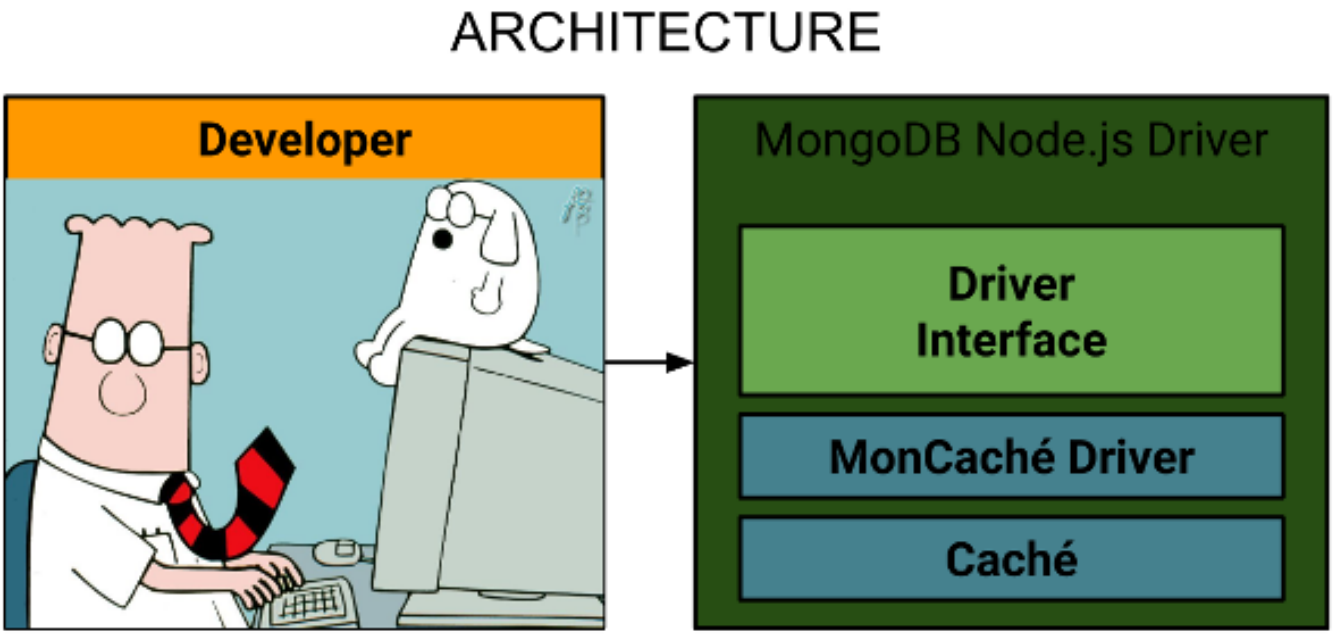
Article
Ricardo Paiva · May 22, 2020 9m read

MonCaché - Caché como parte de MongoDB

¡Hola desarrollador!

En este articulo repasaremos una publicación original de Maks Atygaev sobre la Implementación de una API de MongoDB, basada en InterSystems Caché - MonCaché.



Descargo de responsabilidad: En este artículo se muestra la opinión personal del autor y no tiene ninguna relación con opinión oficial de InterSystems.

Idea

La idea del proyecto es implementar las características básicas de la API en MongoDB (v2.4.9) , con la finalidad de buscar, guardar, actualizar y eliminar documentos de una manera que permita el uso de InterSystems Caché, en lugar de MongoDB, sin cambiar el código del lado del cliente.

Motivación

Probablemente, si tomamos una interfaz basada en MongoDB y utilizamos InterSystems Caché para el almacenamiento de los datos, podríamos ver un aumento en el rendimiento. El proyecto comenzó como un proyecto de investigación durante mis estudios universitarios.

Hey, ¡¿por qué no?! ¯_(ツ)_/¯

Limitaciones

En el transcurso de este proyecto de investigación se hicieron algunas simplificaciones:

- solamente se utilizaron tipos de datos primitivos: *nulos*, *booleanos*, *numéricos*, *cadenas de texto*, *conjuntos*, *objetos*, ObjectId;
- el código del lado del cliente funciona con MongoDB mediante un controlador de MongoDB
- el código del lado del cliente utiliza el controlador MongoDB Node.js
- el código del lado del cliente solamente utiliza las funciones básicas de la API en MongoDB:
    - find, findOne — para buscar los documentos;
    - save, insert — para guardar los documentos;
    - update — para actualizar los documentos;
    - remove — para eliminar los documentos;
    - count — para contar los documentos.

Implementación

La tarea finalmente se dividió en las siguientes subtareas:

- recrear la interfaz del controlador MongoDB Node.js para las funciones básicas que se seleccionaron,
- implementar esta interfaz utilizando InterSystems Caché para el almacenamiento de los datos:
    - diseñar un esquema que represente las bases de datos en Caché,
    - diseñar un esquema que represente a las colecciones en Caché,
    - diseñar un esquema que represente a los documentos en Caché,
    - diseñar un esquema que permita la interacción con Caché mediante Node.js,
    - implementar los esquemas que se diseñaron y probarlos brevemente. :)

Detalles sobre la implementación

Realizar la primera subtarea no fue complicado, de modo que iré al grano y describiré la parte de la implementación de la interfaz.

MongoDB define una base datos como un contenedor físico para las colecciones. Una colección como un conjunto de documentos. Un documento como un conjunto de datos. Un documento es similar a los documentos en JSON, pero permite un mayor número de clases - BSON.

En InterSystems Caché todos los datos se almacenan en los globales. Por simplicidad, puede pensarse en ellos como estructuras de datos ordenadas de manera jerárquica.

En este proyecto, todos los datos se almacenarán en un único global: ^MonCache.

Por lo tanto, necesitamos diseñar un esquema que represente a las bases de datos, colecciones y documentos utilizando estructuras de datos ordenadas de manera jerárquica.

Esquema para representar las bases de datos en Caché

Para diseñar el global, únicamente implementé uno de entre los muchos diseños posibles, ya que cada diseño tiene diferentes beneficios y limitaciones.

En MongoDB puede haber varias bases de datos en una sola instancia, lo cual significa que necesitamos diseñar un esquema para representarlas que nos permita almacenar varias bases de datos por separado. Es importante señalar que MongoDB es compatible con bases de datos que no contienen colecciones (me referiré a ellas como bases de datos “ vacías ”).

Elegí la forma más sencilla y obvia para solucionar este problema. Las bases de datos se representan como un nodo de primer nivel en el global ^MonCache. Además, dicho nodo obtiene un valor "" para permitir que

sea compatible con las bases de datos " vacías " . La cosa es que, si no hace este paso y solamente agrega nodos hijo, cuando los elimine también eliminará al nodo padre (esa es la forma en que funcionan los globales).

Por lo tanto, cada base de datos se representa en Caché de la siguiente manera:

```
^MonCache(<db>) = ""
```

Por ejemplo, la representación de la base de datos " mydatabase " se verá de la siguiente manera:

```
^MonCache("mydatabase") = ""
```

Esquema para representar las colecciones en Caché

MongoDB define a una colección como uno de los elementos en una base de datos. Todas las colecciones que estén en una sola base de datos tienen un nombre único, el cual puede utilizarse para identificar de manera precisa a la colección. Este hecho fue lo que me ayudó a encontrar una manera sencilla para representar a las colecciones en un global y, en particular, a utilizar nodos de segundo nivel. Ahora debemos resolver dos pequeños problemas. El primero es que las colecciones, al igual que las bases de datos, pueden estar vacías. El segundo es que una colección es un conjunto de documentos. Y todos estos documentos deben estar separados unos de otros. Para ser honesto, no se me ocurrió una mejor idea que tener un contador, algo similar a un valor que se incrementará de manera automática, como un valor en la colección de nodos. Todos los documentos tienen un número único. Cuando se inserta un nuevo documento, se crea un nuevo nodo con el mismo nombre que tiene el valor del contador actual, y el valor del contador se incrementa en 1.

Por lo tanto, cada una de las colecciones en Caché se representa de la siguiente manera:

```
^MonCache(<db>) = ""
^MonCache(<db>, <collection>) = 0
```

Por ejemplo, la colección *" mycollection "* que se encuentra en la base de datos *" mydatabase "* se representará de la siguiente manera:

```
^MonCache("mydatabase") = ""
^MonCache("mydatabase", "mycollection") = 0
```

Esquema para representar a los documentos en Caché

En este proyecto, un documento es un documento en JSON, que se amplia mediante una clase adicional: ObjectId. Tuve que diseñar un esquema que representara a los documentos para estructuras de datos ordenadas de manera jerárquica. Ahí fue donde me enfrenté a algunas sorpresas. En primer lugar, no podía usar el valor nulo " nativo " en Caché, ya que no era compatible. Otra cosa es que los valores booleanos se implementan mediante las constantes 0 y 1. En otras palabras, 1 significa verdadero y 0 significa falso. El problema más grande fue que debía encontrar una forma para almacenar a ObjectId. Al final, todos estos problemas se resolvieron exitosamente y de la manera más sencilla, o al menos eso creía yo. En el siguiente artículo, examinaré cada uno de los tipos de datos y cómo se representan.

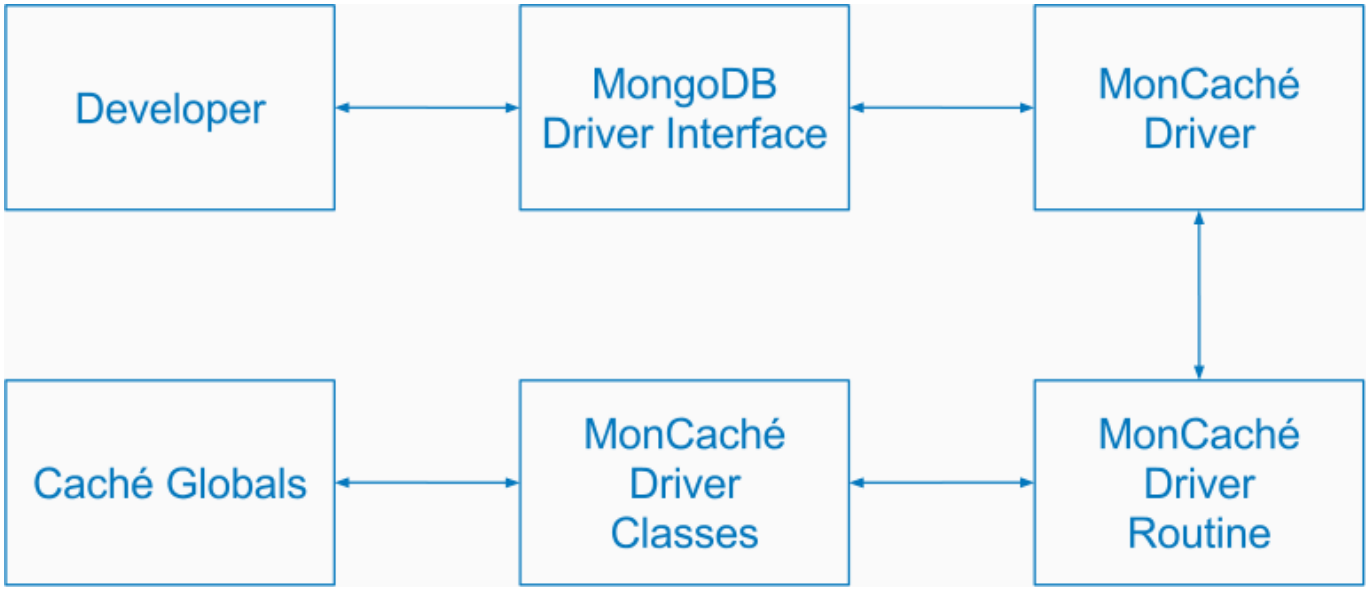
Esquema para interactuar con Caché

La elección del controlador Node.js pareció ser una decisión lógica y sencilla para trabajar con InterSystems Caché (puede encontrar otros controladores para interactuar con Caché en los documentos de apoyo que se encuentran disponibles en el sitio web). Sin embargo, tenga en cuenta que la capacidad del controlador puede no ser suficiente. Quería realizar varias inserciones con una sola transacción. Es por eso que decidí desarrollar un conjunto de clases en Caché ObjectScript, que se utilizaron para emular la API de MongoDB en el lado de Caché.

El controlador de Caché, Node.js, no podía acceder a las clases en Caché, pero podía llamar al programa

desde Caché. Este hecho resultó en la creación de una pequeña herramienta: Una especie de puente entre el controlador y las clases en Caché.

Al final, el esquema se veía de la siguiente manera:



Mientras trabajaba en el proyecto, establecí un formato especial que llamé NSNJSON (Not So Normal JSON) que me permitió "pasar de contrabando" ObjectId, valores nulos, verdaderos y falsos a través del controlador a Caché. Puede encontrar más información sobre este formato en la página correspondiente de GitHub — NSNJSON.

Funcionalidad de MonCaché

Para la búsqueda de documentos están disponibles los siguientes criterios:

- $eq — equivalencia;
- $ne — no equivalencia;
- $not — negación;
- $lt — menor que;
- $gt — mayor que;
- $exists — existe.

Para las operaciones de actualización de documentos están disponibles los siguientes operadores:

- $set — configurar un valor;
- $inc — incrementar un valor por un número específico;
- $mul — multiplicar un valor por un número específico;
- $unset — eliminar un valor;
- $rename — cambiar el nombre de un valor.

Ejemplo

Tomé este código de la página oficial del controlador y lo modifiqué un poco.

```
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();
  });
});
```

¡Este código se puede modificar fácilmente para hacerlo compatible con MonCaché!

¡Lo único que necesitamos para hacerlo es cambiar el driver!

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

Después de ejecutar el código, el global de ^MonCache se verá de la siguiente manera:

```
^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
```

Demostración

Aparte de todo lo demás, lanzamos una pequeña demostración de la aplicación (código fuente), la cual también se implementó con Node.js para demostrar cómo se realiza el cambio del controlador desde MongoDB Node.js hacia MonCaché Node.js, sin que fuera necesario reiniciar el servidor y cambiar el código fuente. La aplicación es una pequeña herramienta para realizar operaciones CRUD en productos y oficinas, así como una interfaz para cambiar la configuración (modificar el controlador).

El servidor permite crear productos y funciones que se guardan para después almacenarse en la configuración que se seleccionó (Caché o MongoDB).

La pestaña "Órdenes" contiene una lista de las órdenes. Ya establecí los registros, pero el formulario aún no está completo. Usted es bienvenido a colaborar en el proyecto (código fuente).

Puede cambiar la configuración desde la página "Configuración". En esta página hay dos botones: MongoDB y MonCache. Puede seleccionar la configuración que desee al hacer clic sobre el botón correspondiente. Cuando se cambia la configuración, la aplicación del cliente se conectará nuevamente a la fuente de los datos (una abstracción que separa la aplicación del controlador que se utiliza en ese momento).

Conclusión

En resumen, permítanme responder la pregunta más importante. ¡Exacto! Logramos aumentar el rendimiento

cuando se realizan las operaciones básicas.

El proyecto MonCaché se publicó en GitHub y está disponible bajo una licencia del MIT.

Guía Breve

- Instale Caché
- Cargue los componentes que sean necesarios para MonCaché en Caché
- Establezca un área en Caché para MONCACHE
- En Caché, establezca un usuario que se llame *"moncache"* con la contraseña *"ehcacnom"* (es *"moncache"* al revés)
- Establezca una variable de entorno *MONCACHE_USERNAME = moncache*
- Establezca una variable de entorno *MONCACHE_PASSWORD = ehcacnom*
- Establezca una variable de entorno *MONCACHE_NAMESPACE = MONCACHE*
- En su proyecto, modifique la dependencia de *'mongodb'* hacia *'moncache-driver'*
- ¡Inicie su proyecto! :-)

Programa Académico de InterSystems

Si está interesado en iniciar su propio proyecto de investigación basado en las tecnologías de InterSystems, puede visitar una página especializada que se dedica a brindar información sobre los programas académicos de InterSystems.

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