

Использование MongoDB / Clojure

Москва

4 марта 2010 г.

Илья Обшадко, ENTARENA Inc.

- необходимость хранения сложных объектов
- нечеткость исходной схемы данных
- быстрая эволюция функциональных требований

- создание реляционной схемы в соответствии с объектной моделью
- хранение данных в таблицах
- использование ORM для трансляции между объектным и реляционным представлением

- документоориентированная база данных
- все объекты хранятся в своем натуральном виде
- для хранения объектов используются коллекции

- все объекты БД представляют собой объекты JSON (BSON)
- стандартные типы данных: `string`, `int`, `boolean`, `double`, `null`, `array`, `object`
- дополнительные типы данных: `object id`, `binary data`, `regexp`, `code`

- объекты группируются в коллекции /грубый аналог реляционных таблиц/
- коллекция может содержать любые объекты /schema-free/
- коллекция может быть проиндексирована по любым полям, в том числе вложенным

▸ вся работа с БД представлена как JavaScript-вызовы

```
$ bin/mongo
> use mydb
> record1 = { key1: 'value1', key2: [1, 2, 3] }
> record2 = { key1: 'value2', key2: [4, 5, 6] }
> db.mycoll.save ( record1 )
> db.mycoll.save ( record2 )
> db.mycoll.find()
{ "_id" : ObjectId("4b8eb748230f141638cae177"), "key1" : "value1", "key2" : [ 1, 2, 3 ] } { "_id" :
ObjectId("4b8eb74d230f141638cae178"), "key1" : "value2", "key2" : [ 4, 5, 6 ] }
> db.mycoll.ensureIndex({key1: 1})
> db.mycoll.find({key1:'value1'}) { "_id" : ObjectId("4b8eb748230f141638cae177"), "key1" :
"value1", "key2" : [ 1, 2, 3 ] }
> db.mycoll.update({key1:'value2'}, {'$set':{key3:'some other data'}})> db.mycoll.find() { "_id"
: ObjectId("4b8eb748230f141638cae177"), "key1" : "value1", "key2" : [ 1, 2, 3 ] } { "_id" :
ObjectId("4b8eb74d230f141638cae178"), "key1" : "value2", "key2" : [ 4, 5, 6 ], "key3" : "some
other data" }
```

- business entities в отдельных коллекциях
- detailed records внедрены в объекты
- объекты с соотношениями many-to-many - в отдельных коллекциях
- NEVER NEGLECT COMMON SENSE

- Lisp-образный язык для JVM
- языковые типы данных - строка, число, вектор, список, функция, хэш, множество
- прозрачно интегрируется с Java
- хэши можно хранить как элементы коллекций Mongo

- Java-драйвер предоставляет всю необходимую функциональность
- типы данных легко преобразовываются между MongoDB и Clojure
- ленивость дает дополнительный выигрыш

Clojure: MongoDB API

```
(defn mongo-find
  ([collection query skip limit]
   (let
     [result (.find collection (native-to-dboject query))
      result (if skip (.skip result skip) result)
      result (if limit (.limit result limit) result)]
     (map dboject-to-native
      (iterator-seq (.iterator result)))))
....
(defn dboject-to-native [dboject]
  (cond
    (instance? java.util.List dboject)
      (into [] (map dboject-to-native dboject))

    (instance? com.mongodb.ObjectId dboject)
      (str *oid-prefix* (.toString dboject))

    (instance? com.mongodb.DBObject dboject)
      (if (.get dboject "$keyword")
        (keyword (.get dboject "$keyword"))
        (into {}
          (map #(let [[k v] %] (vector (keyword k) (dboject-to-native v))) dboject)))

    :default
      dboject
  ))
....
```

Clojure: MongoDB API contd.

```
(defn native-to-dboject [data]
  (cond
    (map? data)
      (let [result (BasicDBObject.)]
        (doseq [[k v] data]
          (.put result (if (keyword? k) (name k) (str k))
                (native-to-dboject v))))
      result)

    (vector? data)
      (collection-to-dboject data)

    (set? data)
      (throw (IllegalArgumentException. "sets are not supported by MongoDB, use vector"))

    (list? data)
      (throw (IllegalArgumentException. "lists are not supported by MongoDB, use vector"))

    (keyword? data)
      (native-to-dboject {:$keyword (name data)}))

    (mongo-oid? data)
      (com.mongodb.ObjectId. (.substring data (.length *oid-prefix*)))

    :default
      data
  ))
```

Спасибо!

Илья Обшадко, ENTARENA Inc.
ilya.obshadko@entarena.com

