Perl 6 MongoDB driver

Marcel Timmerman < mt1957@gmail.com >

Abstract

MongoDB is a *Non SQL* database which uses *Binary JSON (BSON)* to store and load information in a database. With the mongodb package a shell program called mongo is available to give instructions to a mongodb server.

To work with data on the server from within a program a driver is needed. There are drivers for many program languages. This document describes a driver for the Perl6 language. In the perl6 ecosystem, which might grow into a cpan like system later, there are two packages needed to work with the driver. These are *MongoDB* and BSON. BSON is automatically installed with other necessary modules.

The latest version of this document is generated on date2018-05-08

Table of Contents

1. Introduction

The purpose of this document is to show how things are accomplished in this driver in the light of the MongoDB developer documents and how to work with the perl6 mongodb driver.

However, this document will not tell you how to design your database among other things. There are plenty of good books and documents out there, not to mention, the mongodb website.

There are quite a few modules written to perform the tasks at hand but not all modules will be explained here because many of them are modules defining classes to be used in the background and are not used by applications directly.

Furthermore, this document is not a reference. There are other documents for that, written to document the attributes, (sub)methods and subs in a class. There will be a list of references at the end of the document.

This document assumes that the reader is aware of at least the basics of the mongodb database and what one can do with it. Also some perl 6 knowledge will be necessary.

As a last remark, the driver is still in development. Although many parts are accomplished, some parts still need to be implemented like authentication agains kerberos or LDAP. Furthermore, there are some improvements needed to speedup the operations.

The following sections will be explained:

- *Implementation*.
  - *Server states*.
  - *Topology*. The topology of a client is defined by the set of servers and their states.
  - *Round trip time*.
  - *Read concern*.
  - *Write concern*.
  - *URI*. The URI tells the software how to connect and select the proper server.
  - *Server selection process*. This is the process of finding a server depending on the server state and the topology.
- *Modules and classes*. There are many more classes but the classes named below are the ones which will be used by you.
  - *BSON::Document*. This is the basic vehicle to insert, update retrieve and send commands to the database server. In this section there is an explanation of the supported types as well as different ways to make requests. Some detailed perl6 is necessary to understand mistakes often made when creating the data structures.
  - *MongoDB*. Here a lot of the programs constants, enumerations and subtypes are defined as well as the import of logging subs.
  - *MongoDB::Client*. This module is the starting point of all applications which need access to a mongodb database server. The client receives the uri from the user of the module to access one or more servers.
  - *MongoDB::Collection*. The Collection module is used to search through the collection data of the database.
  - *MongoDB::Cursor*. The Cursor is used to process the found data returned from the search.
  - *MongoDB::Database*. The Database is the most used module to send commands to the server.
- *Programs*. There are also a few programs in this package to help you with a few administrative tasks;
  - Starting and stopping of servers
  - Add, remove and modify accounts
  - Creating a replicaset
- *MongoDB Servers*.
- *Examples*.
- *Things to add to this package*. The program is far from complete. The most important items to add are the following;

- Secure connection to the server. This is at least needed when authentication is performed. If not, the username/password information is visible when you know where to look.
- Higher level commands. There is a need for some convenience methods like insert, update, remove etcetera. In the past there were methods like these but I have removed them for two reasons: 1) the methods were based on the wire protocol which will be obsoleted partly in the future by MongoDB and 2) MongoDB has replaced them by operations all executable using the MongoDB runCommand function and therefore by the run-command method from the MongoDB::Database class.
-

2. Implementation

A short description of what happens when a user instantiates a MongoDB::Client object providing it with a URI. The Client object parses the URI for server names. These are used by the Client to instantiate a MongoDB::Server object. The server object instantiates a MongoDB::Monitor object to keep contact with the server and to process any changes when they appear. The Monitor object informs the Server object to set the Server state. The Client object will then use the Server states to define the topology of that group of servers.

Conflicts can happen when servers defined in the URI not belong together in a group. For example a replica server and a standalone server can not be taken together in one URI because both servers can accept commands to change data. The Client object cannot guess to which of the servers the command must be send to. In such cases the topology is set to *MongoDB::TT-Unknown* which will block any operation.

2.1. Server states

Server states are situations in which a server can be. The server is queried regularly with a so called *ismaster* command. The result of this command is processed to see what kind of server we are dealing with. Based on these results the *MongoDB::Server* object sets a state which is displayed in the table below.

Table 1. Server states depending on isMaster outcome

Server state	isMaster command result
SS-Unknown	a) Initial state before processing the response. b) After a network error or failed ismaster call. c) "ok: 1" not in ismaster response.
SS-Standalone	There is no key/value pair "msg: isdbgrid", there is no setName, and there is no "isreplicaset: true" key/value pair.
SS-Mongos	Key/value pair "msg: isdbgrid" is found in the Monitor response.
SS-PossiblePrimary	Not yet processed, but another secondary server member thinks it is the primary.
SS-RSPrimary	Key/value "ismaster: true" and "setName" is found in the Monitor response.
SS-RSSecondary	"secondary: true", "setName" in response.
SS-RSArbiter	"arbiterOnly: true", "setName" in response.
SS-RSOther	"setName" in response, "hidden: true" or not primary, secondary, nor arbiter. E.g. starting up or recovering.
SS-RSGhost	"isreplicaset: true" in response. E.g. briefly during server startup, in an uninitialized replica set, or when the server is shunned (removed from the replica set config).

2.2. Topology

Table 2. Topology controlled by server states

Topology type	Server states
TT-Unknown	When a deployment has this topology type, no servers are suitable for read or write operations. These are servers which did not respond on initial connection or threw an exception because of e.g. a DNS lookup failure. All server states of these servers herded by the Client object is SS-Unknown.
TT-Single	A deployment of topology type TT-Single contains only a single server which can have any state except SS-Unknown. This topology type signifies a direct connection intended to receive all read and write operations.
TT-Sharded	A deployment of topology type TT-Sharded contains one or more servers of type SS-Mongos or SS-Unknown of at least one is SS-Mongos.
TT-ReplicaSet-NoPrimary	A deployment with this topology type can have a mix of server types: SS-RSSecondary, SS-RSArbiter, SS-RSOther, SS-RSGhost, SS-Unknown or SS-PossiblePrimary.
TT-ReplicaSet-WithPrimary	A deployment with this topology type can have a mix of server types: SS-RSPrimary, SS-RSSecondary, SS-RSArbiter, SS-RSOther, SS-RSGhost, SS-Unknown or SS-PossiblePrimary.

2.3. Round Trip Time

2.4. Read concern

2.5. Write concern

2.6. URI

Table 3. Implemented uri connection options

Option	Description
replicaSet	Specifies the name of the replica set, if the mongod is a member of a replica set. When connecting to a replica set it is important to give a seed list of at least two mongod instances. If you only provide the connection point of a single mongod instance, and omit the replicaSet, the client will create a standalone connection.

2.7. Server selection

- Record the server selection start time
- If the topology wire version is invalid, raise an error
- Find suitable servers by topology type and operation type
- If there are any suitable servers, choose one at random from those within the latency window and return it; otherwise, continue to step
- Request an immediate topology check, then block the server selection thread until the topology changes or until the server selection timeout has elapsed
- If more than serverSelectionTimeoutMS milliseconds have elapsed since the selection start time, raise a server selection error
- Goto Step

3. Modules and classes

3.1. MongoDB

3.2. MongoDB::Client

3.2.1. Making a connection

3.3. BSON::Document

3.4. MongoDB::Database

3.4.1. run-command()

3.5. MongoDB::Collection

3.5.1. find()

3.6. MongoDB::Cursor

3.6.1. fetch()

3.6.2. iterating over documents

3.7. MongoDB::Server

3.8. MongoDB::Server::Control

4. BSON

4.1. Supported types

Table 4. Supported types of the BSON package

BSON	Perl6	Description
Double	Num	An eight byte floating point number. The perl6 type choosen is a 'Num' which stores a floating-point number. On most platforms, it's an IEEE 754 64-bit floating point number, aka "double precision" (From perl 6 doc). The 'Rat' is not choosen because it can not be converted back the way it was thereby loosing accuracy.
String	Str	A normal string type.
Document	BSON::Document	As the document itself a subdocument is also a BSON::Document. Hashes are refused because the keys are not necessary kept in the same order as is stored by the user. This is important when searches are done. The seach query is also encoded using the BSON::Document and on the server not decoded. So the query is matched against binary data which is ofcourse faster.
Array	Array	

BSON	Perl6	Description
Binary	Buf	The perl6 Buf type is used to express the BSON binary type. However, the BSON specification also covers for types such as Function, UUID and MD5. Furthermore user defined types can also be be specified. Ideas for this are the perl6 types Rat, Set, IntStr, Hash, List etc. Also very large or small Int values could encoded this way.
ObjectId	BSON::ObjectId	This object is generated on the server by default. However, it can be used to refer to other objects or to create the document *_id* themselves.
Boolean	Bool	
Date	DateTime	
Null	Any	Any undefined variable or Type object is used to express the Null BSON type. It will also convert to Any only. So any other used Type object is lost when decoding the document.
Javascript	BSON::Javascript	
Javascript with scope	BSON::Javascript	
32 bit int	Int	The perl6 Int type can represent integers from -∞ to +∞. The software tests the Int number if it falls in the 4 byte range. When outside that range, it tests for the 8 byte range and converts to the BSON 64 bit type. When even smaller/larger, an exception is thrown.
Timestamp	-	Not yet supported because it is for internal MongoDB use.
64 bit int	Int	See 32 bit Int.
Decimal128	-	Not yet supported.

5. MongoDB servers

5.1. Supported versions

5.2. mongod

5.3. mongos

6. Examples

6.1. Starting and stopping a server using the configuration

This method, using a configuration file, is also used to test the modules to help starting and stopping a locally installed server. There are several steps in order to configure it properly.

- *Configuration file*.
- *Server selection*.
- *Starting and stopping*.

6.1.1. Configuration file

6.1.2. Server selection

6.1.3. Starting and stopping

6.2. Making a replica server

6.2.1. Preparing

6.2.2. Initializing

6.3. Develop your own set of helper functions

7. References to books, websites, articles and pod-documents

7.1. Web Pages

MongoDB Manual covering all aspects of what is possible. Source is from MongoDB, Inc. EPub edition [http://docs.mongodb.com/master/MongoDB-manual.epub]

MongoDB Driver Glossary and References

B

Binary JSON

: BSON is a computer data interchange format used mainly as a data storage and network transfer format in the MongoDB database. See also on WikipediA [https://nl.wikipedia.org/wiki/BSON].

J

JavaScript Object Notation

: JavaScript Object Notation) is an open-standard format that uses human-readable text to transmit data objects consisting of attribute-value pairs. See also on WikipediA [https://nl.wikipedia.org/wiki/JSON].

M

MongoDB

: MongoDB (from humongous) is a free and open-source cross-platform document-oriented database program.

N

Non SQL

: A NoSQL (originally referring to "non *Structured Query Language*", "non relational" or "not only SQL" database provides a

	mechanism for storage and retrieval of data which is modeled in means other than the tabular relations used in relational databases.

S

Structured Query Language	SQL or Structured Query Language is a special-purpose domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS)

Index