Mongo DB ha crecido rápidamente hasta convertirse en una popular base de datos para las aplicaciones web y es un ajuste perfecto para aplicaciones Node.js, lo que le permite escribir Javascript para el cliente, backend y la capa de base de datos. Su naturaleza schemaless es un mejor partido a nuestras estructuras de datos en constante evolución en las aplicaciones web y el soporte integrado para las consultas de ubicación es una ventaja que es difícil de ignorar. El tiro en Replica Sets para escalar, y estamos mirando realmente agradable plataforma para hacer crecer su almacenamiento necesita ahora y en el futuro.

Ahora para tapar descaradamente mi conductor. Se puede descargar a través de la NGP, o descabellada desde el repositorio GitHub. Para instalar a través de la NGP, haga lo siguiente:

npm install mongodb

Una vez que este negocio está cuidado, vamos a pasar a través de los tipos disponibles para el conductor y luego cómo conectarse a la instancia de Mongo DB antes de enfrentar el uso de algunas operaciones CRUD.

**Mongo DB data types**

Así que no es una cosa importante a tener en cuenta cuando se trabaja con Mongo DB, y esa es la diferencia leve correlación entre tipos de Mongo DB apoya y tipos de datos nativos de Javascript. Vamos a echar un vistazo a los tipos soportados fuera de la caja y luego cómo tipos son promovidos por el conductor para adaptarse lo más cercano a los tipos de Javascript nativos como sea posible.

* **Float**
* **Double class**
* **Integers**
* **Long class**
* **Date**
* **RegExp**
* **String**
* **Binary class**
* **Code class**
* **ObjectID class**
* **DbRef class**
* **Symbol class**

Como vemos el tipo de número puede ser un poco difícil debido a la forma en números enteros se implementan en Javascript. La última versión del controlador hará correcta hasta la conversión a 53 bits de complejidad. Si usted necesita para manejar grandes números enteros la recomendación es utilizar la clase larga para operar en los números.

**Conectando a la Base de Datos**

Vamos a conseguir alrededor de la creación de una conexión con la base de datos MongoDB. Saltar directamente en el código vamos a hacer la conexión directa y luego mirar el código.

// Retrieve

var MongoClient = require('mongodb').MongoClient;

// Connect to the db

MongoClient.connect("mongodb://localhost:27017/exampleDb", function(err, db) {

if(!err) {

console.log("We are connected");

}

});

Echemos un rápido vistazo a cómo funciona el código de conexión. Utilice el método de let db.connect utilizar un URI para conectarse a la base de datos Mongo, donde localhost: 27017 es el host del servidor y el puerto y exampleDb el PP queremos conectar. Después de la url notar el hash que contiene la clave auto\_reconnect. Reconexión automática le dice al conductor para volver a intentar el envío de un comando al servidor si se produce un fallo durante su ejecución.

Otra opción útil que te puede pasar en es

poolSize, esto le permite controlar cuántos se abren conexiones TCP en paralelo. El valor por defecto es 5, pero se puede establecer tan alto como usted desea. El controlador utilizará una estrategia de operación por turnos para enviar y leer de la conexión TCP.

Estamos en marcha y funcionando con una conexión a la base de datos. Vamos a pasar y mirar lo que las colecciones son y cómo funcionan.

Mongo DB y Colecciones

Las colecciones son el equivalente a las tablas en las bases de datos tradicionales y contienen todos sus documentos. Una base de datos puede tener muchas colecciones. Entonces, ¿cómo hacemos para definir y utilizar las colecciones. Bueno, hay un par de métodos que podemos utilizar. Vamos a saltar directamente al código y luego mirar el código.

la requiere y y otras cosas inicializar omitido por razones de brevedad

// Retrieve

var MongoClient = require('mongodb').MongoClient;

// Connect to the db

MongoClient.connect("mongodb://localhost:27017/exampleDb", function(err, db) {

if(err) { return console.dir(err); }

db.collection('test', function(err, collection) {});

db.collection('test', {w:1}, function(err, collection) {});

db.createCollection('test', function(err, collection) {});

db.createCollection('test', {w:1}, function(err, collection) {});

});

Three different ways of creating a collection object but slightly different in behavior. Let’s go through them and see what they do

db.collection('test', function(err, collection) {});

This function will not actually create a collection on the database until you actually insert the first document.

db.collection('test', {strict:true}, function(err, collection) {});

Notice the **{strict:true}** option. This option will make the driver check if the collection exists and issue an error if it does not.

db.createCollection('test', function(err, collection) {});

This command will create the collection on the Mongo DB database before returning the collection object. If the collection already exists it will ignore the creation of the collection.

db.createCollection('test', {strict:true}, function(err, collection) {});

El {strict: true} opción hará que el método devuelve un error si ya existe la colección.

Con una conexión db abierta y una colección definida estamos dispuestos a hacer alguna operación CRUD sobre los datos.