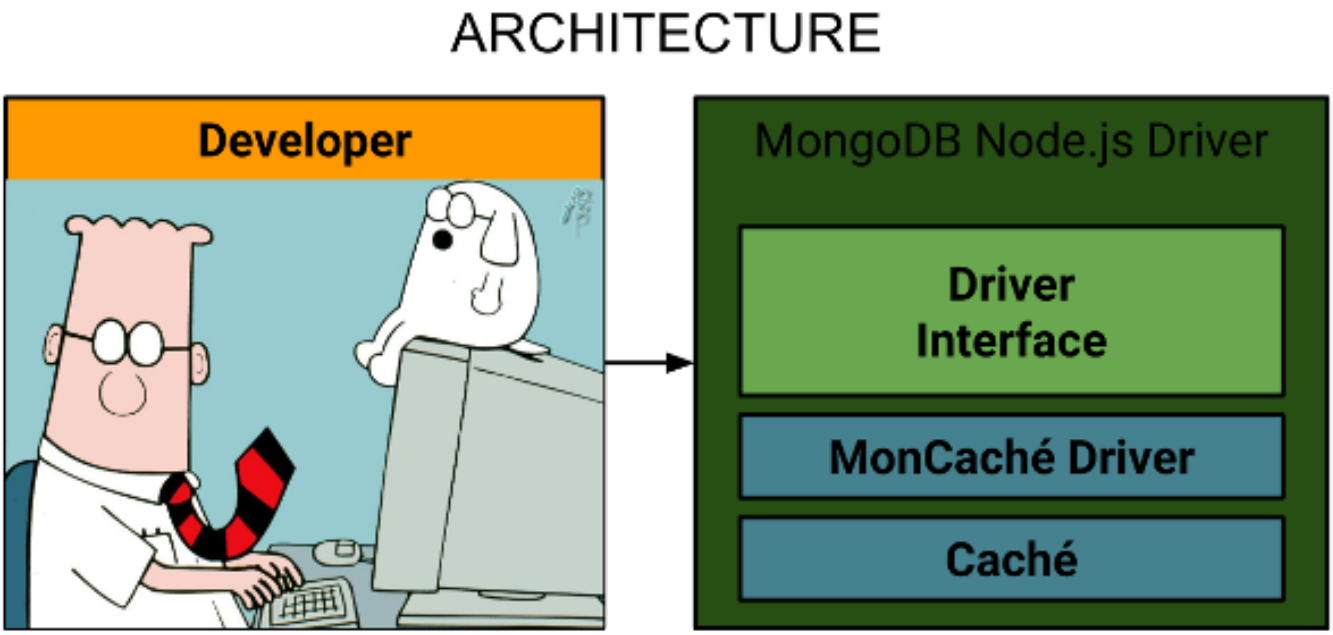
記事
Toshihiko Minamoto · 2021年6月15日 11m read

# MonCaché - MongoDB としての Caché

**MonCaché — InterSystems Caché での MongoDB API 実装**




**免責事項：この記事は筆者の私見を反映したものであり、InterSystemsの公式見解とは関係ありません。**

## 構想

プロジェクトの構想は、クライアント側のコードを変更せずに MongoDB の代わりに InterSystems Caché を使用できるように、ドキュメントを検索、保存、更新、および削除するための基本的な MongoDB（v2.4.9） API 機能を実装するところにあります。

## 動機

おそらく、MongoDB に基づくインターフェースを使って、データストレージに InterSystems Caché を使用すると、パフォーマンスが向上するのではないか。
このプロジェクトは、学士号を取得するための研究プロジェクトとして始まりました。

ダメなわけないよね？！ ¯_(ツ)_/¯

## 制限

この研究プロジェクトを進める過程で、いくつかの簡略化が行われました。

- プリミティブ型のデータのみを使用する: *null*、*boolean*、*number*、*string*、*array*、*object*、*ObjectId*

- クライアント側のコードは MongoDB ドライバーを使って MongoDB と連携する。
- クライアント側のコードでは、MongoDB Node.js ドライバーを使用する。
- クライアント側のコードでは、MongoDB API の基本的な機能のみを使用する:
    - *find*、*findOne* — ドキュメントの検索
    - *save*、*insert* — ドキュメントの保存
    - *update* — ドキュメントの更新
    - *remove* — ドキュメントの削除
    - *count* — ドキュメント数のカウント

# 実装

タスクは最終的に次のサブタスクに分割されました。

- 選択した基本機能ように、MongoDB Node.js ドライバーのインターフェースを再作成する。
- このインターフェースを InterSystems Caché を使用してデータストレージ用に実装する。
    - Caché でデータベースの表現スキームを設計する。
    - Caché でコレクションの表現スキームを設計する。
    - Caché でドキュメントの表現スキームを設計する。
    - Node.js を使って Caché と対話するためのスキームを設計する。
    - 設計したスキームを実装して、少しテストする。 :)

# 実装の詳細

最初のサブタスクは問題ではなかったので、中間の説明を省略して、インターフェースの実装部分を説明します。

MongoDB は、データベースをコレクションの物理的なコンテナとして定義します。
コレクションはドキュメントのセットで、ドキュメントはデータのセットです。 ドキュメントは JSON ドキュメントのようですが、大量の許容型を持っています。つまり BSON です。

InterSystems Caché では、すべてのデータはグローバルに保存されます。
簡単に言えば、階層データ構造として考えることができます。

このプロジェクトでは、すべてのデータは単一のグローバル(^MonCache )に保存されます。

そのため、階層データ構造を使用して、データベース、コレクション、およびドキュメントの表現スキームを設計する必要があります。

## Caché におけるデータベースの表現スキーム

**実装したグローバルレイアウトは、数多くの潜在的なレイアウトの 1 つに過ぎず、これらのレイアウトには、それぞれにメリットと制限があります。**

MongoDB では、1 つのインスタンスに複数のデータベースが存在する可能性があるため、複数の分離されたデータベースを格納できるようにする表現スキームを設計する必要があります。MongoDBはコレクションをまったく含まないデータベースをサポートしていることに注意しておきましょう（これらを「空」のデータベースと呼ぶことにします）。

私はこの問題に対し、一番単純で一番明白なソリューションを選択しました。 データベースは、^MonCache グローバルの第 1 レベルノードとして表されます。
さらに、そのようなノードは、「空」のデータベースのサポートを有効にするために ”” 値を取得します。 問題は、これを行わずに子ノードを追加するだけの場合、それらを削除すると親ノードも削除されるということです（これがグローバルの動作です）。

したがって、各データベースは Caché 内で次のように表されます。
^MonCache(<db>) = ""

たとえば、「mydatabase」データベースの表現は次のようになります。
^MonCache("mydatabase") = ""

## Caché におけるコレクションの表現スキーム

MongoDBは、コレクションをデータベースの要素として定義します。単一のデータベース内にあるすべてのコレクションには、正確なコレクションの識別に使用できる一意の名前があります。
このことが、グローバルでコレクションを表現する単純な方法を見つけ、第 2 レベルのノードを使用する上で役立ちました。 次に、2 つの小さな問題を解決する必要があります。 1 つ目の問題は、コレクションがデータベースと同じように空であることができるということです。 2 つ目は、コレクションが一連のドキュメントであるということです。
そして、これらのドキュメントはすべて、互いに分離している必要があります。 正直なところ、コレクションノードの値として自動的に増分する値のようなカウンターを使う以外に良いアイデアが浮かびませんでした。
すべてのドキュメントには一意の番号があります。 新しいドキュメントが挿入されると、現在のカウンターの値と同じ名前が付いた新しいノードが作成され、カウンターの値が 1 つ増加するというアイデアです。

したがって、各 Caché コレクションは、次のように表されます。
^MonCache(<db>) = ""
^MonCache(<db>, <collection>) = 0

たとえば、「***mydatabase***」データベース内の「***mycollection***」コレクションは次のように表されます。
^MonCache("mydatabase") = ""
^MonCache("mydatabase", "mycollection") = 0

## Caché におけるドキュメントの表現スキーム

このプロジェクトでは、ドキュメントは追加の ObjectID という型で拡張された JSON ドキュメントです。
ドキュメントの表現スキームは、階層データ構造で設計する必要がありました。
いくつかの驚きに遭遇したのはここです。 まず、Caché では「ネイティブ」の null がサポートされていなかったために、使用できなかったことです。 もう 1 つは、ブール値が 0 と 1 の定数で実装されているということです。 つまり、1 が true で 0 が false ということなのです。
一番予想していた問題は、ObjectId を格納する方法を考え出す必要があるということでした。
結局、これらの問題はすべて最も簡単な方法でうまく解決されたのです。というか、解決されたと思いました。
以下では、各データ型とその表現について説明します。

## Caché のインタラクションスキーム

Node.js ドライバーの選択は、InterSystems Caché と連携する上で論理的かつ単純な決定であるように思われました（ドキュメントサイトには、Caché と対話するためのドライバーがほかにも掲載されています）。 ただし、ドライバーの機能では不十分です。
私が行いたかったのは、1回のトランザクションで複数の挿入を実行することだったので、Caché 側で MongoDB API をエミュレートするために使用される一連の Caché ObjectScript クラスを開発することにしました。

Caché Node.js ドライバーは、Caché クラスにアクセスできませんでしたが、Caché からプロフラム呼び出しを行うことができました。
このことから、小さなツールが作成されました。ドライバーと Caché クラスを繋ぐ、一種のブリッジです。

結局、スキームは次のようになりました。

プロジェクトの作業を進めながら、NSNJSON（Not So Normal JSON: あんまり普通じゃない JSON）と名付けた、ドライバーを介して ObjectId、null、true、および false を Caché に「密輸」する特別な形式を作成しました。 この形式の詳細については、GitHub の対応するページ（NSNJSON）をご覧ください。

# MONCACHÉ の機能

ドキュメント検索には、次の基準を使用できます。

- ***$eq*** — 等号
- ***$ne*** — 不等号
- ***$not*** — 否定
- ***$lt*** — より小さい（未満）
- ***$gt*** — より大きい
- ***$exists*** — 有無、存在

ドキュメントの更新操作には、次の演算子を使用できます。

- ***$set*** — 値の設定
- ***$inc*** — 指定された数による増分
- ***$mul*** — 指定された数による乗算
- ***$unset*** — 値の削除
- ***$rename*** — 値の名前変更

## 例

以下のコードは、私がドライバーの公式ページから取得して少し修正したコードです。

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

このコードは、MonCaché と互換性を持つように簡単に変更することができます。

ドライバーを変更するだけです！

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

このコードが実行されると、^MonCache グローバルは次のようになります。

```
  ^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
```

## デモ

ほかのすべてとは別に、小型のデモアプリケーション（ソースコード
）を公開しました。また、サーバーの再起動とソースコードの変更を行わないで、MongoDB Node.js から MonCaché Node.js へのドライバーの変更を実演するために、Node.js を使って実装しました。
アプリケーションは、CRUD 演算を製品やオフィスで実行するための小さなツールであり、構成を変更（ドライバーを変更）するためのインア−フェースでもあります。

サーバーでは、構成で選択されたストレージ（Caché または MongoDB）に保存される製品とオフィスを作成できます。

［Orders］タブには、注文のリストが含まれています。
レコードは作成されていますが、フォームは未完成です。
プロジェクトの援護を歓迎しています（ソースコード）。

構成は、Configuration ページで変更できます。 このページには、MongoDB と MonCache の 2 つのボタンがあります。 対応するボタンをクリックすると、希望する構成を選択できます。 構成が変更されると、クライアントアプリケーションはデータソースに再接続します（実際に使用されているドライバーからアプリケーションを分離する概念）。

## まとめ

結論を出すために、根本的な質問に答えさせてください。 そうです！
基本的な操作におけるパフォーマンスを向上させることができました。

MonCaché プロジェクトは GitHub に公開されており、MIT ラインセンスの下に提供されています。

## 簡易マニュアル

- Caché のインストール
- 必要な MonCaché コンポーネントを Caché に読み込む
- Caché で MONCACHE 領域を作成する
- Caché で、ユーザー名が「***moncache***」でパスワードが「***ehcacnom***」（「***moncache***」**の逆**）のユーザーを作成する
- 環境変数 ***MONCACHE_USERNAME*** = ***moncache*** を作成する
- 環境変数 ***MONCACHE_PASSWORD*** = ***ehcacnom*** を作成する
- 環境変数 ***MONCACHE_NAMESPACE*** = ***MONCACHE*** を作成する
- プロジェクトで、依存関係を ***'mongodb'*** から ***'moncache-driver'*** に変更する
- プロジェクトを始動！ :-)

## InterSystems 教育プログラム

InterSystems テクノロジーに基づく独自の研究プロジェクトを始めたい方は、InterSystems 教育プログラム 専用の特設サイトをご覧ください。

#API #JSON #Node.js #オブジェクトデータモデル #データベース #データモデル #ドキュメントデータモデル（NoSQL） #Caché

**ソースURL:**
https://jp.community.intersystems.com/post/moncach%C3%A9-mongodb-%E3%81%A8%E3%81%97%E3%81%A6%E3%81%AE-cach%C3%A9