class MongoDB::Client

Class to define connections to servers

Table of Contents


```
package MongoDB { class Client { ... } }
```

Synopsis

```
my MongoDB::Client $client .= new(:uri<mongodb://>);
if $client.nbr-servers {
  my MongoDB::Database $d1 = $client.database('my_db1');
  my MongoDB::Collection $c1 = $d1.collection('my_cll1');
  my MongoDB::Collection $c2 = $client.collection('my_db2.my_cll2');
}
```

Description

This class is your most often used class. It maintains the connection to the servers specified in the given uri. In the background it herds a set of MongoDB::Server objects.

Readonly attributes

read-concern

```
has BSON::Document $.read-concern;
```

The read-concern is a structure to have some control over the read operations to which server the operations are directed to. Default is an empty structure. The structure will be explained elsewhere.

found-master

```
has Bool $.found-master = False;
```

While the client is processing the given uri it will set this flag when a master server is detected.

Methods

new

```
submethod BUILD ( Str:D :$uri, BSON::Document :$read-concern, Int )
```

Create a MongoDB::Client object.

Note. It is important to keep the following in mind to prevent memory leakage. The object must be cleaned up by hand before the variable is reused. This is because the Client object creates some background processes to keep an eye on the server and to update server object states and topology.

```
my MongoDB::Client $c .= new( ... );
# ... work with object
$c.cleanup;
```

Some help is given by the object creation. When it notices that the object is defined along with some internal variables, it will destroy that object first before continuing. This also means that you must not use another object to create one!

```
my MongoDB::Client $c1, $c2;

# first time use, no leakage
$c1 .= new(...);

# In this proces $c1 will be destroyed!!
$c2 = $c1.new(...);

# This is ok however because we want to overwrite the object anyway
$c2 .= new(...);

# And this will result in memory leakage because $c2 was already defined.
$c2 = MongoDB::Client.new(...);
```

read-concern

Read concern will overwrite the default concern.

uri

Uri defines the servers and control options. The string is like a normal uri with mongodb as a protocol name. The difference however lies in the fact that more that one server can be defined. The uri definition states that at least a servername must be stated in the uri. Here in this package the absence of any name defaults to localhost. See also the MongoDB page to look for options and definition.

Example uri	Explanation
mongodb://	most simple specification, localhost using port 27017
mongodb://:65000	localhost on port 65000
mongodb://:56,:876	two servers localhost on port 56 and 876
mongodb://example.com	server example.com on port 27017
mongodb://pete:mypasswd@	server localhost:27017 on which pete must login using mypasswd
mongodb://pete:mypasswd@/mydb	same as above but login on database mydb
mongodb:///?replicaSet=myreplset	localhost:27017 must belong to a replica set named myreplset
mongodb://u1:pw1@nsa.us:666,my.datacenter.gov/nsa/?replicaSet=foryoureyesonly	User u1 with password pw1 logging in on database nsa on server nsa.us:666 and my.datacenter.gov:27017 which must both be member of a replica set named foryoureyesonly.

Note that the servers named in the uri must have something in common such as a replica set.

Servers are refused when there is some problem between them e.g. both are master servers. In such situations another Client should be created for the other server. See table below.

Next a table where some processing results are shown for uri. In the table there are short names use like n#(=digit): for normal server, r#: a replica server, R# for replica names, r1R1 server is server for replicaset R1, i# are replicaset servers which must be initialized before they become real servers, a# are arbiters and s# mongos servers. An uninitialized replicaserver (i) is neither master nor secondary. Port numbers are irrelevant here. When two servers in a replica set R1 are used, the table shows 'r1R1, r2R1, R1' and the uri could be something like 'mongodb://r1,r2/?replicaSet=R1'.

Servers in uri	Result of processing in client
n1	The server n1 will be found and accepted
n1,n2	Only one of two servers can be accepted because both might be master. Which server is accepted depends on who is fastest.
n1,R1	The server n1 will not be accepted because its not in a replicaset.
n1,r1R1	Only server n1 is accepted because no replicaset is mentioned in uri.
n1,r1R1,R1	Only server r1 is accepted.
i1R1	Server i1 accepted.
i1R1,R1	Server i1 is not accepted because its not a real replica server yet.
r1R1,r2R1,R1	Servers r1 and r2 are both accepted. There is a master and the other should be a secondary. In this case it should be possible to leave out one of the two servers because the server monitoring process would find the other servers in the replicaset.

The options which can be used in the uri are in the following tables. See also this information for more details.

Replica set options	Impl	Use
replicaSet	x	Specifies the name of the replica set, if the mongod is a member of a replica set.

Connection options	Impl	Use
ssl		0 or 1. 1 Initiate the connection with TLS/SSL. The default value is false.
connectTimeoutMS		The time in milliseconds to attempt a connection before timing out.
socketTimeoutMS		The time in milliseconds to attempt a send or receive on a socket before the attempt times out.

Connect pool options	Impl	Use
maxPoolSize		The maximum number of connections in the connection pool. The default value is 100.
minPoolSize		The minimum number of connections in the connection pool. The default value is 0.
maxIdleTimeMS		The maximum number of milliseconds that a connection can remain idle in the pool before being removed and closed.
waitQueueMultiple		A number that the driver multiples the maxPoolSize value to, to provide the maximum number of threads allowed to wait for a connection to become available from the pool.
waitQueueTimeoutMS		The maximum time in milliseconds that a thread can wait for a connection to become available. For default values, see the MongoDB Drivers and Client Libraries documentation.

Write concern options	Impl	Use
w		Corresponds to the write concern w Option. The w option requests acknowledgement that the write operation has propagated to a specified number of mongod instances or to mongod instances with specified tags. You can specify a number, the string majority, or a tag set.
wtimeoutMS		Corresponds to the write concern wtimeout. wtimeoutMS specifies a time limit, in milliseconds, for the write concern. When wtimeoutMS is 0, write operations will never time out.
journal		Corresponds to the write concern j Option option. The journal option requests acknowledgement from MongoDB that the write operation has been written to the journal

Read concern options	Impl	Use
readConcernLevel		The level of isolation. Accepts either "local" or "majority".

Read preference options	Impl	Use
readPreference		Specifies the replica set read preference for this connection. The read preference values are the following: primary, primaryPreferred, secondary, secondaryPreferred, nearest
readPreferenceTags		Specifies a tag set as a comma-separated list of colon-separated key-value pairs

Authentication options	Impl	Use
authSource		Specify the database name associated with the user credentials, if the users collection do not exist in the database where the client is connecting. authSource defaults to the database specified in the connection string.
authMechanism	part	Specify the authentication mechanism that MongoDB will use to authenticate the connection. Possible values include: SCRAM-SHA-1, MONGODB-CR, MONGODB-X509, GSSAPI (Kerberos), PLAIN (LDAP SASL)
gssapiServiceName		Set the Kerberos service name when connecting to Kerberized MongoDB instances. This value must match the service name set on MongoDB instances.

nbr-servers

```
method nbr-servers ( --> Int )
```

Return number of servers found processing the uri in new(). When called directly after new() it may not have the proper count yet caused by delays in processing especially when processing replicasets.

server-status

```
method server-status ( Str:D $server-name --> ServerStatus )
```

Return the status of some server. See MongoDB for the defined values.

client-topology

```
method client-topology ( --> TopologyType ) {
```

Return the topology of the set of servers represents. See MongoDB for the defined values. For the moment it must be implemented yet.

select-server

```
multi method select-server (
  BSON::Document:D :$read-concern!
  --> MongoDB::Server
)

multi method select-server (
  ServerStatus:D :$needed-state!,
  Int :$check-cycles is copy = -1
  --> MongoDB::Server
)

multi method select-server (
  Int :$check-cycles is copy = -1
  --> MongoDB::Server
)
```

Select a server for operations. It returns a Server object. In single server setups it is always the server you want to have. When however selecting a server from a replicaset the server is selected according to several rules such as read-concern, operation type (read or write) and round trip time to the server. When read-concern is not defined, the data is taken from this Clients read-concern. This method is used internally and of no concern of the user.

database

```
method database (
  Str:D $name, BSON::Document :$read-concern
  --> MongoDB::Database
)
```

Create a Database object. In mongodb a database and its collections are only created when data is written in a collection.

The read-concern when defined will overide the one of the Client. If not defined, the structure of the client is taken.

collection

```
method collection (
  Str:D $full-collection-name, BSON::Document :$read-concern
  --> MongoDB::Collection
)
```

A shortcut to define a database and collection at once. The names for the database and collection are given in the string full-collection-name. This is a string of two names separated by a dot '.'.

When the read-concern is defined it overides the one from Client. If not defined, the structure of the client is taken.

cleanup

method cleanup ( )

Stop any background work on the Server object as well as the Monitor object. Cleanup structures so the object can be cleaned further by the GC later.