NIIT

Coding JavaScript in MongoDB

Objectives

- By the end of this session, you should be able to:
  - Introduction
  - Connecting to the Driver
  - Executing Simple Queries
  - Callbacks and Error Handling in Node.js
  - Callbacks in Node.js
  - Advanced Queries

Introduction

- In many production situations, it will be software that connects with the database in place of the user. MongoDB is a great place to store and query your data, but often, it's most essential use is to serve as a backend for large-scale applications. These applications write, read, and update data programmatically, usually after being triggered by some condition or user interface.

- To connect your software with a database, you will typically use a library (often provided by the database creator) known as a driver. This driver will help you connect, analyze, read, and write to your database without having to write multiple lines of code for simple actions. It provides functions and abstractions for common use cases, as well as frameworks for working with data extracted from the database.

- Node.js has become one of the primary languages for web-based applications, which we will learn about later in this chapter. However, for now, it is sufficient to know that the ease of integrating Node and MongoDB has proved highly beneficial for both technologies. This symbiotic relationship has also led to the creation of a large numbers of successful Node/MongoDB implementations, from small mobile apps to large-scale web applications.

Connecting to the Driver

- At a high level, the process of using the Node.js driver with MongoDB is similar to connecting directly with the shell. You will specify a MongoDB server URI, several connection parameters, and you can execute queries against collections. This should all be quite familiar; the main difference will be that these instructions will be written in JavaScript instead of Bash or PowerShell.
- Introduction to Node.js
  - The js in Node.js stands for JavaScript because JavaScript is the programming language that Node.js understands. JavaScript typically runs in a browser. However, you can think of Node.js as an engine that executes the JavaScript files on your computer.
  - it is recommended to use an application that will help you with syntax highlighting and formatting, such as Visual Studio Code or Sublime.

Connecting to the Driver (contd.)

- Getting the MongoDB Driver for Node.js
  - The easiest way to install the MongoDB driver for Node.js is to use npm. npm, or the node package manager, is a package management tool used to add, update, and manage different packages used in Node.js programs. In this case, the package you want to add is the MongoDB driver, so, in the directory where the scripts are stored, run the following command in your Terminal or Command Prompt:

```
> npm install mongo --save
```

- The Database and Collection Objects
  - **MongoClient** is the first object you must create in your code. This represents your connection to the MongoDB server.
  - **database** object. Like the mongo shell, once the connection is established, run your commands against a specific database in your server.
  - **collection** object is used to send queries.

Connecting to the Driver (contd.)

- Connection Parameters
  - It's important to know how to establish the connection to MongoClient. There are only two parameters when creating a new client: the URL for your server and any additional connection options. The connection options are optional.
  - Just like the mongo shell, serverURL supports all the MongoDB URI options, meaning you can specify a configuration in this connection string itself, rather than in the second optional parameter.

Executing Simple Queries

- We can run some simple queries against the database. Running queries in the Node.js driver is very similar to running queries in the shell.
- When writing Node.js applications, one of the critical concerns is to ensure that your code is written in such a way that it can be modified, extended, or understood easily, either by yourself in the future or by other professionals who may need to work on the application
  - Creating and Executing find Queries
    - When sending queries in the mongo shell, you must pass a document to the command as a filter for your documents. This is the same in the Node.js driver. You can pass the document directly. However, it is advisable to define the filter as a variable separately and then assign a value.

Executing Simple Queries (contd.)

- Using Cursors and Query Results
  - With larger result sets, you should use cursors
  - In Node.js, there are many ways to access your cursor, of which three are more common patterns, as follows:
    - **toArray:** This will take all the results of the query and place them in a single array. This is easy to use but not very efficient when you are expecting a large result from your query.
    - **each:** This will iterate through each document in the result set, one at a time. This is a good pattern if you want to inspect or use each document in the result.
    - **next:** This will allow you to access the next document in the result set. This is the best pattern to use if you are only looking for a single document or a subset of your results without having to iterate through the entire result.

Callbacks and Error Handling in Node.js

- Callbacks are extra functions (blocks of code) that are passed as parameters to another function that executes first.
- Callbacks allow you to specify the logic to execute only after a function has completed. The reason we have to use callbacks in Node.js instead of simply having the statements be in order is that Node.js is asynchronous.

Callbacks in Node.js

- A callback is a function provided as a parameter to a second function, which allows both functions to be run in order.
- Without using callbacks (or any other synchronization pattern), both functions would start executing right after the other. When using a driver, this would create errors, because the second function may be dependent on the first function finishing before it begins.
- Basic Error Handling in Node.js
  - In the case of callbacks, the err parameter will always exist; however, if there is no error, the value of err is null. This "error-first" pattern to catch errors in asynchronous code is standard practice in NodeJS.

Advanced Queries

- Inserting Data with the Node.js Driver
  - Similar to the mongo shell, we can use either the insertOne or insertMany function to write data into our collection. These functions are called on the collection object. The only parameter we need to pass into these functions is a single document in the case of insertOne, or an array of documents in the case of insertMany.
- Updating and Deleting Data with the Node.js Driver
  - Updating and deleting documents with the driver follows the same pattern as the insert function, where the collection object passes through a callback, checks for errors, and analyzes the results of the operation. All these functions will return a results document. However, between the three operations, the format and information contained within a result document may differ.

Advanced Queries (contd.)

- Writing Reusable Functions
  - in larger, more complex applications, you will want to run many different operations in the same program, depending on the context. For example, in your application, you may want to run the same query multiple times and compare the respective results, or you may want the second query to be modified depending on the output of the first.
- Reading Input from the Command Line
  - Node.js provides some simple ways for us to read input from the command line and use it in our code. Node.js provides a module called **readline** that will allow us to ask the user for input, accept that input, and then use it.
- Creating an Interactive Loop
  - we can create an interactive loop, meaning the application will keep asking for input until an exit condition is met. To make sure we keep looping, we can place our prompt in a function that calls itself, which will keep running the code inside its block until the exit condition stated becomes true.

Summary

- In this session, you learned about:
  - The basic concepts that are essential to the creation of a MongoDB-powered application using the Node.js driver.
  - We even learned to handle errors and create interactive applications.
  - Having a thorough understanding of how these applications are built gives you a unique insight into MongoDB development and how your peers may interact with your MongoDB data.
- In the next chapter, we will dive deeper into improving the performance of your MongoDB interactions and create efficient indexes that will speed up your queries.