CSCI 4140 – Tutorial 10

Mongoose: Elegant MongoDB object modeling for Node.js

Matt YIU, Man Tung (mtyiu@cse)
SHB 118
Office Hour: Wednesday, 3-5 pm

2016.04.07

Outline

- Introduction
- Getting started
- Connecting to the database
- Object modeling
- CRUD operations – Create, Read, Update, Delete

Introduction – MongoDB

- MongoDB is a **document-oriented** database
  - Classified as a **NoSQL** database
- MongoDB stores data in **JSON-like documents** with **dynamic schemas**
- Often use with Node.js due to their shared use of JavaScript and its object notation (JSON)
- Heroku provides MongoDB add-ons
  - E.g., mLab MongoDB

Introduction – mongoose

- mongoose is an **object data modeling (ODM)** library for MongoDB
- Available as a **Node module**
  - Can be installed with **npm**
- Supported features:
  - Built-in type casting
  - Validation
  - Query building
  - Business logic hooks

Getting started

- Let's start with a Node.js application
- Install Mongoose for your application:

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

- Enable the MongoDB service in Heroku
  - Follow the instructions on pp. 9-10, "**Deploying Node.js Applications on Heroku**"

Connecting to the database

```
var mongoose = require( 'mongoose' );
var uristring = process.env.MONGOLAB_URI || 'mongodb://XXXXX';

var db = mongoose.connection;

db.on( 'error', console.error );
db.once( 'open', function() {
    console.log( 'Connection established' );
    db.close();
    console.log( 'Disconnected' );
} );

mongoose.connect( uristring );
```

Connecting to the database

```
var mongoose = require( 'mongoose' );
var uristring = process.env.MONGOLAB_URI || 'mongodb://XXXXX';

var db = mongoose.connection;

db.on( 'error', console.error );
db.once( 'open', function() {
    console.log( 'Connection established' );
    db.close();
    console.log( 'Disconnected' );
} );

mongoose.connect( uristring );
```

Load the mongoose module

Read the config var provided by Heroku if available. Otherwise, use the URI string provided (you can copy the URI string from Heroku dashboard for local development).

Set up event listeners for “error” and “open” events.

Connect!

Object modeling

- Mongoose maps documents to objects
- Mongoose requires you to define the **schema** before creating an object

Use **mongoose.Schema()** to define a schema.

```
var roleSchema = mongoose.Schema( {
    type : Number,
    description : String
} );
var userSchema = mongoose.Schema( {
    name : String,
    age : Number,
    active : Boolean,
    roles : [ roleSchema ]
} );
```

Property name

You can have array of a schema type by just setting the property type like this.

Property type

Supported types:

- String
- Number
- Date
- Buffer
- Boolean
- Mixed
- Objectid
- Array

Object modeling

- Mongoose maps documents to objects
- Mongoose requires you to define the **schema** before creating an object

```
var roleSchema = mongoose.Schema( {
    type : Number,
    description : String
} );
var userSchema = mongoose.Schema( {
    name : String,
    age : Number,
    active : Boolean,
    roles : [ roleSchema ]
} );
```

Object modeling

- To create or query for an object, you need to create a **model** for this schema first:

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

- It is often useful to put the model in a Node module:

```
module.exports = User;
```

- See **example_code/model.js**

CRUD in Mongoose: Create

- After the connection is established, you can create a new object using the model:

```
var Alice = new User( {
    name : 'Alice',
    age : 18,
    active : true,
    roles : [ { type : 1, description : 'Guest' } ]
} );

Alice.save( function( err ) {
    if ( err )
        throw err;
    else
        console.log( 'Saved' );
} );
```

Instantiate a new user

Save it to the database

Callback function after the object is saved to the database or an error occurred

create.js

CRUD in Mongoose: Read

- After the connection is established, you can read objects using the model:

```
User.find( function( err, users ) {
    // users is an array of User
} );

User.find( { name : 'Alice' }, function( err, users ) {
    // users is an array of User
} );
```

Find all users

Find users whose name is “Alice”.

read.js

CR*U*D in Mongoose: Update

- After the connection is established, you can update an object using the model:

Use the object ID to find the object

```
User.findById( id, function( err, user ) {
    if ( err )
        throw err;
    user.age = 22;
    user.save( function( err ) {
        if ( err )
            throw err;
        else
            console.log( 'Updated' );
    } );
} );
```

Update the property in the object

Save the updated object to the database

update.js

CRU*D* in Mongoose: Delete

- After the connection is established, you can delete an object using the model:

```
User.findById( id, function( err, user ) {
    if ( err )
        throw err;

    user.remove( function( err ) {
        console.log( 'Removed' );
    } );
} );
```

delete.js

Remove the object from the database

Read the documentation!

- There are many different **variants** for executing CRUD operations with Mongoose
  - The set of slides only cover the basic usage
  - Read the documentation for more details: http://mongoosejs.com/docs/
- To examine the data in MongoDB, you can access the **MongoLab UI** from Heroku Dashboard (similar to phpMyAdmin for MySQL database)
  - Reference: https://devcenter.heroku.com/articles/nodejs-mongoose#introspection

– End –