Mongoose ODM (Object-Document Mapper)

Data Mapper and Document-oriented DB

- a Data Access Layer that performs bidirectional transfer of data between a persistent data store (often a relational database) and an in-memory data representation (the domain layer).
- to keep the in-memory representation and the persistent data store independent of each other and the data mapper itself
- **document-oriented database / document store** : a computer program, designed for storing, retrieving and managing document-oriented information.
-

Mongoose

- mongodb object modeling for node.js
- Install : **$ npm install mongoose**
- **How to Use :**

```
const mongoose = require('mongoose');
mongoose.connect('mongoodb://localhost/aneka');
const db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', function() {
	console.log('were connected!')
});
const kittySchema = new mongoose.Schem({
	name: String
});

const Kitten = mongoose.model('Kitten', kittySchema);

const silence = new Kitten({ name: 'Silence' });
```

Define Schema

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

var blogSchema = new Schema({
  title:  String, // String is shorthand for {type: String}
  comments: [{ body: String, date: Date }],
  hidden: Boolean,
  meta: {
    favs:  Number
  }
});
```

- Instance methods

```
  // define a schema
  var animalSchema = new Schema({ name: String, type: String });
  // assign a function to the "methods" object of our animalSchema
  animalSchema.methods.findSimilarTypes = function(cb) {
    return mongoose.model('Animal').find({ type: this.type }, cb);
  };
// HOW TO USE
var Animal = mongoose.model('Animal', animalSchema);
  var dog = new Animal({ type: 'dog' });
  dog.findSimilarTypes(function(err, dogs) {
    console.log(dogs); // woof
  });
```

Static Methods

```
// Assign a function to the "statics" object of our animalSchema
animalSchema.statics.findByName = function(name) {
  return this.find({ name: new RegExp(name, 'i') });
};
// Or, equivalently, you can call `animalSchema.static()`.
animalSchema.static('findByBreed', function(breed) {
  return this.find({ breed });
});

const Animal = mongoose.model('Animal', animalSchema);
let animals = await Animal.findByName('fido');
animals = animals.concat(await Animal.findByBreed('Poodle'));
```

Models

- responsible for creating and reading documents from the underlying MongoDB database.
- When you call mongoose.model() on a schema, Mongoose compiles a model for you.

```
var schema = new mongoose.Schema({ name: 'string',
size: 'string' });

 var Tank = mongoose.model('Tank', schema);
```

Documents

- Mongoose documents represent a one-to-one mapping to documents as stored in MongoDB. Each document is an instance of its Model.
- An instance of a model is called a document

```
const MyModel = mongoose.model('Test', new Schema({ name:
String }));
const doc = new MyModel();

doc instanceof MyModel; // true
doc instanceof mongoose.Model; // true
doc instanceof mongoose.Document; // true
```

Queries of Mongoose

- Static Method : Model.deleteMany(), Model.deleteOne(), Model.find(), Model.findById(), Model.findByIdAndDelete(), Model.findByIdAndRemove(), Model.findByIdAndUpdate(), Model.findOne(), Model.findOneAndDelete(), Model.findOneAndRemove(), Model.findOneAndReplace(), Model.findOneAndUpdate(), Model.replaceOne(), Model.updateMany(), Model.updateOne()
- 2 ways of Execute Mongoose Queries
  - Callback
  - .then() function
-
-

Execute Mongoose Queries with Callback

- you specify your query as a JSON document. The JSON document's syntax is the same as the MongoDB shell.
- the callback follows the pattern callback(error, results). What results is depends on the operation: For findOne() it is a potentially-null single document, find() a list of documents, count() the number of documents, update() the number of documents affected, etc. The API docs for Models provide more detail on what is passed to the callbacks.

```
var Person = mongoose.model('Person', yourSchema);
// find each person with a last name matching 'Ghost',
selecting the `name` and `occupation` fields
Person.findOne({ 'name.last': 'Ghost' }, 'name occupation',
function (err, person) {
  if (err) return handleError(err);
  // Prints "Space Ghost is a talk show host".
  console.log('%s %s is a %s.', person.name.first,
person.name.last,
    person.occupation);
 });
```

```
// With a JSON doc
Person.
  find({
    occupation: /host/,
    'name.last': 'Ghost',
    age: { $gt: 17, $lt: 66 },
    likes: { $in: ['vaporizing', 'talking'] }
  }).
  limit(10).
  sort({ occupation: -1 }).
  select({ name: 1, occupation: 1 }).
  exec(callback);
```

```
// Using query builder
Person.
  find({ occupation: /host/ }).
  where('name.last').equals('Ghost').
  where('age').gt(17).lt(66).
  where('likes').in(['vaporizing', 'talking']).
  limit(10).
  sort('-occupation').
  select('name occupation').

  exec(callback);
```

Execute Mongoose Queries with .then() function

- Convenience for co and async/await
- Calling a query's .then() can execute the query multiple times

```
const q = MyModel.updateMany({}, { isDeleted: true },
function() {
  console.log('Update 1');
});

q.then(() => console.log('Update 2')); // execute 1 again

q.then(() => console.log('Update 3')); // execute 1 again
```

Queries VS Aggregation

- use queries where possible.
- Aggregation can do many of the same things that queries can
- Mongoose hydrate() query result, but Aggregation results are always POJOs, not Mongoose documents.
-

```
const docs = await Person.aggregate([{ $match: {
'name.last': 'Ghost' } }]);

docs[0] instanceof mongoose.Document; // false
```

- Mongoose also doesn't cast aggregation pipelines : you're responsible for ensuring the values you pass in to an aggregation pipeline have the correct type

```
const doc = await Person.findOne();
const idString = doc._id.toString();
// Finds the `Person`, because Mongoose casts `idString` to
an ObjectId
const queryRes = await Person.findOne({ _id: idString });
// Does **not** find the `Person`, because Mongoose doesn't
cast aggregation
// pipelines.
 const aggRes = await Person.aggregate([{ $match: { _id:
 idString } }])
```

References to other documents ?

more info reference documents

Thanks