parameters:

- `[path]` <String>
- `val` <Number>

private

See:

- $nin

When called with one argument, the most recent path passed to `where()` is used.

Query#nor(`array`)

Specifies arguments for a `$nor` condition.

Parameters:

- `array` <Array> array of conditions

Returns:

- <Query> this

See:

- $nor

Example

```
query.nor([{ color: 'green' }, { status: 'ok' }])
```

Query#or(`array`)

Specifies arguments for an `$or` condition.

private

Parameters:

- `array` <Array> array of conditions

Returns:

- <Query> this

See:

- $or

Example

```
query.or([{ color: 'red' }, { status: 'emergency' }])
```

Query#polygon(`[path]`, `[coordinatePairs...]`)

Specifies a $polygon condition

Parameters:

- `[path]` <String, Array>
- `[coordinatePairs...]` <Array, Object>

Returns:

- <Query> this

See:

- $polygon
- http://www.mongodb.org/display/DOCS/Geospatial+Indexing

Example

☐ private

mongoose

- [home]
- [FAQ]
- [plugins]
- [change log]
- [support]
- [fork]
- [guide]
- [API docs]
- [quick start]
- [contributors]
- [prior releases]

```
query.where('loc').within().polygon([10,20], [13, 25], [7,15]
query.polygon('loc', [10,20], [13, 25], [7,15])
```

Query#populate(`path`, `[select]`, `[model]`, `[match]`, `[options]`)

Specifies paths which should be populated with other documents.

Parameters:

- `path` <Object, String> either the path to populate or an object specifying all parameters
- `[select]` <Object, String> Field selection for the population query
- `[model]` <Model> The model you wish to use for population. If not specified, populate will look up the model by the name in the Schema's `ref` field.
- `[match]` <Object> Conditions for the population query
- `[options]` <Object> Options for the population query (sort, etc)

Returns:

- <Query> this

See:

- population
- Query#select
- Model.populate

Example:

```
Kitten.findOne().populate('owner').exec(function (err, kitter
  console.log(kitten.owner.name) // Max
})

Kitten.find().populate({
```

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

```
    path: 'owner'
  , select: 'name'
  , match: { color: 'black' }
  , options: { sort: { name: -1 }}
}).exec(function (err, kittens) {
  console.log(kittens[0].owner.name) // Zoopa
})

// alternatively
Kitten.find().populate('owner', 'name', null, {sort: { name:
  console.log(kittens[0].owner.name) // Zoopa
})
```

Paths are populated after the query executes and a response is received. A separate query is then executed for each path specified for population. After a response for each query has also been returned, the results are passed to the callback.

show code

Query#read(`pref`, `[tags]`)

Determines the MongoDB nodes from which to read.

Parameters:

- `pref` <String> one of the listed preference options or aliases
- `[tags]` <Array> optional tags for this query

Returns:

- <Query> this

See:

- mongodb
- driver

☐ private

mongoose

Preferences:

```
primary - (default) Read from primary only. Operations will 
secondary            Read from secondary if available, otherw
primaryPreferred     Read from primary if available, otherwi
secondaryPreferred   Read from a secondary if available, othe
nearest              All operations read from among the neare
```

Aliases

```
p   primary
pp  primaryPreferred
s   secondary
sp  secondaryPreferred
n   nearest
```

Example:

```
new Query().read('primary')
new Query().read('p')  // same as primary

new Query().read('primaryPreferred')
new Query().read('pp') // same as primaryPreferred

new Query().read('secondary')
new Query().read('s')  // same as secondary

new Query().read('secondaryPreferred')
new Query().read('sp') // same as secondaryPreferred

new Query().read('nearest')
new Query().read('n')  // same as nearest

// read from secondaries with matching tags
new Query().read('s', [{ dc:'sf', s: 1 },{ dc:'ma', s: 2 }])
```

Read more about how to use read preferrences here and here.

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

Query#regex(`[path]`, `val`)

Specifies a $regex query condition.

Parameters:

- `[path]` <String>
- `val` <Number>

See:

- $regex

When called with one argument, the most recent path passed to `where()` is used.

Query#remove(`[criteria]`, `[callback]`)

Declare and/or execute this query as a remove() operation.

Parameters:

- `[criteria]` <Object, Query> mongodb selector
- `[callback]` <Function>

Returns:

- <Query> this

See:

- remove

Example

private

```
Model.remove({ artist: 'Anne Murray' }, callback)
```

Note

The operation is only executed when a callback is passed. To force execution without a callback, you must first call `remove()` and then execute it by using the `exec()` method.

```
// not executed
var query = Model.find().remove({ name: 'Anne Murray' })

// executed
query.remove({ name: 'Anne Murray' }, callback)
query.remove({ name: 'Anne Murray' }).remove(callback)

// executed without a callback
query.exec()

// summary
query.remove(conds, fn); // executes
query.remove(conds)
query.remove(fn) // executes
query.remove()
```

show code

Query#select(`arg`)

Specifies which document fields to include or exclude (also known as the query "projection")

Parameters:

- `arg` <Object, String>

private

Returns:

- <Query> this

See:

- SchemaType

When using string syntax, prefixing a path with `-` will flag that path as excluded. When a path does not have the `-` prefix, it is included. Lastly, if a path is prefixed with `+`, it forces inclusion of the path, which is useful for paths excluded at the schema level.

Example

```
// include a and b, exclude other fields
query.select('a b');

// exclude c and d, include other fields
query.select('-c -d');

// or you may use object notation, useful when
// you have keys already prefixed with a "-"
query.select({ a: 1, b: 1 });
query.select({ c: 0, d: 0 });

// force inclusion of field excluded at schema level
query.select('+path')
```

NOTE:

Cannot be used with `distinct()`.

v2 had slightly different syntax such as allowing arrays of field names. This support was removed in v3.

Query#selected()

private

Determines if field selection has been made.

Returns:

- <Boolean>

Query#selectedExclusively()

Determines if exclusive field selection has been made.

Returns:

- <Boolean>

```
query.selectedExclusively() // false
query.select('-name')
query.selectedExclusively() // true
query.selectedInclusively() // false
```

Query#selectedInclusively()

Determines if inclusive field selection has been made.

Returns:

- <Boolean>

```
query.selectedInclusively() // false
query.select('name')
query.selectedInclusively() // true
```

private

Query#setOptions(`options`)

Sets query options.

Parameters:

- `options` <Object>

Options:

- tailable *
- sort *
- limit *
- skip *
- maxscan *
- batchSize *
- comment *
- snapshot *
- hint *
- readPreference **
- lean *
- safe

** denotes a query helper method is also available*
*** query helper method to set `readPreference` is `read()`*

show code

Query#size(`[path]`, `val`)

private

Specifies a $size query condition.

Parameters:

- `[path]` <String>
- `val` <Number>

See:

- $size

When called with one argument, the most recent path passed to `where()` is used.

Example

```
MyModel.where('tags').size(0).exec(function (err, docs) {
  if (err) return handleError(err);

  assert(Array.isArray(docs));
  console.log('documents with 0 tags', docs);
})
```

Query#skip(`val`)

Specifies the number of documents to skip.

Parameters:

- `val` <Number>

See:

- cursor.skip

Example

```
query.skip(100).limit(20)
```

private

mongoose

Note

Cannot be used with `distinct()`

Query#slaveOk(v)

DEPRECATED Sets the slaveOk option.

Parameters:

- v <Boolean> defaults to true

Returns:

- <Query> this

See:

- mongodb
- slaveOk
- read()

Deprecated in MongoDB 2.2 in favor of read preferences.

Example:

```
query.slaveOk() // true
query.slaveOk(true)
query.slaveOk(false)
```

☐ private

Query#slice(`[path]`, `val`)

Specifies a $slice projection for an array.

Parameters:

- `[path]` <String>
- `val` <Number> number/range of elements to slice

Returns:

- <Query> this

See:

- mongodb
- $slice

Example

```
query.slice('comments', 5)
query.slice('comments', -5)
query.slice('comments', [10, 5])
query.where('comments').slice(5)
query.where('comments').slice([-10, 5])
```

Query#snapshot()

Specifies this query as a `snapshot` query.

Returns:

- <Query> this

See:

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

- snapshot

Example

```
query.snapshot() // true
query.snapshot(true)
query.snapshot(false)
```

Note

Cannot be used with `distinct()`

Query#sort(`arg`)

Sets the sort order

Parameters:

- `arg` <Object, String>

Returns:

- <Query> this

See:

- cursor.sort

If an object is passed, values allowed are `asc`, `desc`, `ascending`, `descending`, `1`, and `-1`.

If a string is passed, it must be a space delimited list of path names. The sort order of each path is ascending unless the path name is prefixed with -
which will be treated as descending.

Example

☐ private

```
// sort by "field" ascending and "test" descending
query.sort({ field: 'asc', test: -1 });

// equivalent
query.sort('field -test');
```

Note

Cannot be used with `distinct()`

show code

Query#stream(`[options]`)

Returns a Node.js 0.8 style read stream interface.

Parameters:

- `[options]` <Object>

Returns:

- <QueryStream>

See:

- QueryStream

Example

```
// follows the nodejs 0.8 stream api
Thing.find({ name: /^hello/ }).stream().pipe(res)

// manual streaming
var stream = Thing.find({ name: /^hello/ }).stream();
```

☐ private

```
stream.on('data', function (doc) {
  // do something with the mongoose document
}).on('error', function (err) {
  // handle the error
}).on('close', function () {
  // the stream is closed
});
```

Valid options

- `transform`: optional function which accepts a mongoose document. The return value of the function will be emitted on `data`.

Example

```
// JSON.stringify all documents before emitting
var stream = Thing.find().stream({ transform: JSON.stringify
stream.pipe(writeStream);
```

show code

Query#tailable(`bool`, `[opts]`, `[opts.numberOfRetries]`, `[opts.tailableRetryInterval]`)

Sets the tailable option (for use with capped collections).

Parameters:

- `bool` <Boolean> defaults to true
- `[opts]` <Object> options to set
- `[opts.numberOfRetries]` <Number> if cursor is exhausted, retry this many times before giving up
- `[opts.tailableRetryInterval]` <Number> if cursor is exhausted, wait this many milliseconds before retrying

private

See:

- tailable

Example

```
query.tailable() // true
query.tailable(true)
query.tailable(false)
```

Note

Cannot be used with `distinct()`

show code

Query#then(`[resolve]`, `[reject]`)

Executes the query returning a `Promise` which will be resolved with either the doc(s) or rejected with the error.

Parameters:

- `[resolve]` <Function>
- `[reject]` <Function>

Returns:

- <Promise>

show code

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

Query#toConstructor()

Converts this query to a customized, reusable query constructor with all arguments and options retained.

Returns:

- <Query> subclass-of-Query

Example

```
// Create a query for adventure movies and read from the prin
// node in the replica-set unless it is down, in which case w
// read from a secondary node.
var query = Movie.find({ tags: 'adventure' }).read('primaryPr

// create a custom Query constructor based off these settings
var Adventure = query.toConstructor();

// Adventure is now a subclass of mongoose.Query and works th
// default query parameters and options set.
Adventure().exec(callback)

// further narrow down our query results while still using th
Adventure().where({ name: /^Life/ }).exec(callback);

// since Adventure is a stand-alone constructor we can also a
// helper methods and getters without impacting global querie
Adventure.prototype.startsWith = function (prefix) {
  this.where({ name: new RegExp('^' + prefix) })
  return this;
}
Object.defineProperty(Adventure.prototype, 'highlyRated', {
  get: function () {
    this.where({ rating: { $gt: 4.5 }});
    return this;
  }
})
Adventure().highlyRated.startsWith('Life').exec(callback)
```

☐ private

mongoose

New in 3.7.3

show code

Query#update(`[criteria]`, `[doc]`, `[options]`, `[callback]`)

Declare and/or execute this query as an update() operation.

Parameters:

- `[criteria]` <Object>
- `[doc]` <Object> the update command
- `[options]` <Object>
- `[callback]` <Function>

Returns:

- <Query> this

See:

- Model.update
- update

All paths passed that are not $atomic operations will become $set ops.

Example

```
Model.where({ _id: id }).update({ title: 'words' })

// becomes

Model.where({ _id: id }).update({ $set: { title: 'words' }})
```

Valid options:

private
- `safe` (boolean) safe mode (defaults to value set in schema (true))

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

- `upsert` (boolean) whether to create the doc if it doesn't match (false)
- `multi` (boolean) whether multiple documents should be updated (false)
- `runValidators`: if true, runs update validators on this command. Update validators validate the update operation against the model's schema.
- `setDefaultsOnInsert`: if this and `upsert` are true, mongoose will apply the defaults specified in the model's schema if a new document is created. This option only works on MongoDB >= 2.4 because it relies on MongoDB's `$setOnInsert` operator.
- `strict` (boolean) overrides the `strict` option for this update
- `overwrite` (boolean) disables update-only mode, allowing you to overwrite the doc (false)
- `context` (string) if set to 'query' and `runValidators` is on, `this` will refer to the query in custom validator functions that update validation runs. Does nothing if `runValidators` is false.

Note

Passing an empty object `{}` as the doc will result in a no-op unless the `overwrite` option is passed. Without the `overwrite` option set, the update operation will be ignored and the callback executed without sending the command to MongoDB so as to prevent accidently overwritting documents in the collection.

Note

The operation is only executed when a callback is passed. To force execution without a callback, we must first call update() and then execute it by using the `exec()` method.

```
var q = Model.where({ _id: id });
q.update({ $set: { name: 'bob' }}).update(); // not executed

q.update({ $set: { name: 'bob' }}).exec(); // executed

// keys that are not $atomic ops become $set.
// this executes the same command as the previous example.
q.update({ name: 'bob' }).exec();

// overwriting with empty docs
var q = Model.where({ _id: id }).setOptions({ overwrite: true
```

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

```
q.update({ }, callback); // executes

// multi update with overwrite to empty doc
var q = Model.where({ _id: id });
q.setOptions({ multi: true, overwrite: true })
q.update({ });
q.update(callback); // executed

// multi updates
Model.where()
     .update({ name: /^match/ }, { $set: { arr: [] }}, { mult

// more multi updates
Model.where()
     .setOptions({ multi: true })
     .update({ $set: { arr: [] }}, callback)

// single update by default
Model.where({ email: 'address@example.com' })
     .update({ $inc: { counter: 1 }}, callback)
```

API summary

```
update(criteria, doc, options, cb) // executes
update(criteria, doc, options)
update(criteria, doc, cb) // executes
update(criteria, doc)
update(doc, cb) // executes
update(doc)
update(cb) // executes
update(true) // executes
update()
```

show code

private Query#where([path], [val])

Specifies a `path` for use with chaining.

Parameters:

- `[path]` <String, Object>
- `[val]` <T>

Returns:

- <Query> this

Example

```
// instead of writing:
User.find({age: {$gte: 21, $lte: 65}}, callback);

// we can instead write:
User.where('age').gte(21).lte(65);

// passing query conditions is permitted
User.find().where({ name: 'vonderful' })

// chaining
User
.where('age').gte(21).lte(65)
.where('name', /^vonderful/i)
.where('friends').slice(10)
.exec(callback)
```

Query#within()

Defines a `$within` or `$geoWithin` argument for geo-spatial queries.

Returns:

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js
querystream.js
connection.js
utils.js
browser.js
drivers/node-mongodb-native/collection.js
drivers/node-mongodb-native/connection.js
error/messages.js
error/validation.js
error.js
querycursor.js
virtualtype.js
schema.js
document.js
types/subdocument.js

- <Query> this

See:

- $polygon
- $box
- $geometry
- $center
- $centerSphere

Example

```
query.where(path).within().box()
query.where(path).within().circle()
query.where(path).within().geometry()

query.where('loc').within({ center: [50,50], radius: 10, uni
query.where('loc').within({ box: [[40.73, -73.9], [40.7, -73
query.where('loc').within({ polygon: [[],[],[],[]] });

query.where('loc').within([], [], []) // polygon
query.where('loc').within([], []) // box
query.where('loc').within({ type: 'LineString', coordinates:
```

MUST be used after `where()`.

NOTE:

As of Mongoose 3.7, `$geoWithin` is always used for queries. To change this behavior, see Query.use$geoWithin.

NOTE:

In Mongoose 3.7, `within` changed from a getter to a function. If you need the old syntax, use this.

private Query#use$geoWithin

mongoose

Flag to opt out of using `$geoWithin`.

```
mongoose.Query.use$geoWithin = false;
```

MongoDB 2.4 deprecated the use of `$within`, replacing it with `$geoWithin`. Mongoose uses `$geoWithin` by default (which is 100% backward compatible with $within). If you are running an older version of MongoDB, set this flag to `false` so your `within()` queries continue to work.

show code

See:

- http://docs.mongodb.org/manual/reference/operator/geoWithin/

schema/array.js

SchemaArray#checkRequired(`value`)

Check if the given value satisfies a required validator. The given value must be not null nor undefined, and have a non-zero length.

Parameters:

- `value` <Any>

Returns:

- <Boolean>

show code

☐ private

mongoose

SchemaArray(`key`, `cast`, `options`)

Array SchemaType constructor

Parameters:

- `key` <String>
- `cast` <SchemaType>
- `options` <Object>

Inherits:

- SchemaType

show code

SchemaArray.schemaName

This schema type's name, to defend against minifiers that mangle function names.

show code

schema/string.js

SchemaString#checkRequired(`value`, `doc`)

Check if the given value satisfies a required validator.

☐ private

Parameters:

- `value` <Any>
- `doc` <Document>

Returns:

- <Boolean>

show code

SchemaString#enum(`[args...]`)

Adds an enum validator

Parameters:

- `[args...]` <String, Object> enumeration values

Returns:

- <SchemaType> this

See:

- Customized Error Messages

Example:

```
var states = ['opening', 'open', 'closing', 'closed']
var s = new Schema({ state: { type: String, enum: states }})
var M = db.model('M', s)
var m = new M({ state: 'invalid' })
m.save(function (err) {
  console.error(String(err)) // ValidationError: `invalid` i
  m.state = 'open'
  m.save(callback) // success
})
```

☐ private

```
// or with custom error messages
var enum = {
  values: ['opening', 'open', 'closing', 'closed'],
  message: 'enum validator failed for path `{PATH}` with valu
}
var s = new Schema({ state: { type: String, enum: enum })
var M = db.model('M', s)
var m = new M({ state: 'invalid' })
m.save(function (err) {
  console.error(String(err)) // ValidationError: enum validat
  m.state = 'open'
  m.save(callback) // success
})
```

show code

SchemaString#lowercase()

Adds a lowercase setter.

Returns:

- <SchemaType> this

Example:

```
var s = new Schema({ email: { type: String, lowercase: true }
var M = db.model('M', s);
var m = new M({ email: 'SomeEmail@example.COM' });
console.log(m.email) // someemail@example.com
```

show code

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

SchemaString#match(`regExp`, `[message]`)

Sets a regexp validator.

Parameters:

- `regExp` <RegExp> regular expression to test against
- `[message]` <String> optional custom error message

Returns:

- <SchemaType> this

See:

- Customized Error Messages

Any value that does not pass `regExp`.test(val) will fail validation.

Example:

```
var s = new Schema({ name: { type: String, match: /^a/ }})
var M = db.model('M', s)
var m = new M({ name: 'I am invalid' })
m.validate(function (err) {
  console.error(String(err)) // "ValidationError: Path `name`
  m.name = 'apples'
  m.validate(function (err) {
    assert.ok(err) // success
  })
})

// using a custom error message
var match = [ /\.html$/, "That file doesn't end in .html ({VA
var s = new Schema({ file: { type: String, match: match }})
var M = db.model('M', s);
var m = new M({ file: 'invalid' });
m.validate(function (err) {
  console.log(String(err)) // "ValidationError: That file doe
})
```

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

Empty strings, `undefined`, and `null` values always pass the match validator. If you require these values, enable the `required` validator also.

```
var s = new Schema({ name: { type: String, match: /^a/, requ
```

show code

SchemaString#maxlength(`value`, `[message]`)

Sets a maximum length validator.

Parameters:

- `value` <Number> maximum string length
- `[message]` <String> optional custom error message

Returns:

- <SchemaType> this

See:

- Customized Error Messages

Example:

```
var schema = new Schema({ postalCode: { type: String, maxlen
var Address = db.model('Address', schema)
var address = new Address({ postalCode: '9512512345' })
address.save(function (err) {
  console.error(err) // validator error
  address.postalCode = '95125';
  address.save() // success
```

private

mongoose

- [home]
- [FAQ]
- [plugins]
- [change log]
- [support]
- [fork]
- [guide]
- [API docs]
- [quick start]
- [contributors]
- [prior releases]

index.js
querystream.js
connection.js
utils.js
browser.js
drivers/node-mongodb-native/collection.js
drivers/node-mongodb-native/connection.js
error/messages.js
error/validation.js
error.js
querycursor.js
virtualtype.js
schema.js
document.js
types/subdocument.js

```
})

// custom error messages
// We can also use the special {MAXLENGTH} token which will l
var maxlength = [9, 'The value of path `{PATH}` (`{VALUE}`) e
var schema = new Schema({ postalCode: { type: String, maxleng
var Address = mongoose.model('Address', schema);
var address = new Address({ postalCode: '9512512345' });
address.validate(function (err) {
  console.log(String(err)) // ValidationError: The value of p
})
```

show code

SchemaString#minlength(`value`, `[message]`)

Sets a minimum length validator.

Parameters:

- `value` <Number> minimum string length
- `[message]` <String> optional custom error message

Returns:

- <SchemaType> this

See:

- Customized Error Messages

Example:

```
var schema = new Schema({ postalCode: { type: String, minleng
var Address = db.model('Address', schema)
var address = new Address({ postalCode: '9512' })
```

private

mongoose

- [home]
- [FAQ]
- [plugins]
- [change log]
- [support]
- [fork]
- [guide]
- [API docs]
- [quick start]
- [contributors]
- [prior releases]

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

```
address.save(function (err) {
  console.error(err) // validator error
  address.postalCode = '95125';
  address.save() // success
})

// custom error messages
// We can also use the special {MINLENGTH} token which will t
var minlength = [5, 'The value of path `{PATH}` (`{VALUE}`) :
var schema = new Schema({ postalCode: { type: String, minlen
var Address = mongoose.model('Address', schema);
var address = new Address({ postalCode: '9512' });
address.validate(function (err) {
  console.log(String(err)) // ValidationError: The value of p
})
```

show code

SchemaString(key, options)

String SchemaType constructor.

Parameters:

- key <String>
- options <Object>

Inherits:

- SchemaType

show code

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

SchemaString#trim()

Adds a trim setter.

Returns:

- <SchemaType> this

The string value will be trimmed when set.

Example:

```
var s = new Schema({ name: { type: String, trim: true }})
var M = db.model('M', s)
var string = ' some name '
console.log(string.length) // 11
var m = new M({ name: string })
console.log(m.name.length) // 9
```

show code

SchemaString#uppercase()

Adds an uppercase setter.

Returns:

- <SchemaType> this

Example:

```
var s = new Schema({ caps: { type: String, uppercase: true }
var M = db.model('M', s);
var m = new M({ caps: 'an example' });
console.log(m.caps) // AN EXAMPLE
```

private

show code

SchemaString.schemaName

This schema type's name, to defend against minifiers that mangle function names.

show code

schema/documentarray.js

DocumentArray(`key`, `schema`, `options`)

SubdocsArray SchemaType constructor

Parameters:

- `key` <String>
- `schema` <Schema>
- `options` <Object>

Inherits:

- SchemaArray

show code

☐ private

mongoose

DocumentArray.schemaName

This schema type's name, to defend against minifiers that mangle function names.

show code

schema/number.js

SchemaNumber#checkRequired(`value`, `doc`)

Check if the given value satisfies a required validator.

Parameters:

- `value` <Any>
- `doc` <Document>

Returns:

- <Boolean>

show code

SchemaNumber#max(`maximum`, `[message]`)

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

Sets a maximum number validator.

Parameters:

- `maximum` <Number> number
- `[message]` <String> optional custom error message

Returns:

- <SchemaType> this

See:

- Customized Error Messages

Example:

```
var s = new Schema({ n: { type: Number, max: 10 })
var M = db.model('M', s)
var m = new M({ n: 11 })
m.save(function (err) {
  console.error(err) // validator error
  m.n = 10;
  m.save() // success
})

// custom error messages
// We can also use the special {MAX} token which will be rep
var max = [10, 'The value of path `{PATH}` ({VALUE}) exceeds
var schema = new Schema({ n: { type: Number, max: max })
var M = mongoose.model('Measurement', schema);
var s= new M({ n: 4 });
s.validate(function (err) {
  console.log(String(err)) // ValidationError: The value of
})
```

show code

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

SchemaNumber#min(value, [message])

Sets a minimum number validator.

Parameters:

- value <Number> minimum number
- [message] <String> optional custom error message

Returns:

- <SchemaType> this

See:

- Customized Error Messages

Example:

```
var s = new Schema({ n: { type: Number, min: 10 })
var M = db.model('M', s)
var m = new M({ n: 9 })
m.save(function (err) {
  console.error(err) // validator error
  m.n = 10;
  m.save() // success
})

// custom error messages
// We can also use the special {MIN} token which will be rep
var min = [10, 'The value of path `{PATH}` ({VALUE}) is bene
var schema = new Schema({ n: { type: Number, min: min })
var M = mongoose.model('Measurement', schema);
var s= new M({ n: 4 });
s.validate(function (err) {
  console.log(String(err)) // ValidationError: The value of 
})
```

show code

☐ private

SchemaNumber(`key`, `options`)

Number SchemaType constructor.

Parameters:

- `key` <String>
- `options` <Object>

Inherits:

- SchemaType

show code

SchemaNumber.schemaName

This schema type's name, to defend against minifiers that mangle function names.

show code

schema/date.js

SchemaDate#checkRequired(`value`, `doc`)

☐ private

Check if the given value satisfies a required validator. To satisfy a required validator, the given value must be an instance of `Date`.

Parameters:

- `value` <Any>
- `doc` <Document>

Returns:

- <Boolean>

show code

SchemaDate#expires(`when`)

Declares a TTL index (rounded to the nearest second) for *Date* types only.

Parameters:

- `when` <Number, String>

Returns:

- <SchemaType> this

This sets the `expireAfterSeconds` index option available in MongoDB >= 2.1.2.
This index type is only compatible with Date types.

Example:

```
// expire in 24 hours
new Schema({ createdAt: { type: Date, expires: 60*60*24 }});
```

☐ private

`expires` utilizes the `ms` module from guille allowing us to use a friendlier syntax:

Example:

```
// expire in 24 hours
new Schema({ createdAt: { type: Date, expires: '24h' }});

// expire in 1.5 hours
new Schema({ createdAt: { type: Date, expires: '1.5h' }});

// expire in 7 days
var schema = new Schema({ createdAt: Date });
schema.path('createdAt').expires('7d');
```

show code

SchemaDate#max(`maximum`, `[message]`)

Sets a maximum date validator.

Parameters:

- `maximum` <Date> date
- `[message]` <String> optional custom error message

Returns:

- <SchemaType> this

See:

- Customized Error Messages

Example:

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

```
var s = new Schema({ d: { type: Date, max: Date('2014-01-01')
var M = db.model('M', s)
var m = new M({ d: Date('2014-12-08') })
m.save(function (err) {
  console.error(err) // validator error
  m.d = Date('2013-12-31');
  m.save() // success
})

// custom error messages
// We can also use the special {MAX} token which will be rep
var max = [Date('2014-01-01'), 'The value of path `{PATH}` (
var schema = new Schema({ d: { type: Date, max: max })
var M = mongoose.model('M', schema);
var s= new M({ d: Date('2014-12-08') });
s.validate(function (err) {
  console.log(String(err)) // ValidationError: The value of p
})
```

show code

SchemaDate#min(`value`, `[message]`)

Sets a minimum date validator.

Parameters:

- `value` <Date> minimum date
- `[message]` <String> optional custom error message

Returns:

- <SchemaType> this

See:

☐ private

- Customized Error Messages

Example:

```
var s = new Schema({ d: { type: Date, min: Date('1970-01-01')
var M = db.model('M', s)
var m = new M({ d: Date('1969-12-31') })
m.save(function (err) {
  console.error(err) // validator error
  m.d = Date('2014-12-08');
  m.save() // success
})

// custom error messages
// We can also use the special {MIN} token which will be rep
var min = [Date('1970-01-01'), 'The value of path `{PATH}` (
var schema = new Schema({ d: { type: Date, min: min })
var M = mongoose.model('M', schema);
var s= new M({ d: Date('1969-12-31') });
s.validate(function (err) {
  console.log(String(err)) // ValidationError: The value of p
})
```

show code

SchemaDate(key, options)

Date SchemaType constructor.

Parameters:

- key <String>
- options <Object>

Inherits:

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

- SchemaType

show code

SchemaDate.schemaName

This schema type's name, to defend against minifiers that mangle function names.

show code

schema/buffer.js

SchemaBuffer#checkRequired(`value`, `doc`)

Check if the given value satisfies a required validator. To satisfy a required validator, a buffer must not be null or undefined and have non-zero length.

Parameters:

- `value` <Any>
- `doc` <Document>

Returns:

- <Boolean>

show code

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

SchemaBuffer(`key`, `options`)

Buffer SchemaType constructor

Parameters:

- `key` <String>
- `options` <Object>

Inherits:

- SchemaType

show code

SchemaBuffer.schemaName

This schema type's name, to defend against minifiers that mangle function names.

show code

schema/boolean.js

SchemaBoolean#checkRequired(`value`)

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

Check if the given value satisfies a required validator. For a boolean to satisfy a required validator, it must be strictly equal to true or to false.

Parameters:

- `value` <Any>

Returns:

- <Boolean>

show code

SchemaBoolean(`path`, `options`)

Boolean SchemaType constructor.

Parameters:

- `path` <String>
- `options` <Object>

Inherits:

- SchemaType

show code

SchemaBoolean.schemaName

This schema type's name, to defend against minifiers that mangle function names.

private

mongoose

show code

schema/objectid.js

ObjectId#auto(`turnOn`)

Adds an auto-generated ObjectId default if turnOn is true.

Parameters:

- `turnOn` <Boolean> auto generated ObjectId defaults

Returns:

- <SchemaType> this

show code

ObjectId#checkRequired(`value`, `doc`)

Check if the given value satisfies a required validator.

Parameters:

- `value` <Any>
- `doc` <Document>

Returns:

- <Boolean>

☐ private

show code

ObjectId(`key`, `options`)

ObjectId SchemaType constructor.

Parameters:

- `key` <String>
- `options` <Object>

Inherits:

- SchemaType

show code

ObjectId.schemaName

This schema type's name, to defend against minifiers that mangle function names.

show code

schema/mixed.js

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

Mixed(`path`, `options`**)**

Mixed SchemaType constructor.

Parameters:

- `path` <String>
- `options` <Object>

Inherits:

- SchemaType

show code

Mixed.schemaName

This schema type's name, to defend against minifiers that mangle function names.

show code

schema/embedded.js

Embedded(`schema`, `key`, `options`**)**

Sub-schema schematype constructor

☐ private **Parameters:**

mongoose

- `schema` <Schema>
- `key` <String>
- `options` <Object>

Inherits:

- SchemaType

show code

aggregate.js

Aggregate#addCursorFlag(`flag`, `value`)

Adds a cursor flag

Parameters:

- `flag` <String>
- `value` <Boolean>

See:

- mongodb

Example:

```
var cursor = Model.aggregate(..).cursor({ batchSize: 1000 })
cursor.each(function(error, doc) {
  // use doc
});
```

show code

☐ private

Aggregate([ops])

Aggregate constructor used for building aggregation pipelines.

Parameters:

- `[ops]` <Object, Array> aggregation operator(s) or operator array

See:

- MongoDB
- driver

Example:

```
new Aggregate();
new Aggregate({ $project: { a: 1, b: 1 } });
new Aggregate({ $project: { a: 1, b: 1 } }, { $skip: 5 });
new Aggregate([{ $project: { a: 1, b: 1 } }, { $skip: 5 }]);
```

Returned when calling Model.aggregate().

Example:

```
Model
.aggregate({ $match: { age: { $gte: 21 }}})
.unwind('tags')
.exec(callback)
```

Note:

- The documents returned are plain javascript objects, not mongoose documents (since any shape of document can be returned).
- Requires MongoDB >= 2.1

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

- Mongoose does **not** cast pipeline stages. `new Aggregate({ $match: { _id: '00000000000000000000000a' } });` will not work unless `_id` is a string in the database. Use `new Aggregate({ $match: { _id: mongoose.Types.ObjectId('00000000000000000000000a') } });` instead.

show code

Aggregate#allowDiskUse(`value`, `[tags]`)

Sets the allowDiskUse option for the aggregation query (ignored for < 2.6.0)

Parameters:

- `value` <Boolean> Should tell server it can use hard drive to store data during aggregation.
- `[tags]` <Array> optional tags for this query

See:

- mongodb

Example:

```
Model.aggregate(..).allowDiskUse(true).exec(callback)
```

show code

Aggregate#append(`ops`)

☐ private

Appends new operators to this aggregate pipeline

Parameters:

- `ops` <Object> operator(s) to append

Returns:

- <Aggregate>

Examples:

```
aggregate.append({ $project: { field: 1 }}, { $limit: 2 });

// or pass an array
var pipeline = [{ $match: { daw: 'Logic Audio X' }} ];
aggregate.append(pipeline);
```

show code

Aggregate#cursor(`options`)

Sets the cursor option option for the aggregation query (ignored for < 2.6.0).
Note the different syntax below: .exec() returns a cursor object, and no callback
is necessary.

Parameters:

- `options` <Object> set the cursor batch size

See:

- mongodb

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

Example:

```
var cursor = Model.aggregate(..).cursor({ batchSize: 1000 })
cursor.each(function(error, doc) {
  // use doc
});
```

show code

Aggregate#exec([callback])

Executes the aggregate pipeline on the currently bound Model.

Parameters:

- `[callback]` <Function>

Returns:

- <Promise>

See:

- Promise

Example:

```
aggregate.exec(callback);

// Because a promise is returned, the `callback` is optional
var promise = aggregate.exec();
promise.then(..);
```

show code

private

Aggregate#explain(`callback`)

Execute the aggregation with explain

Parameters:

- `callback` <Function>

Returns:

- <Promise>

Example:

```
Model.aggregate(..).explain(callback)
```

show code

Aggregate#group(`arg`)

Appends a new custom $group operator to this aggregate pipeline.

Parameters:

- `arg` <Object> $group operator contents

Returns:

- <Aggregate>

See:

- $group

private

Examples:

```
aggregate.group({ _id: "$department" });
```

Aggregate#limit(`num`)

Appends a new $limit operator to this aggregate pipeline.

Parameters:

- `num` <Number> maximum number of records to pass to the next stage

Returns:

- <Aggregate>

See:

- $limit

Examples:

```
aggregate.limit(10);
```

Aggregate#lookup(`options`)

Appends new custom $lookup operator(s) to this aggregate pipeline.

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

Parameters:

- `options` <Object> to $lookup as described in the above link

Returns:

- <Aggregate>

See:

- $lookup

Examples:

```
aggregate.lookup({ from: 'users', localField: 'userId', fore
```

show code

Aggregate#match(`arg`)

Appends a new custom $match operator to this aggregate pipeline.

Parameters:

- `arg` <Object> $match operator contents

Returns:

- <Aggregate>

See:

- $match

Examples:

private

```
aggregate.match({ department: { $in: [ "sales", "engineering'
```

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

Aggregate#model(`model`)

Binds this aggregate to a model.

Parameters:

- `model` <Model> the model to which the aggregate is to be bound

Returns:

- <Aggregate>

show code

Aggregate#near(`parameters`)

Appends a new $geoNear operator to this aggregate pipeline.

Parameters:

- `parameters` <Object>

Returns:

- <Aggregate>

See:

☐ private

- $geoNear

NOTE:

MUST be used as the first operator in the pipeline.

Examples:

```
aggregate.near({
  near: [40.724, -73.997],
  distanceField: "dist.calculated", // required
  maxDistance: 0.008,
  query: { type: "public" },
  includeLocs: "dist.location",
  uniqueDocs: true,
  num: 5
});
```

Aggregate#project(`arg`)

Appends a new $project operator to this aggregate pipeline.

Parameters:

- `arg` <Object, String> field specification

Returns:

- <Aggregate>

See:

- projection

Mongoose query selection syntax is also supported.

Examples:

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

```
// include a, include b, exclude _id
aggregate.project("a b -_id");

// or you may use object notation, useful when
// you have keys already prefixed with a "-"
aggregate.project({a: 1, b: 1, _id: 0});

// reshaping documents
aggregate.project({
    newField: '$b.nested'
  , plusTen: { $add: ['$val', 10]}
  , sub: {
       name: '$a'
    }
})

// etc
aggregate.project({ salary_k: { $divide: [ "$salary", 1000 ]
```

show code

Aggregate#read(`pref`, `[tags]`)

Sets the readPreference option for the aggregation query.

Parameters:

- `pref` <String> one of the listed preference options or their aliases
- `[tags]` <Array> optional tags for this query

See:

- mongodb
- driver

private

Example:

```
Model.aggregate(..).read('primaryPreferred').exec(callback)
```

show code

Aggregate#sample(size)

Appends new custom $sample operator(s) to this aggregate pipeline.

Parameters:

- size <Number> number of random documents to pick

Returns:

- <Aggregate>

See:

- $sample

Examples:

```
aggregate.sample(3); // Add a pipeline that picks 3 random d
```

show code

Aggregate#skip(num)

private

Appends a new $skip operator to this aggregate pipeline.

Parameters:

- `num` <Number> number of records to skip before next stage

Returns:

- <Aggregate>

See:

- $skip

Examples:

```
aggregate.skip(10);
```

Aggregate#sort(`arg`)

Appends a new $sort operator to this aggregate pipeline.

Parameters:

- `arg` <Object, String>

Returns:

- <Aggregate> this

See:

- $sort

If an object is passed, values allowed are `asc`, `desc`, `ascending`, `descending`, `1`, and `-1`.

☐ private

If a string is passed, it must be a space delimited list of path names. The sort order of each path is ascending unless the path name is prefixed with `-` which will be treated as descending.

Examples:

```
// these are equivalent
aggregate.sort({ field: 'asc', test: -1 });
aggregate.sort('field -test');
```

show code

Aggregate#then(`[resolve]`, `[reject]`)

Provides promise for aggregate.

Parameters:

- `[resolve]` <Function> successCallback
- `[reject]` <Function> errorCallback

Returns:

- <Promise>

See:

- Promise

Example:

```
Model.aggregate(..).then(successCallback, errorCallback);
```

show code

☐ private

Aggregate#unwind(`fields`)

Appends new custom $unwind operator(s) to this aggregate pipeline.

Parameters:

- `fields` <String> the field(s) to unwind

Returns:

- <Aggregate>

See:

- $unwind

Note that the `$unwind` operator requires the path name to start with '$'. Mongoose will prepend '$' if the specified field doesn't start '$'.

Examples:

```
aggregate.unwind("tags");
aggregate.unwind("a", "b", "c");
```

show code

schematype.js

SchemaType#default(`val`)

private Sets a default value for this SchemaType.

Parameters:

- `val` <Function, T> the default value

Returns:

- <defaultValue>

Example:

```
var schema = new Schema({ n: { type: Number, default: 10 })
var M = db.model('M', schema)
var m = new M;
console.log(m.n) // 10
```

Defaults can be either `functions` which return the value to use as the default or the literal value itself. Either way, the value will be cast based on its schema type before being set during document creation.

Example:

```
// values are cast:
var schema = new Schema({ aNumber: { type: Number, default: 4
var M = db.model('M', schema)
var m = new M;
console.log(m.aNumber) // 4.815162342

// default unique objects for Mixed types:
var schema = new Schema({ mixed: Schema.Types.Mixed });
schema.path('mixed').default(function () {
  return {};
});

// if we don't use a function to return object literals for M
// each document will receive a reference to the same object
// a "shared" object instance:
var schema = new Schema({ mixed: Schema.Types.Mixed });
schema.path('mixed').default({});
var M = db.model('M', schema);
var m1 = new M;
m1.mixed.added = 1;
```

private

mongoose

```
console.log(m1.mixed); // { added: 1 }
var m2 = new M;
console.log(m2.mixed); // { added: 1 }
```

show code

SchemaType#get(`fn`)

Adds a getter to this schematype.

Parameters:

- `fn` <Function>

Returns:

- <SchemaType> this

Example:

```
function dob (val) {
  if (!val) return val;
  return (val.getMonth() + 1) + "/" + val.getDate() + "/" + v
}

// defining within the schema
var s = new Schema({ born: { type: Date, get: dob })

// or by retreiving its SchemaType
var s = new Schema({ born: Date })
s.path('born').get(dob)
```

Getters allow you to transform the representation of the data as it travels from the raw mongodb document to the value that you see.

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

Suppose you are storing credit card numbers and you want to hide everything except the last 4 digits to the mongoose user. You can do so by defining a getter in the following way:

```
function obfuscate (cc) {
  return '****-****-****-' + cc.slice(cc.length-4, cc.length
}

var AccountSchema = new Schema({
  creditCardNumber: { type: String, get: obfuscate }
});

var Account = db.model('Account', AccountSchema);

Account.findById(id, function (err, found) {
  console.log(found.creditCardNumber); // '****-****-****-12
});
```

Getters are also passed a second argument, the schematype on which the getter was defined. This allows for tailored behavior based on options passed in the schema.

```
function inspector (val, schematype) {
  if (schematype.options.required) {
    return schematype.path + ' is required';
  } else {
    return schematype.path + ' is not';
  }
}

var VirusSchema = new Schema({
  name: { type: String, required: true, get: inspector },
  taxonomy: { type: String, get: inspector }
})

var Virus = db.model('Virus', VirusSchema);

Virus.findById(id, function (err, virus) {
  console.log(virus.name);     // name is required
  console.log(virus.taxonomy); // taxonomy is not
})
```

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

show code

SchemaType#index(`options`)

Declares the index options for this schematype.

Parameters:

- `options` <Object, Boolean, String>

Returns:

- <SchemaType> this

Example:

```
var s = new Schema({ name: { type: String, index: true })
var s = new Schema({ loc: { type: [Number], index: 'hashed' 
var s = new Schema({ loc: { type: [Number], index: '2d', spar
var s = new Schema({ loc: { type: [Number], index: { type: '
var s = new Schema({ date: { type: Date, index: { unique: tru
Schema.path('my.path').index(true);
Schema.path('my.date').index({ expires: 60 });
Schema.path('my.path').index({ unique: true, sparse: true })
```

NOTE:

Indexes are created in the background by default. Specify `background: false` to override.

Direction doesn't matter for single key indexes

show code

☐ private

mongoose

- [home]
- [FAQ]
- [plugins]
- [change log]
- [support]
- [fork]
- [guide]
- [API docs]
- [quick start]
- [contributors]
- [prior releases]

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

SchemaType#required(`required`, `[message]`)

Adds a required validator to this SchemaType. The validator gets added to the front of this SchemaType's validators array using `unshift()`.

Parameters:

- `required` <Boolean> enable/disable the validator
- `[message]` <String> optional custom error message

Returns:

- <SchemaType> this

See:

- Customized Error Messages
- SchemaArray#checkRequired
- SchemaBoolean#checkRequired
- SchemaBuffer#checkRequired
- SchemaNumber#checkRequired
- SchemaObjectId#checkRequired
- SchemaString#checkRequired

Example:

```
var s = new Schema({ born: { type: Date, required: true })

// or with custom error message

var s = new Schema({ born: { type: Date, required: '{PATH} is

// or through the path API

Schema.path('name').required(true);

// with custom error messaging

Schema.path('name').required(true, 'grrr :( ');

// or make a path conditionally required based on a function
```

☐ private

mongoose

```
var isOver18 = function() { return this.age &gt;= 18; };
Schema.path('voterRegistrationId').required(isOver18);
```

The required validator uses the SchemaType's `checkRequired` function to
determine whether a given value satisfies the required validator. By default,
a value satisfies the required validator if `val != null` (that is, if
the value is not null nor undefined). However, most built-in mongoose schema
types override the default `checkRequired` function:

show code

SchemaType(`path`, `[options]`, `[instance]`)

SchemaType constructor

Parameters:

- `path` <String>
- `[options]` <Object>
- `[instance]` <String>

show code

SchemaType#select(`val`)

Sets default `select()` behavior for this path.

private

Parameters:

- `val` <Boolean>

Returns:

- <SchemaType> this

Set to `true` if this path should always be included in the results, `false` if it should be excluded by default. This setting can be overridden at the query level.

Example:

```
T = db.model('T', new Schema({ x: { type: String, select: tru
T.find(..); // field x will always be selected ..
// .. unless overridden;
T.find().select('-x').exec(callback);
```

show code

SchemaType#set(`fn`)

Adds a setter to this schematype.

Parameters:

- `fn` <Function>

Returns:

- <SchemaType> this

Example:

```
function capitalize (val) {
  if (typeof val !== 'string') val = '';
```

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

```
  return val.charAt(0).toUpperCase() + val.substring(1);
}

// defining within the schema
var s = new Schema({ name: { type: String, set: capitalize }}
                                                             
// or by retreiving its SchemaType
var s = new Schema({ name: String })
s.path('name').set(capitalize)
```

Setters allow you to transform the data before it gets to the raw mongodb document and is set as a value on an actual key.

Suppose you are implementing user registration for a website. Users provide an email and password, which gets saved to mongodb. The email is a string that you will want to normalize to lower case, in order to avoid one email having more than one account -- e.g., otherwise, avenue@q.com can be registered for 2 accounts via avenue@q.com and AvEnUe@Q.CoM.

You can set up email lower case normalization easily via a Mongoose setter.

```
function toLower (v) {
  return v.toLowerCase();
}

var UserSchema = new Schema({
  email: { type: String, set: toLower }
})

var User = db.model('User', UserSchema)

var user = new User({email: 'AVENUE@Q.COM'})
console.log(user.email); // 'avenue@q.com'

// or
var user = new User
user.email = 'Avenue@Q.com'
console.log(user.email) // 'avenue@q.com'
```

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

As you can see above, setters allow you to transform the data before it gets to the raw mongodb document and is set as a value on an actual key.

NOTE: we could have also just used the built-in `lowercase: true` SchemaType option instead of defining our own function.

```
new Schema({ email: { type: String, lowercase: true }})
```

Setters are also passed a second argument, the schematype on which the setter was defined. This allows for tailored behavior based on options passed in the schema.

```
function inspector (val, schematype) {
  if (schematype.options.required) {
    return schematype.path + ' is required';
  } else {
    return val;
  }
}

var VirusSchema = new Schema({
  name: { type: String, required: true, set: inspector },
  taxonomy: { type: String, set: inspector }
})

var Virus = db.model('Virus', VirusSchema);
var v = new Virus({ name: 'Parvoviridae', taxonomy: 'Parvovi

console.log(v.name);     // name is required
console.log(v.taxonomy); // Parvovirinae
```

show code

SchemaType#sparse(bool)

private

Declares a sparse index.

Parameters:

- `bool` <Boolean>

Returns:

- <SchemaType> this

Example:

```
var s = new Schema({ name: { type: String, sparse: true })
Schema.path('name').index({ sparse: true });
```

show code

SchemaType#text(`bool`)

Declares a full text index.

Parameters:

- `bool` <Boolean>

Returns:

- <SchemaType> this

Example:

```
var s = new Schema({name : {type: String, text : true })
 Schema.path('name').index({text : true});
```

private

show code

SchemaType#unique(`bool`)

Declares an unique index.

Parameters:

- `bool` <Boolean>

Returns:

- <SchemaType> this

Example:

```
var s = new Schema({ name: { type: String, unique: true }});
Schema.path('name').index({ unique: true });
```

NOTE: violating the constraint returns an `E11000` error from MongoDB when saving, not a Mongoose validation error.

show code

SchemaType#validate(`obj`, `[errorMsg]`, `[type]`)

Adds validator(s) for this document path.

Parameters:

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

- `obj` <RegExp, Function, Object> validator
- `[errorMsg]` <String> optional error message
- `[type]` <String> optional validator type

Returns:

- <SchemaType> this

Validators always receive the value to validate as their first argument and must return `Boolean`. Returning `false` means validation failed.

The error message argument is optional. If not passed, the default generic error message template will be used.

Examples:

```
// make sure every value is equal to "something"
function validator (val) {
  return val == 'something';
}
new Schema({ name: { type: String, validate: validator }});

// with a custom error message

var custom = [validator, 'Uh oh, {PATH} does not equal "somet
new Schema({ name: { type: String, validate: custom }});

// adding many validators at a time

var many = [
    { validator: validator, msg: 'uh oh' }
  , { validator: anotherValidator, msg: 'failed' }
]
new Schema({ name: { type: String, validate: many }});

// or utilizing SchemaType methods directly:

var schema = new Schema({ name: 'string' });
schema.path('name').validate(validator, 'validation of `{PATH
```

Error message templates:

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

From the examples above, you may have noticed that error messages support basic templating. There are a few other template keywords besides `{PATH}` and `{VALUE}` too. To find out more, details are available here

Asynchronous validation:

Passing a validator function that receives two arguments tells mongoose that the validator is an asynchronous validator. The first argument passed to the validator function is the value being validated. The second argument is a callback function that must called when you finish validating the value and passed either `true` or `false` to communicate either success or failure respectively.

```
schema.path('name').validate(function (value, respond) {
  doStuff(value, function () {
    ...
    respond(false); // validation failed
  })
}, '{PATH} failed validation.');

// or with dynamic message

schema.path('name').validate(function (value, respond) {
  doStuff(value, function () {
    ...
    respond(false, 'this message gets to the validation erro
  });
}, 'this message does not matter');
```

You might use asynchronous validators to retreive other documents from the database to validate against or to meet other I/O bound validation needs.

Validation occurs `pre('save')` or whenever you manually execute document#validate.

If validation fails during `pre('save')` and no callback was passed to receive the error, an `error` event will be emitted on your Models associated db connection, passing the validation error object along.

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

```
var conn = mongoose.createConnection(..);
conn.on('error', handleError);

var Product = conn.model('Product', yourSchema);
var dvd = new Product(..);
dvd.save(); // emits error on the `conn` above
```

If you desire handling these errors at the Model level, attach an `error` listener to your Model and the event will instead be emitted there.

```
// registering an error listener on the Model lets us handle
Product.on('error', handleError);
```

show code

promise.js

Promise#addBack(`listener`)

Adds a single function as a listener to both err and complete.

Parameters:

- `listener` <Function>

Returns:

- <Promise> this

It will be executed with traditional node.js argument position when the promise is resolved.

private

```
promise.addBack(function (err, args...) {
  if (err) return handleError(err);
  console.log('success');
})
```

Alias of mpromise#onResolve.

Deprecated. Use `onResolve` instead.

Promise#addCallback(`listener`)

Adds a listener to the `complete` (success) event.

Parameters:

- `listener` <Function>

Returns:

- <Promise> this

Alias of mpromise#onFulfill.

Deprecated. Use `onFulfill` instead.

Promise#addErrback(`listener`)

Adds a listener to the `err` (rejected) event.

Parameters:

☐ private

- `listener` <Function>

Returns:

- <Promise> this

Alias of mpromise#onReject.

Deprecated. Use `onReject` instead.

Promise#catch(`onReject`)

ES6-style `.catch()` shorthand

Parameters:

- `onReject` <Function>

Returns:

- <Promise>

Promise#end()

Signifies that this promise was the last in a chain of `then()`s: if a handler passed to the call to `then` which produced this promise throws, the exception will go uncaught.

See:

- mpromise#end

private

Example:

```
var p = new Promise;
p.then(function(){ throw new Error('shucks') });
setTimeout(function () {
  p.fulfill();
  // error was caught and swallowed by the promise returned 
  // p.then(). we either have to always register handlers on
  // the returned promises or we can do the following...
}, 10);

// this time we use .end() which prevents catching thrown er
var p = new Promise;
var p2 = p.then(function(){ throw new Error('shucks') }).end
setTimeout(function () {
  p.fulfill(); // throws "shucks"
}, 10);
```

Promise#error(`err`)

Rejects this promise with `err`.

Parameters:

- `err` <Error, String>

Returns:

- <Promise> this

If the promise has already been fulfilled or rejected, not action is taken.

Differs from #reject by first casting `err` to an `Error` if it is not `instanceof Error`.

show code

☐ private

Promise#on(`event`, `listener`)

Adds `listener` to the `event`.

Parameters:

- `event` <String>
- `listener` <Function>

Returns:

- <Promise> this

See:

- mpromise#on

If `event` is either the success or failure event and the event has already been emitted, the`listener` is called immediately and passed the results of the original emitted event.

Promise(`fn`)

Promise constructor.

Parameters:

- `fn` <Function> a function which will be called when the promise is resolved that accepts `fn(err, ...){}` as signature

Inherits:

- mpromise

private

Events:

- `err`: Emits when the promise is rejected
- `complete`: Emits when the promise is fulfilled

Promises are returned from executed queries. Example:

```
var query = Candy.find({ bar: true });
var promise = query.exec();
```

DEPRECATED. Mongoose 5.0 will use native promises by default (or bluebird,
if native promises are not present) but still
support plugging in your own ES6-compatible promises library. Mongoose 5.0
will **not** support mpromise.

show code

Promise#reject(`reason`)

Rejects this promise with `reason`.

Parameters:

- `reason` <Object, String, Error>

Returns:

- <Promise> this

See:

- mpromise#reject

If the promise has already been fulfilled or rejected, not action is taken.

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

Promise#resolve(`[err]`, `[val]`)

Resolves this promise to a rejected state if `err` is passed or a fulfilled state if no `err` is passed.

Parameters:

- `[err]` <Error> error or null
- `[val]` <Object> value to fulfill the promise with

If the promise has already been fulfilled or rejected, not action is taken.

`err` will be cast to an Error if not already instanceof Error.

NOTE: overrides mpromise#resolve to provide error casting.

show code

Promise#then(`onFulFill`, `onReject`)

Creates a new promise and returns it. If `onFulfill` or `onReject` are passed, they are added as SUCCESS/ERROR callbacks to this promise after the nextTick.

Parameters:

- `onFulFill` <Function>
- `onReject` <Function>

Returns:

- <Promise> newPromise

See:

- promises-A+
- mpromise#then

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

Conforms to promises/A+ specification.

Example:

```
var promise = Meetups.find({ tags: 'javascript' }).select('_i
promise.then(function (meetups) {
  var ids = meetups.map(function (m) {
    return m._id;
  });
  return People.find({ meetups: { $in: ids }).exec();
}).then(function (people) {
  if (people.length &lt; 10000) {
    throw new Error('Too few people!!!');
  } else {
    throw new Error('Still need more people!!!');
  }
}).then(null, function (err) {
  assert.ok(err instanceof Error);
});
```

Promise.complete(`args`)

Fulfills this promise with passed arguments.

Parameters:

- `args` <T>

Alias of mpromise#fulfill.

Deprecated. Use `fulfill` instead.

☐ private

mongoose

Promise.ES6(`resolver`)

ES6-style promise constructor wrapper around mpromise.

show code

Parameters:

- `resolver` <Function>

Returns:

- <Promise> new promise

Promise.fulfill(`args`)

Fulfills this promise with passed arguments.

Parameters:

- `args` <T>

See:

- https://github.com/aheckmann/mpromise#fulfill

ES6Promise.js

ES6Promise(`fn`)

☐ private

mongoose

ES6 Promise wrapper constructor.

Parameters:

- `fn` <Function> a function which will be called when the promise is resolved that accepts `fn(err, ...){}` as signature

Promises are returned from executed queries. Example:

```
var query = Candy.find({ bar: true });
var promise = query.exec();
```

DEPRECATED. Mongoose 5.0 will use native promises by default (or bluebird,
if native promises are not present) but still
support plugging in your own ES6-compatible promises library.
Mongoose 5.0
will **not** support mpromise.

show code

model.js

Model#$where(`argument`)

Creates a `Query` and specifies a `$where` condition.

Parameters:

- `argument` <String, Function> is a javascript string or anonymous function

Returns:

- <Query>

☐ private

See:

- Query.$where

Sometimes you need to query for things in mongodb using a JavaScript expression. You can do so via `find({ $where: javascript })`, or you can use the mongoose shortcut method $where via a Query chain or from your mongoose Model.

```
Blog.$where('this.username.indexOf("val") !== -1').exec(funct
```

Model#increment()

Signal that we desire an increment of this documents version.

See:

- versionKeys

Example:

```
Model.findById(id, function (err, doc) {
  doc.increment();
  doc.save(function (err) { .. })
})
```

show code

private Model#model(name)

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

Returns another Model instance.

Parameters:

- `name` <String> model name

Example:

```
var doc = new Tank;
doc.model('User').findById(id, callback);
```

show code

Model(`doc`)

Model constructor

Parameters:

- `doc` <Object> values with which to create the document

Inherits:

- Document

Events:

- `error`: If listening to this event, it is emitted when a document was saved without passing a callback and an `error` occurred. If not listening, the event bubbles to the connection used to create this Model.
- `index`: Emitted after `Model#ensureIndexes` completes. If an error occurred it is passed with the event.

☐ private

mongoose

- `index-single-start`: Emitted when an individual index starts within `Model#ensureIndexes`. The fields and options being used to build the index are also passed with the event.
- `index-single-done`: Emitted when an individual index finishes within `Model#ensureIndexes`. If an error occurred it is passed with the event. The fields, options, and index name are also passed.

Provides the interface to MongoDB collections as well as creates document instances.

show code

Model#remove(`[fn]`)

Removes this document from the db.

Parameters:

- `[fn]` <function(err, product)> optional callback

Returns:

- <Promise> Promise

Example:

```
product.remove(function (err, product) {
  if (err) return handleError(err);
  Product.findById(product._id, function (err, product) {
    console.log(product) // null
  })
})
```

As an extra measure of flow control, remove will return a Promise (bound to `fn` if passed) so it could be chained, or hooked to recive errors

private

Example:

```
product.remove().then(function (product) {
   ...
}).onRejected(function (err) {
   assert.ok(err)
})
```

show code

Model#save(`[options]`, `[options.safe]`, `[options.validateBeforeSave]`, `[fn]`)

Saves this document.

Parameters:

- `[options]` <Object> options optional options
- `[options.safe]` <Object> overrides schema's safe option
- `[options.validateBeforeSave]` <Boolean> set to false to save without validating.
- `[fn]` <Function> optional callback

Returns:

- <Promise> Promise

See:

- middleware

Example:

```
product.sold = Date.now();
product.save(function (err, product, numAffected) {
```

private

```
   if (err) ..
})
```

The callback will receive three parameters

- `err` if an error occurred
- `product` which is the saved `product`
- `numAffected` will be 1 when the document was successfully persisted to MongoDB, otherwise 0. Unless you tweak mongoose's internals, you don't need to worry about checking this parameter for errors - checking `err` is sufficient to make sure your document was properly saved.

As an extra measure of flow control, save will return a Promise.

Example:

```
product.save().then(function(product) {
   ...
});
```

For legacy reasons, mongoose stores object keys in reverse order on initial
save. That is, `{ a: 1, b: 2 }` will be saved as `{ b: 2, a: 1 }` in
MongoDB. To override this behavior, set
the `toObject.retainKeyOrder` option
to true on your schema.

show code

Model.aggregate(`[...]`, `[callback]`)

Performs aggregations on the models collection.

show code

Parameters:

private

- `[...]` <Object, Array> aggregation pipeline operator(s) or operator array
- `[callback]` <Function>

Returns:

- <Aggregate, Promise>

See:

- Aggregate
- MongoDB

If a `callback` is passed, the `aggregate` is executed and a `Promise` is returned. If a callback is not passed, the `aggregate` itself is returned.

Example:

```
// Find the max balance of all accounts
Users.aggregate(
  { $group: { _id: null, maxBalance: { $max: '$balance' }}},
  { $project: { _id: 0, maxBalance: 1 }},
  function (err, res) {
    if (err) return handleError(err);
    console.log(res); // [ { maxBalance: 98000 } ]
  });

// Or use the aggregation pipeline builder.
Users.aggregate()
  .group({ _id: null, maxBalance: { $max: '$balance' } })
  .select('-id maxBalance')
  .exec(function (err, res) {
    if (err) return handleError(err);
    console.log(res); // [ { maxBalance: 98 } ]
});
```

NOTE:

- Arguments are not cast to the model's schema because `$project` operators allow redefining the "shape" of the documents at any stage of the pipeline, which may leave documents in an incompatible format.

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

- The documents returned are plain javascript objects, not mongoose documents (since any shape of document can be returned).
- Requires MongoDB >= 2.1

Model.count(`conditions`, `[callback]`)

Counts number of matching documents in a database collection.

show code

Parameters:

- `conditions` <Object>
- `[callback]` <Function>

Returns:

- <Query>

Example:

```
Adventure.count({ type: 'jungle' }, function (err, count) {
  if (err) ..
  console.log('there are %d jungle adventures', count);
});
```

Model.create(`doc(s)`, `[callback]`)

☐ private

mongoose

- [home]
- [FAQ]
- [plugins]
- [change log]
- [support]
- [fork]
- [guide]
- [API docs]
- [quick start]
- [contributors]
- [prior releases]

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

Shortcut for saving one or more documents to the database. `MyModel.create(docs)` does `new MyModel(doc).save()` for every doc in docs.

show code

Parameters:

- `doc(s)` <Array, Object, *>
- `[callback]` <Function> callback

Returns:

- <Promise>

Hooks Triggered

- `save()`

Example:

```
// pass individual docs
Candy.create({ type: 'jelly bean' }, { type: 'snickers' }, fu
  if (err) // ...
});

// pass an array
var array = [{ type: 'jelly bean' }, { type: 'snickers' }];
Candy.create(array, function (err, candies) {
  if (err) // ...

  var jellybean = candies[0];
  var snickers = candies[1];
  // ...
});

// callback is optional; use the returned promise if you lik
var promise = Candy.create({ type: 'jawbreaker' });
promise.then(function (jawbreaker) {
  // ...
})
```

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

Model.discriminator(`name`, `schema`)

Adds a discriminator type.

show code

Parameters:

- `name` <String> discriminator model name
- `schema` <Schema> discriminator model schema

Example:

```
function BaseSchema() {
  Schema.apply(this, arguments);

  this.add({
    name: String,
    createdAt: Date
  });
}
util.inherits(BaseSchema, Schema);

var PersonSchema = new BaseSchema();
var BossSchema = new BaseSchema({ department: String });

var Person = mongoose.model('Person', PersonSchema);
var Boss = Person.discriminator('Boss', BossSchema);
```

Model.distinct(`field`, `[conditions]`, `[callback]`)

Creates a Query for a `distinct` operation.

private

mongoose

show code

Parameters:

- `field` <String>
- `[conditions]` <Object> optional
- `[callback]` <Function>

Returns:

- <Query>

Passing a `callback` immediately executes the query.

Example

```
Link.distinct('url', { clicks: {$gt: 100}}, function (err, re
  if (err) return handleError(err);

  assert(Array.isArray(result));
  console.log('unique urls with more than 100 clicks', result
})

var query = Link.distinct('url');
query.exec(callback);
```

Model.ensureIndexes(`[options]`, `[cb]`)

Sends `ensureIndex` commands to mongo for each index declared in the schema.

show code

Parameters:

- `[options]` <Object> internal options
- `[cb]` <Function> optional callback

private

Returns:

- <Promise>

Example:

```
Event.ensureIndexes(function (err) {
  if (err) return handleError(err);
});
```

After completion, an `index` event is emitted on this `Model` passing an error if one occurred.

Example:

```
var eventSchema = new Schema({ thing: { type: 'string', uniq
var Event = mongoose.model('Event', eventSchema);

Event.on('index', function (err) {
  if (err) console.error(err); // error occurred during inde
})
```

NOTE: It is not recommended that you run this in production. Index creation may impact database performance depending on your load. Use with caution.

The `ensureIndex` commands are not sent in parallel. This is to avoid the `MongoError: cannot add index with a background operation in progress` error. See this ticket for more information.

Model.find(`conditions, [projection], [options], [callback]`)

Finds documents

show code

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

Parameters:

- `conditions` <Object>
- `[projection]` <Object> optional fields to return (http://bit.ly/1HotzBo)
- `[options]` <Object> optional
- `[callback]` <Function>

Returns:

- <Query>

See:

- field selection
- promise

The `conditions` are cast to their respective SchemaTypes before the command is sent.

Examples:

```
// named john and at least 18
MyModel.find({ name: 'john', age: { $gte: 18 }});

// executes immediately, passing results to callback
MyModel.find({ name: 'john', age: { $gte: 18 }}, function (er

// name LIKE john and only selecting the "name" and "friends'
MyModel.find({ name: /john/i }, 'name friends', function (err

// passing options
MyModel.find({ name: /john/i }, null, { skip: 10 })

// passing options and executing immediately
MyModel.find({ name: /john/i }, null, { skip: 10 }, function

// executing a query explicitly
var query = MyModel.find({ name: /john/i }, null, { skip: 10
query.exec(function (err, docs) {});

// using the promise returned from executing a query
var query = MyModel.find({ name: /john/i }, null, { skip: 10
```

private

```
var promise = query.exec();
promise.addBack(function (err, docs) {});
```

Model.findById(`id, [projection], [options], [callback]`)

Finds a single document by its _id field. `findById(id)` is almost* equivalent to `findOne({ _id: id })`. If you want to query by a document's
`_id`, use `findById()` instead of `findOne()`.

show code

Parameters:

- `id` <Object, String, Number> value of <code>_id</code> to query by
- `[projection]` <Object> optional fields to return (http://bit.ly/1HotzBo)
- `[options]` <Object> optional
- `[callback]` <Function>

Returns:

- <Query>

See:

- field selection
- lean queries

The `id` is cast based on the Schema before sending the command.

Note: `findById()` triggers `findOne` hooks.

- Except for how it treats `undefined`. If you use `findOne()`, you'll see that `findOne(undefined)` and `findOne({ _id: undefined })` are equivalent to `findOne({})` and return arbitrary documents.

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

However, mongoose translates `findById(undefined)` into `findOne({ _id: null })`.

Example:

```
// find adventure by id and execute immediately
Adventure.findById(id, function (err, adventure) {});

// same as above
Adventure.findById(id).exec(callback);

// select only the adventures name and length
Adventure.findById(id, 'name length', function (err, adventu

// same as above
Adventure.findById(id, 'name length').exec(callback);

// include all properties except for `length`
Adventure.findById(id, '-length').exec(function (err, adventu

// passing options (in this case return the raw js objects, r
Adventure.findById(id, 'name', { lean: true }, function (err,

// same as above
Adventure.findById(id, 'name').lean().exec(function (err, doc
```

Model.findByIdAndRemove(`id, [options], [callback]`)

Issue a mongodb findAndModify remove command by a document's _id field. `findByIdAndRemove(id, ...)` is equivalent to `findOneAndRemove({ _id: id }, ...)`.

show code

Parameters:

☐ private

mongoose

- `id` <Object, Number, String> value of <code>_id</code> to query by
- `[options]` <Object>
- `[callback]` <Function>

Returns:

- <Query>

See:

- Model.findOneAndRemove
- mongodb

Finds a matching document, removes it, passing the found document (if any) to the callback.

Executes immediately if `callback` is passed, else a `Query` object is returned.

Options:

- `sort`: if multiple docs are found by the conditions, sets the sort order to choose which doc to update
- `select`: sets the document fields to return

Examples:

```
A.findByIdAndRemove(id, options, callback) // executes
A.findByIdAndRemove(id, options)  // return Query
A.findByIdAndRemove(id, callback) // executes
A.findByIdAndRemove(id) // returns Query
A.findByIdAndRemove()           // returns Query
```

Model.findByIdAndUpdate(`id`, `[update]`, `[options]`, `[callback]`)

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

Issues a mongodb findAndModify update command by a document's _id field.
`findByIdAndUpdate(id, ...)` is equivalent to `findOneAndUpdate({ _id: id }, ...)`.

show code

Parameters:

- `id` <Object, Number, String> value of <code>_id</code> to query by
- `[update]` <Object>
- `[options]` <Object>
- `[callback]` <Function>

Returns:

- <Query>

See:

- Model.findOneAndUpdate
- mongodb

Finds a matching document, updates it according to the `update` arg, passing any `options`, and returns the found document (if any) to the callback. The query executes immediately if `callback` is passed else a Query object is returned.

This function triggers `findOneAndUpdate` middleware.

Options:

- `new`: bool - true to return the modified document rather than the original. defaults to false
- `upsert`: bool - creates the object if it doesn't exist. defaults to false.
- `runValidators`: if true, runs update validators on this command. Update validators validate the update operation against the model's schema.
- `setDefaultsOnInsert`: if this and `upsert` are true, mongoose will apply the defaults specified in the model's schema if a new document is created. This option only works on MongoDB >= 2.4 because it relies on MongoDB's `$setOnInsert` operator.
- `sort`: if multiple docs are found by the conditions, sets the sort order to choose which doc to update

private

mongoose

- [home]
- [FAQ]
- [plugins]
- [change log]
- [support]
- [fork]
- [guide]
- [API docs]
- [quick start]
- [contributors]
- [prior releases]

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

- `select`: sets the document fields to return

Examples:

```
A.findByIdAndUpdate(id, update, options, callback) // execute
A.findByIdAndUpdate(id, update, options)  // returns Query
A.findByIdAndUpdate(id, update, callback) // executes
A.findByIdAndUpdate(id, update)           // returns Query
A.findByIdAndUpdate()                     // returns Query
```

Note:

All top level update keys which are not `atomic` operation names are treated as set operations:

Example:

```
Model.findByIdAndUpdate(id, { name: 'jason borne' }, options

// is sent as
Model.findByIdAndUpdate(id, { $set: { name: 'jason borne' }}
```

This helps prevent accidentally overwriting your document with `{ name: 'jason borne' }`.

Note:

Values are cast to their appropriate types when using the findAndModify helpers.
However, the below are never executed.

- defaults
- setters

`findAndModify` helpers support limited defaults and validation. You can enable these by setting the `setDefaultsOnInsert` and `runValidators` options,
respectively.

If you need full-fledged validation, use the traditional approach of first retrieving the document.

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

```
Model.findById(id, function (err, doc) {
  if (err) ..
  doc.name = 'jason borne';
  doc.save(callback);
});
```

Model.findOne(`[conditions]`, `[projection]`, `[options]`, `[callback]`)

Finds one document.

show code

Parameters:

- `[conditions]` <Object>
- `[projection]` <Object> optional fields to return (http://bit.ly/1HotzBo)
- `[options]` <Object> optional
- `[callback]` <Function>

Returns:

- <Query>

See:

- field selection
- lean queries

The `conditions` are cast to their respective SchemaTypes before the command is sent.

Note: `conditions` is optional, and if `conditions` is null or undefined, mongoose will send an empty `findOne` command to MongoDB, which will return

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

an arbitrary document. If you're querying by `_id`, use `findById()` instead.

Example:

```
// find one iphone adventures - iphone adventures??
Adventure.findOne({ type: 'iphone' }, function (err, adventur

// same as above
Adventure.findOne({ type: 'iphone' }).exec(function (err, adv

// select only the adventures name
Adventure.findOne({ type: 'iphone' }, 'name', function (err,

// same as above
Adventure.findOne({ type: 'iphone' }, 'name').exec(function (

// specify options, in this case lean
Adventure.findOne({ type: 'iphone' }, 'name', { lean: true },

// same as above
Adventure.findOne({ type: 'iphone' }, 'name', { lean: true })

// chaining findOne queries (same as above)
Adventure.findOne({ type: 'iphone' }).select('name').lean().e
```

Model.findOneAndRemove(`conditions`, `[options]`, `[callback]`)

Issue a mongodb findAndModify remove command.

show code

Parameters:

private conditions <Object>

mongoose

- [home]
- [FAQ]
- [plugins]
- [change log]
- [support]
- [fork]
- [guide]
- [API docs]
- [quick start]
- [contributors]
- [prior releases]

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

- `[options]` <Object>
- `[callback]` <Function>

Returns:

- <Query>

See:

- mongodb

Finds a matching document, removes it, passing the found document (if any) to the callback.

Executes immediately if `callback` is passed else a Query object is returned.

Options:

- `sort`: if multiple docs are found by the conditions, sets the sort order to choose which doc to update
- `maxTimeMS`: puts a time limit on the query - requires mongodb >= 2.6.0
- `select`: sets the document fields to return

Examples:

```
A.findOneAndRemove(conditions, options, callback) // execute
A.findOneAndRemove(conditions, options)  // return Query
A.findOneAndRemove(conditions, callback) // executes
A.findOneAndRemove(conditions) // returns Query
A.findOneAndRemove()           // returns Query
```

Values are cast to their appropriate types when using the findAndModify helpers.
However, the below are never executed.

- defaults
- setters

`findAndModify` helpers support limited defaults and validation. You can enable these by setting the `setDefaultsOnInsert` and `runValidators` options,

private

If you need full-fledged validation, use the traditional approach of first retrieving the document.

```
Model.findById(id, function (err, doc) {
  if (err) ..
  doc.name = 'jason borne';
  doc.save(callback);
});
```

Model.findOneAndUpdate(`[conditions]`, `[update]`, `[options]`, `[callback]`)

Issues a mongodb findAndModify update command.

show code

Parameters:

- `[conditions]` <Object>
- `[update]` <Object>
- `[options]` <Object>
- `[callback]` <Function>

Returns:

- <Query>

See:

- mongodb

Finds a matching document, updates it according to the `update` arg, passing any `options`, and returns the found document (if any) to the callback. The query executes immediately if `callback` is passed else a Query object is returned.

Options:

private

- `new`: bool - if true, return the modified document rather than the original. defaults to false (changed in 4.0)
- `upsert`: bool - creates the object if it doesn't exist. defaults to false.
- `fields`: {Object|String} - Field selection. Equivalent to `.select(fields).findOneAndUpdate()`
- `maxTimeMS`: puts a time limit on the query - requires mongodb >= 2.6.0
- `sort`: if multiple docs are found by the conditions, sets the sort order to choose which doc to update
- `runValidators`: if true, runs update validators on this command. Update validators validate the update operation against the model's schema.
- `setDefaultsOnInsert`: if this and `upsert` are true, mongoose will apply the defaults specified in the model's schema if a new document is created. This option only works on MongoDB >= 2.4 because it relies on MongoDB's `$setOnInsert` operator.
- `passRawResult`: if true, passes the raw result from the MongoDB driver as the third callback parameter

Examples:

```
A.findOneAndUpdate(conditions, update, options, callback) //
A.findOneAndUpdate(conditions, update, options)  // returns (
A.findOneAndUpdate(conditions, update, callback) // executes
A.findOneAndUpdate(conditions, update)           // returns (
A.findOneAndUpdate()                             // returns (
```

Note:

All top level update keys which are not `atomic` operation names are treated as set operations:

Example:

```
var query = { name: 'borne' };
Model.findOneAndUpdate(query, { name: 'jason borne' }, optio

// is sent as
Model.findOneAndUpdate(query, { $set: { name: 'jason borne' }
```

☐ private

mongoose

This helps prevent accidentally overwriting your document with `{ name: 'jason borne' }`.

Note:

Values are cast to their appropriate types when using the findAndModify helpers.
However, the below are never executed.

- defaults
- setters

`findAndModify` helpers support limited defaults and validation. You can enable these by setting the `setDefaultsOnInsert` and `runValidators` options,
respectively.

If you need full-fledged validation, use the traditional approach of first retrieving the document.

```
Model.findById(id, function (err, doc) {
  if (err) ..
  doc.name = 'jason borne';
  doc.save(callback);
});
```

Model.geoNear(`GeoJSON`, `options`, `[callback]`)

geoNear support for Mongoose

show code

Parameters:

- `GeoJSON` <Object, Array> point or legacy coordinate pair [x,y] to search near
- `options` <Object> for the query

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js
querystream.js
connection.js
utils.js
browser.js
drivers/node-mongodb-native/collection.js
drivers/node-mongodb-native/connection.js
error/messages.js
error/validation.js
error.js
querycursor.js
virtualtype.js
schema.js
document.js
types/subdocument.js

- `[callback]` <Function> optional callback for the query

Returns:

- <Promise>

See:

- http://docs.mongodb.org/manual/core/2dsphere/
- http://mongodb.github.io/node-mongodb-native/api-generated/collection.html?highlight=geonear#geoNear

Options:

- `lean` {Boolean} return the raw object
- All options supported by the driver are also supported

Example:

```
// Legacy point
Model.geoNear([1,3], { maxDistance : 5, spherical : true }, f
   console.log(results);
});

// geoJson
var point = { type : "Point", coordinates : [9,9] };
Model.geoNear(point, { maxDistance : 5, spherical : true }, f
   console.log(results);
});
```

Model.geoSearch(`conditions`, `options`, `[callback]`)

Implements `$geoSearch` functionality for Mongoose

show code

private

Parameters:

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

- `conditions` <Object> an object that specifies the match condition (required)
- `options` <Object> for the geoSearch, some (near, maxDistance) are required
- `[callback]` <Function> optional callback

Returns:

- <Promise>

See:

- http://docs.mongodb.org/manual/reference/command/geoSearch/
- http://docs.mongodb.org/manual/core/geohaystack/

Example:

```
var options = { near: [10, 10], maxDistance: 5 };
Locations.geoSearch({ type : "house" }, options, function(er
  console.log(res);
});
```

Options:

- `near` {Array} x,y point to search for
- `maxDistance` {Number} the maximum distance from the point near that a result can be
- `limit` {Number} The maximum number of results to return
- `lean` {Boolean} return the raw object instead of the Mongoose Model

Model.hydrate(`obj`)

Shortcut for creating a new Document from existing raw data, pre-saved in the DB.

private The document returned has no paths marked as modified initially.

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

show code

Parameters:

- `obj` <Object>

Returns:

- <Document>

Example:

```
// hydrate previous data into a Mongoose document
var mongooseCandy = Candy.hydrate({ _id: '54108337212ffb6d45
```

Model.insertMany(`doc(s)`, `[callback]`)

Shortcut for validating an array of documents and inserting them into MongoDB if they're all valid. This function is faster than `.create()` because it only sends one operation to the server, rather than one for each
document.

show code

Parameters:

- `doc(s)` <Array, Object, *>
- `[callback]` <Function> callback

Returns:

- <Promise>

This function does **not** trigger save middleware.

Example:
private

mongoose

```
var arr = [{ name: 'Star Wars' }, { name: 'The Empire Strike
Movies.insertMany(arr, function(error, docs) {});
```

Model.mapReduce(o, [callback])

Executes a mapReduce command.

show code

Parameters:

- `o` <Object> an object specifying map-reduce options
- `[callback]` <Function> optional callback

Returns:

- <Promise>

See:

- http://www.mongodb.org/display/DOCS/MapReduce

`o` is an object specifying all mapReduce options as well as the map and reduce functions. All options are delegated to the driver implementation. See node-mongodb-native mapReduce() documentation for more detail about options.

Example:

```
var o = {};
o.map = function () { emit(this.name, 1) }
o.reduce = function (k, vals) { return vals.length }
User.mapReduce(o, function (err, results) {
  console.log(results)
})
```

☐ private

Other options:

- `query` {Object} query filter object.
- `sort` {Object} sort input objects using this key
- `limit` {Number} max number of documents
- `keeptemp` {Boolean, default:false} keep temporary data
- `finalize` {Function} finalize function
- `scope` {Object} scope variables exposed to map/reduce/finalize during execution
- `jsMode` {Boolean, default:false} it is possible to make the execution stay in JS. Provided in MongoDB > 2.0.X
- `verbose` {Boolean, default:false} provide statistics on job execution time.
- `readPreference` {String}
- `out*` {Object, default: {inline:1}} sets the output target for the map reduce job.

*** out options:**

- `{inline:1}` the results are returned in an array
- `{replace: 'collectionName'}` add the results to collectionName: the results replace the collection
- `{reduce: 'collectionName'}` add the results to collectionName: if dups are detected, uses the reducer / finalize functions
- `{merge: 'collectionName'}` add the results to collectionName: if dups exist the new docs overwrite the old

If `options.out` is set to `replace`, `merge`, or `reduce`, a Model instance is returned that can be used for further querying. Queries run against this model are all executed with the `lean` option; meaning only the js object is returned and no Mongoose magic is applied (getters, setters, etc).

Example:

```
var o = {};
o.map = function () { emit(this.name, 1) }
o.reduce = function (k, vals) { return vals.length }
o.out = { replace: 'createdCollectionNameForResults' }
o.verbose = true;

User.mapReduce(o, function (err, model, stats) {
  console.log('map reduce took %d ms', stats.processtime)
  model.find().where('value').gt(10).exec(function (err, doc
```

☐ private

```
    console.log(docs);
  });
})

// a promise is returned so you may instead write
var promise = User.mapReduce(o);
promise.then(function (model, stats) {
  console.log('map reduce took %d ms', stats.processtime)
  return model.find().where('value').gt(10).exec();
}).then(function (docs) {
   console.log(docs);
}).then(null, handleError).end()
```

Model.populate(`docs`, `options`, `[callback(err,doc)]`)

Populates document references.

show code

Parameters:

- `docs` <Document, Array> Either a single document or array of documents to populate.
- `options` <Object> A hash of key/val (path, options) used for population.
- `[callback(err,doc)]` <Function> Optional callback, executed upon completion. Receives <code>err</code> and the <code>doc(s)</code>.

Returns:

- <Promise>

Available options:

- path: space delimited path(s) to populate
- select: optional fields to select

private

- match: optional query conditions to match
- model: optional name of the model to use for population
- options: optional query options like sort, limit, etc

Examples:

```
// populates a single object
User.findById(id, function (err, user) {
  var opts = [
      { path: 'company', match: { x: 1 }, select: 'name' }
    , { path: 'notes', options: { limit: 10 }, model: 'overr
  ]

  User.populate(user, opts, function (err, user) {
    console.log(user);
  });
});

// populates an array of objects
User.find(match, function (err, users) {
  var opts = [{ path: 'company', match: { x: 1 }, select: 'na

  var promise = User.populate(users, opts);
  promise.then(console.log).end();
})

// imagine a Weapon model exists with two saved documents:
//   { _id: 389, name: 'whip' }
//   { _id: 8921, name: 'boomerang' }
// and this schema:
// new Schema({
//   name: String,
//   weapon: { type: ObjectId, ref: 'Weapon' }
// });

var user = { name: 'Indiana Jones', weapon: 389 }
Weapon.populate(user, { path: 'weapon', model: 'Weapon' }, fu
  console.log(user.weapon.name) // whip
})

// populate many plain objects
var users = [{ name: 'Indiana Jones', weapon: 389 }]
users.push({ name: 'Batman', weapon: 8921 })
```

☐ private

```
Weapon.populate(users, { path: 'weapon' }, function (err, use
  users.forEach(function (user) {
    console.log('%s uses a %s', users.name, user.weapon.name)
    // Indiana Jones uses a whip
    // Batman uses a boomerang
  });
});
// Note that we didn't need to specify the Weapon model becau
// it is in the schema's ref
```

Model.remove(conditions, [callback])

Removes documents from the collection.

show code

Parameters:

- conditions <Object>
- [callback] <Function>

Returns:

- <Query>

Example:

```
Comment.remove({ title: 'baby born from alien father' }, func

});
```

Note:

To remove documents without waiting for a response from MongoDB, do not pass a callback, then call exec on the returned Query:

private

```
var query = Comment.remove({ _id: id });
query.exec();
```

Note:

This method sends a remove command directly to MongoDB, no Mongoose documents are involved. Because no Mongoose documents are involved, *no middleware (hooks) are executed*.

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

Model.update(`conditions`, `doc`, `[options]`, `[callback]`)

Updates documents in the database without returning them.

show code

Parameters:

- `conditions` <Object>
- `doc` <Object>
- `[options]` <Object>
- `[callback]` <Function>

Returns:

- <Query>

See:

- strict
- response

Examples:

```
MyModel.update({ age: { $gt: 18 } }, { oldEnough: true }, fn
MyModel.update({ name: 'Tobi' }, { ferret: true }, { multi: t
  if (err) return handleError(err);
```

private

mongoose

```
  console.log('The raw response from Mongo was ', raw);
});
```

Valid options:

- `safe` (boolean) safe mode (defaults to value set in schema (true))
- `upsert` (boolean) whether to create the doc if it doesn't match (false)
- `multi` (boolean) whether multiple documents should be updated (false)
- `runValidators`: if true, runs update validators on this command. Update validators validate the update operation against the model's schema.
- `setDefaultsOnInsert`: if this and `upsert` are true, mongoose will apply the defaults specified in the model's schema if a new document is created. This option only works on MongoDB >= 2.4 because it relies on MongoDB's `$setOnInsert` operator.
- `strict` (boolean) overrides the `strict` option for this update
- `overwrite` (boolean) disables update-only mode, allowing you to overwrite the doc (false)

All `update` values are cast to their appropriate SchemaTypes before being sent.

The `callback` function receives `(err, rawResponse)`.

- `err` is the error if any occurred
- `rawResponse` is the full response from Mongo

Note:

All top level keys which are not `atomic` operation names are treated as set operations:

Example:

```
var query = { name: 'borne' };
Model.update(query, { name: 'jason borne' }, options, callba

// is sent as
Model.update(query, { $set: { name: 'jason borne' }}, option
// if overwrite option is false. If overwrite is true, sent
```

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js

querystream.js

connection.js

utils.js

browser.js

drivers/node-mongodb-native/collection.js

drivers/node-mongodb-native/connection.js

error/messages.js

error/validation.js

error.js

querycursor.js

virtualtype.js

schema.js

document.js

types/subdocument.js

This helps prevent accidentally overwriting all documents in your collection with `{ name: 'jason borne' }`.

Note:

Be careful to not use an existing model instance for the update clause (this won't work and can cause weird behavior like infinite loops). Also, ensure that the update clause does not have an _id property, which causes Mongo to return a "Mod on _id not allowed" error.

Note:

To update documents without waiting for a response from MongoDB, do not pass a `callback`, then call `exec` on the returned Query:

```
Comment.update({ _id: id }, { $set: { text: 'changed' }}).exe
```

Note:

Although values are casted to their appropriate types when using update, the following are *not* applied:

- defaults
- setters
- validators
- middleware

If you need those features, use the traditional approach of first retrieving the document.

```
Model.findOne({ name: 'borne' }, function (err, doc) {
  if (err) ..
  doc.name = 'jason borne';
  doc.save(callback);
})
```

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

types/subdocument.js ▸

Model.where(`path`, `[val]`)

Creates a Query, applies the passed conditions, and returns the Query.

show code

Parameters:

- `path` <String>
- `[val]` <Object> optional value

Returns:

- <Query>

For example, instead of writing:

```
User.find({age: {$gte: 21, $lte: 65}}, callback);
```

we can instead write:

```
User.where('age').gte(21).lte(65).exec(callback);
```

Since the Query class also supports `where` you can continue chaining

```
User
.where('age').gte(21).lte(65)
.where('name', /^b/i)
... etc
```

Model#base

Base Mongoose instance the model uses.

private

show code

Model#baseModelName

If this is a discriminator model, `baseModelName` is the name of the base model.

show code

Model#collection

Collection the model uses.

show code

Model#db

Connection the model uses.

show code

☐ private

mongoose

Model#discriminators

Registered discriminators for this model.

show code

Model#modelName

The name of the model

show code

Model#schema

Schema the model uses.

show code

collection.js

Collection(`name`, `conn`, `opts`)

Abstract Collection constructor

private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

Parameters:

- `name` <String> name of the collection
- `conn` <Connection> A MongooseConnection instance
- `opts` <Object> optional collection options

This is the base class that drivers inherit from and implement.

show code

Collection#ensureIndex()

Abstract method that drivers must implement.

show code

Collection#find()

Abstract method that drivers must implement.

show code

Collection#findAndModify()

Abstract method that drivers must implement.

☐ private

show code

Collection#findOne()

Abstract method that drivers must implement.

show code

Collection#getIndexes()

Abstract method that drivers must implement.

show code

Collection#insert()

Abstract method that drivers must implement.

show code

☐ private

mongoose

- [home]
- [FAQ]
- [plugins]
- [change log]
- [support]
- [fork]
- [guide]
- [API docs]
- [quick start]
- [contributors]
- [prior releases]

index.js
querystream.js
connection.js
utils.js
browser.js
drivers/node-mongodb-native/collection.js
drivers/node-mongodb-native/connection.js
error/messages.js
error/validation.js
error.js
querycursor.js
virtualtype.js
schema.js
document.js

Collection#mapReduce()

Abstract method that drivers must implement.

show code

Collection#save()

Abstract method that drivers must implement.

show code

Collection#update()

Abstract method that drivers must implement.

show code

Collection#collectionName

The collection name

show code

☐ private

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs
- quick start
- contributors
- prior releases

index.js ▸

querystream.js ▸

connection.js ▸

utils.js ▸

browser.js ▸

drivers/node-mongodb-native/collection.js ▸

drivers/node-mongodb-native/connection.js ▸

error/messages.js ▸

error/validation.js ▸

error.js ▸

querycursor.js ▸

virtualtype.js ▸

schema.js ▸

document.js ▸

Collection#conn

The Connection instance

show code

Collection#name

The collection name

show code

☐ private