eBook Gratuit

APPRENEZ mongoose

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#mongoose

Table des matières

À propos 1

Chapitre 1: Commencer avec la mangouste 2

Remarques 2

Versions 2

Examples 4

Installation 4

Connexion à la base de données MongoDB: 4

Connexion avec options et rappel 5

Chapitre 2: mangouste pré et post middleware (crochets) 6

Examples 6

Middleware 6

Chapitre 3: Middleware Mongoose 8

Remarques 8

Il existe deux types de middleware 8

Pré et Post Crochets 8

Examples 8

Middleware pour hacher le mot de passe de l'utilisateur avant de l'enregistrer 8

Chapitre 4: Population de Mongoose 11

Syntaxe 11

Paramètres 11

Examples 11

Peupler simple 11

Négliger quelques champs 13

Ne remplir que quelques champs 13

Population imbriquée 14

Chapitre 5: Population de Mongoose 16

Syntaxe 16

Paramètres 16

Examples 16

Un exemple simple de peuplement mangouste 16

Chapitre 6: Requêtes Mongoose 18
Introduction 18
Examples 18
Trouver une requête 18
Chapitre 7: Schémas Mongoose 19
Examples 19
Schéma de base 19
Méthodes de schéma 19
Schéma statique 19
Schéma GeoObjects 20
Enregistrement de l'heure actuelle et de l'heure de mise à jour 20
Chapitre 8: Schémas Mongoose 22
Examples 22
Créer un schéma 22
Crédits 23

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: mongoose

It is an unofficial and free mongoose ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official mongoose.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Commencer avec la mangouste

Remarques

> **Mongoose** est un outil de modélisation d'objet **MongoDB** conçu pour fonctionner dans un environnement asynchrone.
>
> Tout dans Mongoose commence par un schéma. Chaque schéma correspond à une collection MongoDB et définit la forme des documents dans cette collection.

Mongoose facilite le travail avec la base de données MongoDB.

Nous pouvons facilement structurer notre base de données à l'aide de `Schemas and Models` , automatiser certaines choses lorsque des enregistrements sont ajoutés ou mis à jour à l'aide de `Middlewares/Hooks` et obtenir facilement les données dont nous avons besoin en `querying` nos modèles.

Liens importants

- Démarrage rapide de la mangue
- Référentiel Mongoose GitHub
- Documentation Mongoose
- Plugins Mongoose

Versions

Dernière version: Version **4.6.0** publiée le **2 septembre 2016**

Toutes les versions peuvent être trouvées sur
https://github.com/Automattic/mongoose/blob/master/History.md

Version	Date de sortie
1.0.1	2011-02-02
1.1.6	2011-03-22
1.3.0	Le 2011-04-19
1.3.1	2011-04-27
1.3.4	2011-05-17
1.4.0	2011-06-10
1.5.0	2011-06-27

Version	Date de sortie
1.6.0	2011-07-07
2.0.0	2011-08-24
2.3.4	2011-10-18
2.5.0	2012-01-26
3.0.0	2012-08-07
3.1.2	2012-09-10
3.2.0	2012-09-27
3.5.0	2012-12-10
3.5.6	2013-02-14
3.6.0	2013-03-18
3.6.5	2013-04-15
3.8.0	2013-10-31
3.8.10	2014-05-20
3.8.15	2014-08-17
4.0.0	2015-03-25
4.0.6	2015-06-21
4.1.0	2015-07-24
4.2.0	2015-10-22
4.2.10	2015-12-08
4.3.5	2016-01-09
4.4.0	2016-02-02
4.4.4	2016-02-17
4.4.8	2016-03-18
4.4.13	2016-04-21
4.4.18	2016-05-21

Version	Date de sortie
4.5.0	2016-06-13
4.5.5	2016-07-18
4.5.8	2016-08-01
4.5.9	2016-08-14
4.5.10	2016-08-23
4.6.0	2016-09-02

Examples

Installation

Installer **mongoose** est aussi simple que d'exécuter la commande `npm`

```
npm install mongoose --save
```

Mais assurez-vous également d'avoir installé `MongoDB` pour votre système d'exploitation ou avoir accès à une base de données MongoDB.

Connexion à la base de données MongoDB:

1. Importez la mangouste dans l'application:

```
import mongoose from 'mongoose';
```

2. Spécifiez une bibliothèque Promise:

```
mongoose.Promise = global.Promise;
```

3. Connectez-vous à MongoDB:

```
mongoose.connect('mongodb://127.0.0.1:27017/database');

/* Mongoose connection format looks something like this */
mongoose.connect('mongodb://USERNAME:PASSWORD@HOST::PORT/DATABASE_NAME');
```

Remarque:

- Par défaut, mongoose se connecte à MongoDB sur le port `27017` , qui est le port par défaut utilisé par MongoDB.
- Pour vous connecter à MongoDB hébergé ailleurs, utilisez la deuxième syntaxe. Entrez le

nom d'utilisateur MongoDB, le mot de passe, l'hôte, le port et le nom de la base de données.

Le port MongoDB est 27017 par défaut; Utilisez le nom de votre application comme nom de base de données.

Connexion avec options et rappel

Mongoose connect a 3 paramètres, uri, options et la fonction de rappel. Pour les utiliser, voir exemple ci-dessous.

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

var uri = 'mongodb://localhost:27017/DBNAME';

var options = {
    user: 'user1',
    pass: 'pass'
}

mongoose.connect(uri, options, function(err){
    if (err) throw err;
    // if no error == connected
});
```

Lire Commencer avec la mangouste en ligne:
https://riptutorial.com/fr/mongoose/topic/1168/commencer-avec-la-mangouste

Chapitre 2: mangouste pré et post middleware (crochets)

Examples

Middleware

Les middleware (également appelés pré et post hooks) sont des fonctions qui passent le contrôle lors de l'exécution de fonctions asynchrones. Le middleware est spécifié au niveau du schéma et est utile pour écrire des plugins. Mongoose 4.0 dispose de 2 types de middleware: le middleware de document et le middleware de requête. Le middleware de document est pris en charge pour les fonctions de document suivantes.

- init
- valider
- enregistrer
- retirer

Le middleware de requête est pris en charge pour les fonctions Model et Query suivantes.

- compter
- trouver
- trouverOne
- findOneAndRemove
- findOneAndUpdate
- mettre à jour

Le middleware de document et le middleware de requête prennent en charge les pré et post-hooks.

Pré

Il existe deux types de pré-crochets, série et parallèle.

En série

Les middleware série sont exécutés les uns après les autres, lorsque chaque middleware appelle ensuite.

```
var schema = new Schema(..);
schema.pre('save', function(next) {
  // do stuff
  next();
});
```

Parallèle

Le middleware parallèle offre un contrôle de flux plus fin.

var schema = new Schema (..);

```
// `true` means this is a parallel middleware. You **must** specify `true`
// as the second parameter if you want to use parallel middleware.
schema.pre('save', true, function(next, done) {
  // calling next kicks off the next middleware in parallel
  next();
  setTimeout(done, 100);
});
```

La méthode accrochée, dans ce cas enregistrer, ne sera pas exécutée tant que chaque logiciel intermédiaire n'a pas terminé.

Post middleware

post middleware sont exécutés après la méthode hooked et tous ses pré-middleware ont été complétés. post middleware ne reçoivent pas directement le contrôle de flux, par exemple, aucun callback suivant ou fini ne lui est transmis. Les post-hooks permettent d'enregistrer les programmes d'écoute traditionnels pour ces méthodes.

```
schema.post('init', function(doc) {
  console.log('%s has been initialized from the db', doc._id);
});
schema.post('validate', function(doc) {
  console.log('%s has been validated (but not saved yet)', doc._id);
});
schema.post('save', function(doc) {
  console.log('%s has been saved', doc._id);
});
schema.post('remove', function(doc) {
  console.log('%s has been removed', doc._id);
});
```

Lire mangouste pré et post middleware (crochets) en ligne:
https://riptutorial.com/fr/mongoose/topic/4131/mangouste-pre-et-post-middleware--crochets-

Chapitre 3: Middleware Mongoose

Remarques

En mangouste, les **Middlewares** sont aussi appelés `pre` et `post` hooks.

Il existe deux types de middleware

Ces deux middleware prennent en charge les **pré** et **post-** hooks.

1. **Middleware de documents**

   Son support pour les fonctions document `init` , `validate` , `save` et `remove`

2. **Intergiciel de requête**

   Son support pour les fonctions de requête `count` , `find` , `findOne` , `findOneAndRemove` , `findOneAndUpdate` , `insertMany` et `update` .

Pré et Post Crochets

Il existe deux types de **pré-** crochets

1. **en série**

   Comme son nom l'indique, il est exécuté dans un ordre séquentiel

2. **parallèle**

   Middleware parallèle offre un contrôle plus de débit à grains fins et la `hooked method` n'est pas exécutée jusqu'à ce que `done` est appelé par tous les middleware parallèles.

Post Middleware sont exécutés une fois la `hooked method` et tous ses `pre` middleware terminés.

Les méthodes accrochées sont les fonctions supportées par le middleware de document. `init, validate, save, remove`

Examples

Middleware pour hacher le mot de passe de l'utilisateur avant de l'enregistrer

Ceci est un exemple de ***`Serial Document Middleware`***

Dans cet exemple, nous allons écrire un middleware qui convertira le mot de passe en texte brut en mot de passe haché avant de l'enregistrer dans la base de données.

Ce middleware sera automatiquement activé lors de la création d'un nouvel utilisateur ou de la mise à jour des informations utilisateur existantes.

FILENAME: `User.js`

```
// lets import mongoose first
import mongoose from 'mongoose'

// lets create a schema for the user model
const UserSchema = mongoose.Schema(
  {
    name: String,
    email: { type: String, lowercase: true, requird: true },
    password: String,
  },
);


/**
 * This is the middleware, It will be called before saving any record
 */
UserSchema.pre('save', function(next) {

  // check if password is present and is modified.
  if ( this.password && this.isModified('password') ) {

    // call your hashPassword method here which will return the hashed password.
    this.password = hashPassword(this.password);

  }

  // everything is done, so let's call the next callback.
  next();

});


// lets export it, so we can import it in other files.
export default mongoose.model( 'User', UserSchema );
```

Maintenant, chaque fois que nous sauvons un utilisateur, ce middleware vérifie si le mot de passe est **défini** et **modifié** (c'est pourquoi nous n'utilisons pas le mot de passe des utilisateurs si celui-ci n'a pas été modifié).

NOM `App.js` **FICHIER:** `App.js`

```
import express from 'express';
import mongoose from 'mongoose';

// lets import our User Model
import User from './User';

// connect to MongoDB
mongoose.Promise = global.Promise;
mongoose.connect('mongodb://127.0.0.1:27017/database');
```

```
let app = express();
/* ... express middlewares here .... */


app.post( '/', (req, res) => {

  /*
    req.body = {
      name: 'John Doe',
      email: 'john.doe@gmail.com',
      password: '!trump'
    }
   */

  // validate the POST data

  let newUser = new User({
    name: req.body.name,
    email: req.body.email,
    password: req.body.password,
  });

  newUser.save( ( error, record ) => {
    if (error) {
      res.json({
        message: 'error',
        description: 'some error occoured while saving the user in database.'
      });
    } else {
      res.json({
        message: 'success',
        description: 'user details successfully saved.',
        user: record
      });
    }
  });

});



let server = app.listen( 3000, () => {
    console.log(`Server running at http://localhost:3000` );
  }
);
```

Lire Middleware Mongoose en ligne: https://riptutorial.com/fr/mongoose/topic/6646/middleware-mongoose

Chapitre 4: Population de Mongoose

Syntaxe

1. Model.Query.populate (path, [select], [model], [match], [options]);

Paramètres

Param	Détails
chemin	**String** - La clé de champ à remplir
sélectionner	**Object, String** - Sélection de champ pour la requête de population.
modèle	**Modèle** - Instance du modèle référencé
rencontre	**Object** - Remplir les conditions
options	**Object** - Options de requête

Examples

Peupler simple

Mongoose populate est utilisé pour afficher les données des documents référencés provenant d'autres collections.

Disons que nous avons un modèle `Person` qui a référencé des documents appelés `Address` .

Modèle de personne

```
var Person = mongoose.model('Person', {
    fname: String,
    mname: String,
    lname: String,
    address: {type: Schema.Types.ObjectId, ref: 'Address'}
});
```

Modèle d'adresse

```
var Address = mongoose.model('Address', {
    houseNum: String,
    street: String,
    city: String,
    state: String,
    country: String
});
```

Pour renseigner `Address` inside `Person` utilisant ObjectId, en utilisant disons `findOne()` , utilisez la fonction `populate()` et ajoutez l' `address` de la clé de champ comme premier paramètre.

```
Person.findOne({_id: req.params.id})
    .populate('address') // <- use the populate() function
    .exec(function(err, person) {
        // do something.
        // variable `person` contains the final populated data
    });
```

Ou

```
Person.findOne({_id: req.params.id}, function(err, person) {
    // do something
    // variable `person` contains the final populated data
})
.populate('address');
```

La requête ci-dessus doit produire le document ci-dessous.

Personne doc

```
{
    "_id":"123abc",
    "fname":"John",
    "mname":"Kennedy",
    "lname":"Doe",
    "address":"456def" // <- Address' Id
}
```

Adresse Doc

```
{
    "_id":"456def",
    "houseNum":"2",
    "street":"Street 2",
    "city":"City of the dead",
    "state":"AB",
    "country:"PH"
}
```

Doc peuplé

```
 {
    "_id":"123abc",
    "fname":"John",
    "mname":"Kennedy",
    "lname":"Doe",
    "address":{
        "_id":"456def",
        "houseNum":"2",
        "street":"Street 2",
        "city":"City of the dead",
        "state":"AB",
        "country:"PH"
```

```
    }
}
```

Négliger quelques champs

Disons que vous **ne voulez pas que** les champs `houseNum` et `street houseNum` dans le champ d' `address` du document final rempli, utilisez le `populate()` comme suit,

```
Person.findOne({_id: req.params.id})
    .populate('address', '-houseNum -street')  // note the `-` symbol
    .exec(function(err, person) {
        // do something.
        // variable `person` contains the final populated data
    });
```

Ou

```
Person.findOne({_id: req.params.id}, function(err, person) {
    // do something
    // variable `person` contains the final populated data
})
.populate('address', '-houseNum -street');  // note the `-` symbol
```

Cela produira le document final final suivant,

Doc peuplé

```
 {
    "_id":"123abc",
    "fname":"John",
    "mname":"Kennedy",
    "lname":"Doe",
    "address":{
        "_id":"456def",
        "city":"City of the dead",
        "state":"AB",
        "country:"PH"
    }
}
```

Ne remplir que quelques champs

Si vous **ne voulez que** les champs `houseNum` et `street` dans le champ d' `address` du document final rempli, utilisez la fonction `populate()` comme suit dans les deux méthodes ci-dessus,

```
Person.findOne({_id: req.params.id})
    .populate('address', 'houseNum street')
    .exec(function(err, person) {
        // do something.
        // variable `person` contains the final populated data
    });
```

Ou

```
Person.findOne({_id: req.params.id}, function(err, person) {
    // do something
    // variable `person` contains the final populated data
})
.populate('address', 'houseNum street');
```

Cela produira le document final final suivant,

Doc peuplé

```
 {
    "_id":"123abc",
    "fname":"John",
    "mname":"Kennedy",
    "lname":"Doe",
    "address":{
        "_id":"456def",
        "houseNum":"2",
        "street":"Street 2"
    }
}
```

Population imbriquée

Disons que vous avez un schéma d' `user` , qui contient le `name` , le `contactNo` , l' `address` et les `friends` .

```
var UserSchema = new mongoose.Schema({
    name : String,
    contactNo : Number,
    address : String,
    friends :[{
        type: mongoose.Schema.Types.ObjectId,
        ref : User
    }]
});
```

Si vous voulez trouver un utilisateur, ses **amis** et **amis d'amis** , vous devez faire la population sur 2 niveaux, à savoir la **population imbriquée** .

Pour trouver des amis et des amis d'amis:

```
User.find({_id : userID})
    .populate({
        path : 'friends',
        populate : { path : 'friends'}//to find friends of friends
    });
```

Tous les `parameters` et `options` de `populate` peuvent également être utilisés à l'intérieur du peuplement imbriqué pour obtenir le résultat souhaité.

De même, vous pouvez `populate` plus de `levels` fonction de vos besoins.

Il n'est pas recommandé de faire de la population imbriquée pour plus de 3 niveaux. Si vous devez effectuer un remplissage imbriqué sur plus de 3 niveaux, vous devrez peut-être restructurer votre schéma.

Lire Population de Mongoose en ligne: https://riptutorial.com/fr/mongoose/topic/2616/population-de-mongoose

Chapitre 5: Population de Mongoose

Syntaxe

- Query.populate (path, [select], [model], [match], [options])

Paramètres

Paramètre	Explication
chemin	< **Object, String** > soit le chemin à remplir, soit un objet spécifiant tous les paramètres
[sélectionner]	< **Object, String** > Sélection de champ pour la requête de population (peut utiliser `'-id'` pour inclure tout sauf le champ `id`)
[modèle]	< **Modèle** > Le modèle que vous souhaitez utiliser pour population.Si le nom n'est pas spécifié, populate recherche le modèle par son nom dans le champ ref du schéma.
[rencontre]	< **Objet** > Conditions pour la population
[options]	< **Object** > Options pour la requête de population (tri, etc.)

Examples

Un exemple simple de peuplement mangouste

`.populate()` dans Mongoose vous permet de remplir une référence que vous avez dans votre collection ou document actuel avec les informations de cette collection. Le précédent peut paraître déroutant mais je pense qu'un exemple aidera à dissiper toute confusion.

Le code suivant crée deux collections, User et Post:

```
var mongoose = require('mongoose'),
  Schema = mongoose.Schema

var userSchema = Schema({
  name: String,
  age: Number,
  posts: [{ type: Schema.Types.ObjectId, ref: 'Post' }]
});

var PostSchema = Schema({
  user: { type: Schema.Types.ObjectId, ref: 'User' },
  title: String,
  content: String
```

```
});

var User = mongoose.model('User', userSchema);
var Post = mongoose.model('Post', postSchema);
```

Si nous voulions remplir tous les articles pour chaque utilisateur lorsque nous `.find({})` tous les utilisateurs, nous pourrions procéder comme suit:

```
User
  .find({})
  .populate('posts')
  .exec(function(err, users) {
    if(err) console.log(err);
    //this will log all of the users with each of their posts
    else console.log(users);
  })
```

Lire Population de Mongoose en ligne: https://riptutorial.com/fr/mongoose/topic/6578/population-de-mongoose

Chapitre 6: Requêtes Mongoose

Introduction

Mongoose est un pilote Node.JS pour MongoDB. Il offre certains avantages par rapport au pilote MongoDB par défaut, comme l'ajout de types aux schémas. Une différence est que certaines requêtes Mongoose peuvent différer de leurs équivalents MongoDB.

Examples

Trouver une requête

Importer un modèle Mongoose (Voir la rubrique "Schémas Mongoose")

```
var User = require("../models/user-schema.js")
```

La méthode findOne renvoie la première entrée de la base de données correspondant au premier paramètre. Le paramètre doit être un objet où la clé est le champ à rechercher et la valeur est la valeur à mettre en correspondance. Cela peut utiliser la syntaxe de requête MongoDB, telle que l'opérateur point (.) Pour rechercher des sous-champs.

```
User.findOne({"name": "Fernando"}, function(err, result){
    if(err) throw err;    //There was an error with the database.
    if(!result) console.log("No one is named Fernando."); //The query found no results.
    else {
        console.log(result.name); //Prints "Fernando"
    }
}
```

Lire Requêtes Mongoose en ligne: https://riptutorial.com/fr/mongoose/topic/9349/requetes-mongoose

Chapitre 7: Schémas Mongoose

Examples

Schéma de base

Un schéma utilisateur de base:

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

var userSchema = new mongoose.Schema({
    name: String,
    password: String,
    age: Number,
    created: {type: Date, default: Date.now}
});

var User = mongoose.model('User', userSchema);
```

Types de schéma

Méthodes de schéma

Des méthodes peuvent être définies sur les schémas pour aider à faire les choses liées à ce (s) schéma (s), et les garder bien organisés.

```
userSchema.methods.normalize = function() {
    this.name = this.name.toLowerCase();
};
```

Exemple d'utilisation:

```
erik = new User({
    'name': 'Erik',
    'password': 'pass'
});
erik.normalize();
erik.save();
```

Schéma statique

Les statistiques de schéma sont des méthodes pouvant être appelées directement par un modèle (contrairement aux méthodes de schéma, qui doivent être appelées par une instance d'un document Mongoose). Vous affectez une statique à un schéma en ajoutant la fonction à l'objet `statics` du schéma.

Un exemple d'utilisation est la construction de requêtes personnalisées:

```
userSchema.statics.findByName = function(name, callback) {
    return this.model.find({ name: name }, callback);
}

var User = mongoose.model('User', userSchema)

User.findByName('Kobe', function(err, documents) {
    console.log(documents)
})
```

Schéma GeoObjects

Un schéma générique utile pour travailler avec des objets géo tels que des points, des lignes et des polygones. **Mongoose** et **MongoDB** supportent tous deux **Geojson** .

Exemple d'utilisation dans **Node / Express** :

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

// Creates a GeoObject Schema.
var myGeo= new Schema({
                        name: { type: String },
                        geo : {
                                type : {
                                         type: String,
                                         enum: ['Point', 'LineString', 'Polygon']
                                },
                                coordinates : Array
                         }
});

//2dsphere index on geo field to work with geoSpatial queries
myGeo.index({geo : '2dsphere'});
module.exports = mongoose.model('myGeo', myGeo);
```

Enregistrement de l'heure actuelle et de l'heure de mise à jour

Ce type de schéma sera utile si vous souhaitez conserver la trace de vos éléments par temps d'insertion ou de mise à jour.

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

// Creates a User Schema.
var user = new Schema({
                       name: { type: String },
                       age : { type: Integer},
                       sex : { type: String },
                       created_at: {type: Date, default: Date.now},
                       updated_at: {type: Date, default: Date.now}
});

// Sets the created_at parameter equal to the current time
user.pre('save', function(next){
    now = new Date();
```

```
    this.updated_at = now;
    if(!this.created_at) {
        this.created_at = now
    }
    next();
});

module.exports = mongoose.model('user', user);
```

Lire Schémas Mongoose en ligne: https://riptutorial.com/fr/mongoose/topic/2592/schemas-mongoose

Chapitre 8: Schémas Mongoose

Examples

Créer un schéma

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

//assume Player and Board schemas are already made
var Player = mongoose.model('Player');
var Board = mongoose.model('Board');

//Each key in the schema is associated with schema type (ie. String, Number, Date, etc)
var gameSchema = new mongoose.Schema({
  name: String,
  players: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Player'
  }],
  host: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Player'
  },
  board: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Board'
  },
  active: {
    type: Boolean,
    default: true
  },
  state: {
    type: String,
    enum: ['decision', 'run', 'waiting'],
    default: 'waiting'
  },
  numFlags: {
    type: Number,
    enum: [1,2,3,4]
  },
  isWon: {
    type: Boolean,
    default: false
  }
});

mongoose.model('Game', gameSchema);
```

Lire Schémas Mongoose en ligne: https://riptutorial.com/fr/mongoose/topic/6622/schemas-mongoose

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec la mangouste	4444, CENT1PEDE, Community, Delapouite, dublicator, jisoo, Random User, zurfyx
2	mangouste pré et post middleware (crochets)	Naeem Shaikh
3	Middleware Mongoose	Delapouite, Random User
4	Population de Mongoose	CENT1PEDE, Chinni, Medet Tleukabiluly, Ravi Shankar
5	Requêtes Mongoose	Gibryon Bhojraj
6	Schémas Mongoose	AndreaM16, Ian, zurfyx