CRUD у Mongoose

Розглянемо, як виконувати основні операції з даними Mongoose.

Створення документів

У минулих темах було загалом описано створення та додавання об'єктів у Mongoose. Зокрема, об'єкт моделі може викликати метод **save()** :

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const userScheme = new Schema({
    name: String,
    age: Number
});

const User = mongoose.model("User", userScheme);

async function main() {

    await mongoose.connect("mongodb://127.0.0.1:27017/usersdb");

    const tom = new User({name: "Tom", age: 34});
    // добавляем объект в БД
    await tom.save();
    console.log(tom);
}
main().catch(console.log).finally(async()=>await mongoose.disconnect());
```

Але крім цього можна також використовувати метод **User.create()** :

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const userScheme = new Schema({
    name: String,
    age: Number
```

```
});

const User = mongoose.model("User", userScheme);

async function main() {

    await mongoose.connect("mongodb://127.0.0.1:27017/usersdb");

    // добавляем объект в БД
    const user = await User.create({name: "Sam", age: 28})
    console.log(user);
}
main().catch(console.log).finally(async()=>await mongoose.disconnect());
```

В якості параметра метод `User.create()` приймає об'єкт, що зберігається, і повертає збережений об'єкт.

Отримання даних

Для отримання даних можна використовувати цілий набір методів:

- **find** : повертає всі об'єкти, які відповідають критерію фільтрації
- **findById**: возвращает один объект по значению поля _id
- **findOne**: возвращает один объект, который соответствует критерию фильтрации

Метод **find()** в качестве первого параметра принимает критерий фильтрации, а второй параметр - функция обратного вызова, в которую передаются полученные из бд документы:

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const userScheme = new Schema({name: String, age: Number});
const User = mongoose.model("User", userScheme);

async function main() {

    await mongoose.connect("mongodb://127.0.0.1:27017/usersdb");

    // получаем все объекты из БД
    const users = await User.find({});
    console.log(users);
}
main().catch(console.log).finally(async()=>await mongoose.disconnect());
```

Если в качестве критерия фильтрации передаются пустые фигурные скобки ({}), то возвращаются все объекты:

```
[
  {
    _id: new ObjectId("6377c17b71c0bd75cec4d488"),
    name: 'Bill',
    age: 41,
    __v: 0
  },
  {
    _id: new ObjectId("6377c7a46fa33e19ac7a7c41"),
    name: 'Tom',
    age: 34,
    __v: 0
  },
  {
    _id: new ObjectId("6377ce352461051cdc78252a"),
    name: 'Sam',
    age: 28,
    __v: 0
  }
]
```

Изменим код для получения только тех пользователей, у которых имя - Tom:

```
const users = await User.find({name: "Tom"});
```

Метод **findOne()** работает аналогично методу find, только возвращает один объект:

```
const user = await User.findOne({name: "Bill"});
```

И метод **findById()** возвращает документ с определенным идентификатором:

```
const id = "6377c7a46fa33e19ac7a7c41";
const user = await User.findById(id);
```

Удаление данных

Для удаления применяется метод **deleteOne()** (удаляет один объект) и **deleteMany()** (удаляет все объекты, которые соответствуют критери.). В эти методы передается критерий фильтрации документов на удаление. Например, удалим всех пользователей, у которых возраст равен 41:

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

```

```
const userScheme = new Schema({name: String, age: Number});
const User = mongoose.model("User", userScheme);

async function main() {

    await mongoose.connect("mongodb://127.0.0.1:27017/usersdb");

    // удаляем все объекты из БД, у которых age=41
    const result = await User.deleteMany({age:41});
    console.log(result);
}
main().catch(console.log).finally(async()=>await mongoose.disconnect());
```

Метод `User.deleteMany()` возвращает объект, который содержит информацию об операции удаления:

```
{ acknowledged: true, deletedCount: 1 }
```

Так, свойства `deletedCount` хранит количество удаленных строк

Применение метода **deleteOne()** для удаления одного документа будет аналогичным:

```
const result = await User.deleteOne({name:"Tom"})
console.log(result);    // { acknowledged: true, deletedCount: 1 }
```

Также для удаления одного документа можно использовать метод **findOneAndDelete()**:

```
const user = await User.findOneAndDelete({name:"Sam"})
console.log(user);
```

В качестве результата он возвращает удаленный документ.

```
{ _id: new ObjectId("6377bca2d16bfca92631cc10"), name: 'Sam', age: 28 }
```

И частная разновидность этого метода - удаление по полю _id в виде метода **findByIdAndDelete()**:

```
const id = "6377c72806fb915eb6621ffd";
const user = await User.findByIdAndDelete(id)
console.log(user);
```

Изменение данных

Для обновления данных в модели предусмотрены методы **updateOne()** и **updateMany()**. Первый метод обновляет один документ, который соответствует критерию, а второй метод обновляет все документы, которые соответствую критерию выборки:

```
const mongoose = require("mongoose");
```

```
const Schema = mongoose.Schema;

const userScheme = new Schema({name: String, age: Number});
const User = mongoose.model("User", userScheme);

async function main() {

    await mongoose.connect("mongodb://127.0.0.1:27017/usersdb");

    // У всех документов изменяем значение поля name с "Tom" на "Tom Smith"
    const result = await User.updateOne({name: "Tom"}, {name: "Tom Smith"})
    console.log(result);
}
main().catch(console.log).finally(async()=>await mongoose.disconnect());
```

Первый параметр метода - критерий фильтрации. В данном случае мы находим всех пользователей, у которых имя "Tom". А второй параметр описывает, что и как надо изменить. То есть здесь мы меняем имя на "Tom Smith". Возвращает метод результат операции обновления:

```
{
  acknowledged: true,
  modifiedCount: 1,
  upsertedId: null,
  upsertedCount: 0,
  matchedCount: 1
}
```

Аналогично работает метод `updateMany`.

Обновление по id

Нередко для обновления используется фильтрация по _id. И на этот случай мы можем использовать метод **findByIdAndUpdate()**:

```
const id = "6377ce352461051cdc78252a";
const user = await User.findByIdAndUpdate(id, {name: "Sam", age: 25});
console.log("Обновленный объект", user);
```

Первый параметр метода - значения для поля _id у обновляемого документа, а второй - набор новых значений для полей объекта. Результатом метода является обновленный документ:

```
Обновленный объект {
  _id: new ObjectId("6377ce352461051cdc78252a"),
```

```
  name: 'Sam',
  age: 28,
  __v: 0
}
```

Але за промовчанням передається старий стан документа. Якщо ж нам треба отримати документ вже в зміненому стані, то метод findByIdAndUpdate необхідно передати як третій параметр об'єкт `{new: true}`(при значенні false повертається стара копія):

```
const id = "6377ce352461051cdc78252a";
const user = await User.findByIdAndUpdate(id, {name: "Mike", age: 21}, {new:
console.log("Обновленный объект", user);
```

Якщо нам необхідно оновити та повернути оновлений документ не тільки за id, а взагалі за будь-яким критерієм, то можна використовувати метод **findOneAndUpdate** :

```
const user = await User.findOneAndUpdate({name: "Mike"}, {name: "Alex", age:2
console.log("Обновленный объект", user);
```

Перший параметр представляє критерій вибірки. Другий параметр – оновлені значення документа. Третій параметр вказує, що хочемо повернути варіант документа саме після оновлення - `{new: true}`. Результат методу – оновлений документ.

Назад Зміст Вперед