Node.js and MongoDB: A Simple Example

September 22, 2011

***Update**, October 14, 2012:*
If you are interested in creating highly scalable applications with mongoDB, you should really consider using Go (#golang) instead, and check out the Go version of this post.Node.js. After getting through an excellent basic tutorial, I wanted to experiment with connecting to MongoDB, since it would be helpful for some enhancements I have planned for Stealth Mode Watch and BookHunch. There are several drivers available, but the most commonly used and recommended one is node-mongodb-native. Despite the examples, though, it wasn't clear how to collect and return the results of a single query. Both the simple example and the queries example just dump the result to the console within the cursor, and the best answer on the normally reliable StackOverflow site was problematic, too, because the server code was written in a blocking style without callbacks, which defeats the purpose of using Node.js in the first place. Fortunately, though, there is a terrific article about control flow on How to Node which explains how to do multiple asynchronous tasks (either in parallel or serially), and collect all the results. The examples in the control flow article dealt with accessing files from different folders, which was simple enough to change to running MongoDB queries instead. The first step is to install node-mongodb-native using npm:

> *npm config set loglevel info npm install mongodb*

The first line essentially sets npm in a verbose mode, to let you know what it's doing as it runs (a tip from the guys at Nodejitsu.com), and the second actually installs the node-mongodb-native module (it seems that the native parser is obsolete, so there is no need to use the `--mongodb:native` switch in the second line). Here's how to execute multiple queries and collect their results.

```
var Db = require('./node_modules/mongodb').Db,
    Connection = require('./node_modules/mongodb').Connection,
    Server = require('./node_modules/mongodb').Server;

var host = process.env['MONGO_NODE_DRIVER_HOST'] != null ?
           process.env['MONGO_NODE_DRIVER_HOST'] : 'localhost';
var port = process.env['MONGO_NODE_DRIVER_PORT'] != null ?
           process.env['MONGO_NODE_DRIVER_PORT'] : Connection.DEFAULT_PORT;
```

These lines just load the db module and prepare for a connection; no connection has been opened yet. This function executes the query in the db, collects the result documents in a list, and passes them to a callback function when it's done iterating through the query cursor:

```
function runQuery (db, myCollection query, nextFn) {
    // perform the {query} on the collection and invoke the nextFn when done
    db.open(function(err, db) {
 db.collection(myCollection, function(err, collection) {
     collection.find(query, function(err, cursor) {
  cursor.toArray(function(err, docs) {
      console.log("Found " + docs.length + " documents");
      var queryResults = [];
      for(var i=0; i<docs.length; i++) {
   queryResults[queryResults.length] = docs[i];
      }
      db.close();
      nextFn(queryResults);
  });
     });
 });
    });
}
```

So that leaves opening the connection, defining the query or queries, and calling runQuery(). Suppose we have a database consisting of two collections, people and companies, and that we want to search for a given string anywhere among a person's name, a company's name, or even a company's address. Our search function would look like this:

```
function search (personName, companyName, address, nextFn) {
    var data = [], count = 3;
    var doneFn = function(results) {
 data = data.concat(results);
 count--;
 if( count <= 0 ) {
     var uniqueResults = [];
     for(var i=0; i<data.length; i++) {
  if( ! uniqueResults[data[i]['_id']] ) {
      uniqueResults[uniqueResults.length] = data[i];
      uniqueResults[data[i]['_id']] = true;
  }
     }
     nextFn(uniqueResults);
 }
    };
    runQuery(new Db('mydb', new Server(host, port, {})),
      'people',
      {'name':new RegExp('^'+personName, 'i')},
      doneFn);
    runQuery(new Db('mydb', new Server(host, port, {})),
      'companies',
      {'name':new RegExp('^'+companyName, 'i')},
      doneFn);
    runQuery(new Db('mydb', new Server(host, port, {})),
      'companies',
      {'address':new RegExp(address, 'i')},
      doneFn);
}
```

Just like the control flow article, we use a counter in our callback function -- doneFn -- to check when there are no more queries to run, and once that's the case, we iterate through the combined list of results and use the document ObjectId to ensure the list of results is unique. That unique list of results is then passed to the callback function given to search(), i.e., nextFn, which ultimately decides what to do with the combined results. Each call to runQuery() needs its own db connection, opened like this: `new Db('mydb', new Server(host, port, {}))` *because each query will run asynchronously and independently of each other. It's also possible to define a single connection, once, before each of the runQuery() calls happen:* `var db = new Db('mydb', new Server(host, port, {}));` *and just pass the db variable to runQuery(), but then the* `db.close()` *line would have to be removed from inside the runQuery() function, and placed inside the doneFn instead. As for the queries themselves, the beautiful thing about using Javascript and MongoDB together is that any native Javascript object doubles as the query input to MongoDB. So any query that works in the MongoDB command line, will work in Javascript without much or any transformation necessary, unlike the hoops you have to jump through in Python+pymongo. Finally, the function that calls search() has to decide what to do with the results. If we're writing an API and want the results sent back over HTTP as json, all we have to do is this (within the larger context of an http server in Node.js, of course):*

```
apiSearch('Jones', 'Jones', 'California', function(results) {
    var replyJson = {"warning":"No matches found"};
    if( results.length > 0 ) {
 replyJson = {"result":{ "matched":results.length, "matches":results}};
    }
    response.writeHead(200, {"Content-Type": "application/json"});
    response.write(JSON.stringify(replyJson));
    response.end();
});
```

This example searches the database for people or companies named Jones, or companies located in California. The final callback here gets passed the unique list of results from the search.doneFn -- i.e., it is the nextFn passed into search() -- and generates an http response in json format.
