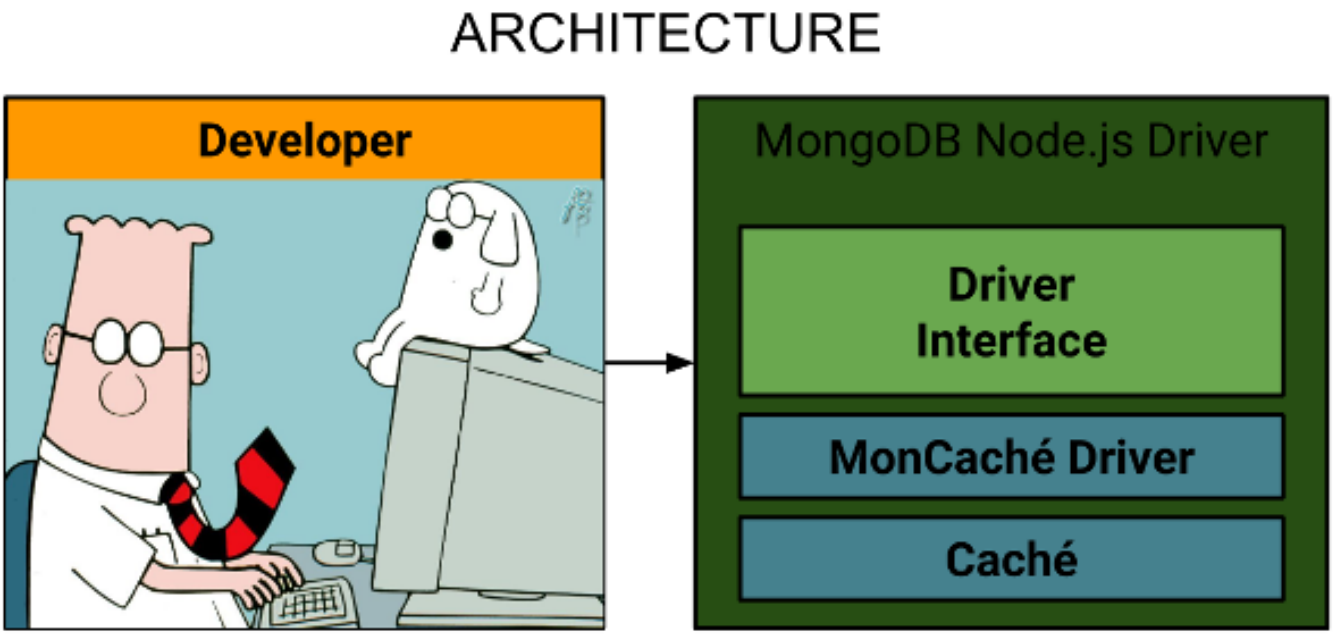
Article
Maks Atygaev · Jul 18, 2017 7m read

MonCaché - Caché as MongoDB

MonCaché — MongoDB API implementation based on InterSystems Caché



Disclaimer: This article reflects author's private opinion and has no relation to the official position of InterSystems.

IDEA

The idea of the project is to implement basic MongoDB (v2.4.9) API features for searching, saving, updating and deleting documents in a way that will allow the use of InterSystems Caché instead of MongoDB without changing the code on the client side.

MOTIVATION

Perhaps, if we take an interface based on MongoDB and use InterSystems Caché for data storage, we may see a performance boost. The project was started as research project in my bachelor's diploma.

Hey, why not?! ¯_(ツ)_/¯

LIMITATIONS

A few simplifications were made in the course of this research project:

- only primitive data types are used: *null*, *boolean*, *number*, *string*, *array*, *object*, *ObjectId*;
- the client side code works with MongoDB via a MongoDB driver;
- the client side code uses the MongoDB Node.js driver;

- the client side code uses only basic functions of the MongoDB API:
  - *find*, *findOne* — document search;
  - *save*, *insert* — document saving;
  - *update* — document updates;
  - *remove* — document removal;
  - *count* — document counting.

IMPLEMENTATION

The task was eventually broken into the following subtasks:

- recreate the interface of the MongoDB Node.js driver for selected basic functions;
- implement this interface using InterSystems Caché for data storage:
  - design a database representation scheme in Caché;
  - design a collection representation scheme in Caché;
  - design a document representation scheme in Caché;
  - design a scheme for interacting with Caché using Node.js;
  - implement the designed schemes and test them a little. :)

IMPLEMENTATION DETAILS

The first subtask was not a problem, so I will cut to the chase and describe the interface implementation part.

MongoDB defines a database as a physical container for collections. A collection as a set of documents. A document as a set of data. A document is like a JSON document, but with a larger number of allowed types - BSON.

In InterSystems Caché, all data are stored in globals. For simplicity, you can think of them as hierarchical data structures.

In this project, all data will be stored in a single global - ^MonCache.

Therefore, we need to design a representation scheme for the database, collections and documents using hierarchical data structures.

Database representation scheme in Caché

Global layout I ' ve implemented just one of many potential layouts where each layout has different benefits and limitations.

In MongoDB, there can be several databases in just one instance, which means that we need to design a representation scheme that will allow us to store several isolated databases. It should be noted that MongoDB supports databases that contain no collections (I will refer to them as " empty " databases).

I chose the simplest and the most obvious solution for this problem. Databases are represented as a first-level node in the ^MonCache global. Besides, such a node gets a "" value to enable the support of " empty " databases. The thing is, if you don ' t do it and just add child nodes, their removal will also remove the parent node (that ' s just the way globals work).

Therefore, each database is represented in Caché in the following form:
^MonCache(<db>) = ""

For example, representation of a " mydatabase " database will look as follows:
^MonCache("mydatabase") = ""

Collection representation scheme in Caché

MongoDB defines a collection as a database element. All collections in a single database have a unique name which can be used for accurate collection identification. This fact helped me find a simple way of representing collections in a global, and namely use second-level nodes. Now we need to solve two little problems. The first one is that collections, just like databases, can be empty. The second one is that a collection is a set of documents. And all of these documents have to be isolated from each other. To be honest, I couldn ' t come up with a better idea than having a counter, something like an auto-increment value, as a collection node value. All documents have a unique number. When a new document is inserted, a new node with a name equal to the current counter value is created, and the counter value is increased by 1.

Therefore, each Caché collection is represented in the following form:
^MonCache(<db>) = ""
^MonCache(<db>, <collection>) = 0

For example, the ***"mycollection"*** collection in the ***"mydatabase"*** database will be represented like this:
^MonCache("mydatabase") = ""
^MonCache("mydatabase", "mycollection") = 0

Document representation scheme in Caché

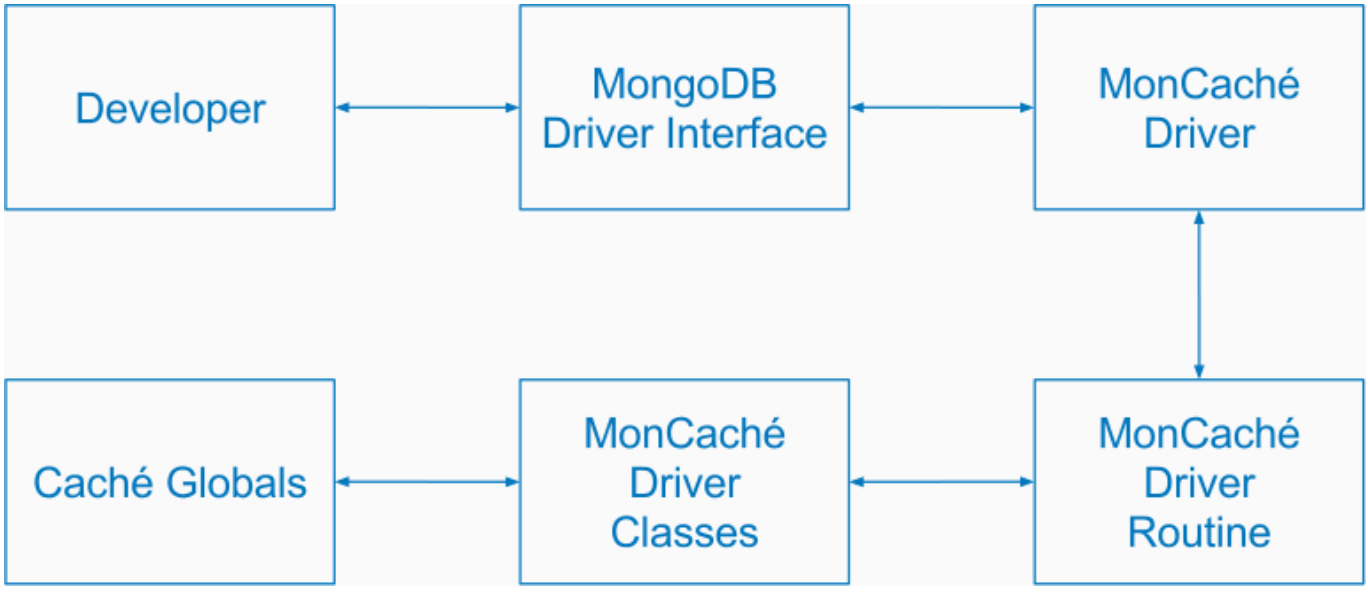
In this project, a document is a JSON document extended by an additional type - ObjectId. I had to design a document representation scheme for hierarchical data structures. That ' s where I faced a few surprises. First of all, I couldn ' t use the " native " null in Caché, since it wasn ' t supported. Another thing is that boolean values are implemented via 0 and 1 constants. In other words, 1 is true, 0 is false. The most expected problem was that I had to come up with a way of storing ObjectId. In the end, all of these problems were successfully solved in the most straightforward manner, or so I thought. In the article below, I will look at each data type and its representation.

Caché interaction scheme

The choice of the Node.js driver seemed to be a logical and simple decision for working with InterSystems Caché (you can find other drivers for interaction with Caché on the documentation website). However, note that the capabilities of the driver will not be sufficient. I wanted to perform several inserts with a single transaction. That is why I decided to develop a set of Caché ObjectScript classes that were used for emulating the MongoDB API on the Caché side.

The Caché Node.js driver could not access Caché classes, but could make program calls from Caché. This fact resulted in the creation of a small tool - a kind of bridge between the driver and Caché classes.

In the end, the scheme looked like this:



While working on the project, I created a special format that I called NSNJSON (Not So Normal JSON) that allowed me to " smuggle " ObjectId, null, true and false values through the driver to Caché. More information about this format can be found on a corresponding page on GitHub — NSNJSON.

MONCACHÉ CAPABILITIES

The following criteria are available for document search:

- ***$eq*** — equivalent;
- ***$ne*** — not equivalent;
- ***$not*** — negation;
- ***$lt*** — less than;
- ***$gt*** — more than;
- ***$exists*** — existence.

The following operators are available for document update operations:

- ***$set*** — value setting;
- ***$inc*** — value increment by a specified number;
- ***$mul*** — value multiplication by a specified number;
- ***$unset*** — value removal;
- ***$rename*** — value renaming.

EXAMPLE

I took this code from the official driver ' s page and modified it a bit.

```
var insertDocuments = function(db, callback) {
 var collection = db.collection('documents');
 collection.insertOne({ site: 'Habrahabr.ru', topic: 276391 }, function(err, result) {
     assert.equal(err, null);
     console.log("Inserted 1 document into the document collection");
     callback(result);
 });
}
var MongoClient = require('mongodb').MongoClient
  , assert = require('assert');
var url = 'mongodb://localhost:27017/myproject';
MongoClient.connect(url, function(err, db) {
    assert.equal(null, err);
    console.log("Successfully connected to the server");
    insertDocument(db, function() {
        db.close();
  });
});
```

This code can be easily modified to make it compatible with MonCaché!

All we need to do it to change the driver!

```
// var MongoClient = require('mongodb').MongoClient
var MongoClient = require('moncache-driver').MongoClient
```

After this code is executed, the ^MonCache global will look like this:

```
^MonCache("myproject","documents")=1
^MonCache("myproject","documents",1,"id","t")="objectid"
```

```
^MonCache("myproject","documents",1,"id","v")="b18cd934860c8b26be50ba34"
^MonCache("myproject","documents",1,"site","t")="string"
^MonCache("myproject","documents",1,"site","v")="Habrahabr.ru"
^MonCache("myproject","documents",1,"topic","t")="number"
^MonCache("myproject","documents",1,"topic","v")=267391
```

DEMO

Apart from everything else, we launched a small demo application (source code), also implemented with Node.js to demonstrate the change of the driver from MongoDB Node.js to MonCaché Node.js without restarting the server and changing the source code. The application is a tiny tool for performing CRUD operations on products and offices, as well as an interface for changing the configuration (driver change).

The server allows you to create products and offices that are saved to a storage selected in the configuration (Caché or MongoDB).

The "Orders" tab contains a list of orders. I have created the records, but the form isn ' t completed yet. You are welcome to help the project (source code).

You can change the configuration on the "Configuration" page. There are two buttons on this page: MongoDB and MonCache. You can select a desired configuration by clicking a corresponding button. When the configuration is changed, the client application reconnects to the data source (an abstraction separating the application from the actually used driver).

CONCLUSION

To draw the bottom line, let me answer the main question. Yes! We did manage to achieve a performance boost for basic operations.

The MonCaché project is published on GitHub and is available under an MIT license.

SHORT MANUAL

- Install Caché
- Load the necessary MonCaché components to Caché
- Create a MONCACHE area in Caché
- In Caché, create a user named ***"moncache"*** with the password ***"ehcacnom"*** (reversed ***"moncache"***)
- Create an environment variable ***MONCACHE_USERNAME*** = ***moncache***
- Create an environment variable ***MONCACHE_PASSWORD*** = ***ehcacnom***
- Create an environment variable ***MONCACHE_NAMESPACE*** = ***MONCACHE***
- In your project, change the dependency from ***'mongodb'*** to ***'moncache-driver'***
- Start your project! :-)

INTERSYSTEMS ACADEMIC PROGRAM

If you are interested in starting your own research project based on InterSystems technologies, you can visit a specialized site dedicated to InterSystems academic programs.

#API #Data Model #Databases #Document Data Model (NoSQL) #JSON #Node.js #Object Data Model #Caché