Monger, a Clojure client for MongoDB

- Home
- All guides
- API reference
- Community
- Code
- Change log
- More Clojure libraries
- Donate
- Clojure Docs

About this guide

This guide covers:

What version of Monger does this guide cover?

This guide covers Monger 3.1 (including preview releases).

Overview

Monger's update API follows the following simple rule: the "syntax" for condition and update document structure is the same or as close as possible to MongoDB shell and the official drivers. In addition, Monger provides several convenience functions for common cases, for example, finding documents by id.

Updating multiple documents

`monger.collection/update` is the most commonly used way of updating documents. It takes a database, collection name, a query condition, an updated document and a number of options, the most commonly used of which is `:multi`. When `:multi` is set to `true`, it instructs MongoDB to update multiple documents that match the query.

```
(ns my.service
  (:require [monger.core :as mg]
            [monger.collection :as mc]))

(let [conn (mg/connect)
      db   (mg/get-db conn "monger-test")
      coll "scores"]
  ;; resets the entire score table by updating all documents to {"score": 0}
  (mc/update db coll {} {:score 0} {:multi true})

  ;; resets only those rows the entire score table where round equals 3
  (mc/update db coll {:round 3} {:score 0} {:multi true}))
```

Updating a Single Document

`monger.collection/update` can be instructed to only update a single document by passing `:multi` option with the value of `false`:

```
(ns my.service
  (:require [monger.core :as mg]
            [monger.collection :as mc]))

(let [conn (mg/connect)
      db   (mg/get-db conn "monger-test")
      coll "scores"]
  (mc/update db coll {:round 3} {:score 0} {:multi false}))
```

`monger.collection/update-by-id` is useful when document id is known:

```
(ns my.service
  (:require [monger.core :as mg]
            [monger.collection :as mc]))

(let [conn (mg/connect)
      db   (mg/get-db conn "monger-test")
      coll "scores"]
  (mc/update-by-id db coll oid {:score 1088}))
```

Overriding Write Concern

It is possible to override default write concern for a single operation using the `:write-concern` key that `monger.collection/update` and `monger.collection/update-by-id` accept:

```
(ns my.service
  (:require [monger.core :as mg]
            [monger.collection :as mc])
  (:import com.mongodb.WriteConcern))

(let [conn (mg/connect)
      db   (mg/get-db conn "monger-test")
      coll "scores"
      opts {:multi true :write-concern WriteConcern/JOURNAL_SAFE}]
  ;; updates score for player "sam", waiting for MongoDB server to
  ;; update recovery journal (trading some throughput for safety)
  (mc/update db coll {:username "sam"} {:score 1088} opts))
```

Upserts

MongoDB supports upserts, "update or insert" operations. To do an upsert with Monger, use `monger.collection/update` function with `:upsert` option set to true:

```
(ns my.service
  (:require [monger.core :as mg]
            [monger.collection :as mc]))

(let [conn (mg/connect)
      db   (mg/get-db conn "monger-test")
      coll "scores"]
  ;; updates score for player "sam" if it exists; creates a new document otherwise
  (mc/update db coll {:player "sam"} {$set {:score 1088}} {:upsert true}))
```

Note that upsert only inserts one document. Learn more about upserts in this MongoDB documentation section.

Atomic Modifiers

Overview

Modifier operations are highly-efficient and useful when updating existing values; for instance, they're great for incrementing counters, setting individual fields, updating fields that are arrays and so on.

MongoDB supports modifiers via update operation and Monger API works the same way: you pass a document with modifiers to `monger.collection/update`. For example, to increment number of views for a particular page:

```
(ns my.service
  (:require [monger.core :as mg]
            [monger.collection :as mc]
            [monger.operators :refer :all]))

(let [conn (mg/connect)
      db   (mg/get-db conn "monger-test")
      coll visits]
  (mc/update db coll {:url "http://megacorp.com"} {$inc {:visits 1}}))
```

Whenever you have a MongoDB shell example, query and update documents will have the same structure with Monger. Monger provides a number of macros in the `monger.operators` namespace to make your code look a little bit cleaner: for example, instead of `"$set"` you can use `$set` and it will be expanded to `"$set"` when your Clojure code is compiled (at the macro expansion time).

Atomic modifiers are documented in detail in this MongoDB documentation guide. Below we will demonstrate eachsome of them with (very short) code examples, but using them in general is identical to how you use them in the MongoDB shell.

Using $inc Operator

`$inc` operator increments (or decrements) one or more fields that have numeric values:

```
(ns my.service
  (:require [monger.core :as mg]
            [monger.collection :as mc]
            [monger.operators :refer :all]))

(let [conn (mg/connect)
      db   (mg/get-db conn "monger-test")
      coll "visits"]
  (mc/update db coll {:url "http://megacorp.com"} {$inc {:visits 1}}))
```

Using $set Operator

`$set` operator changes (sets) one or more fields for a document (or multiple documents)

```
(ns my.service
  (:require [monger.core :as mg]
            [monger.collection :as mc]
            [monger.operators :refer :all]))

(let [conn (mg/connect)
      db   (mg/get-db conn "monger-test")
      coll "products"]
  ;; sets "weight" field in the document to 20.5
  (mc/update db coll {:_id oid} {$set {:weight 20.5}})

  ;; sets several fields atomically
  (mc/update db coll {:_id oid} {$set {:weight 20.5 :color "blue" :width 10.75}}))
```

Using $unset Operator

`$unset` operator clears (removes) a single field from a document:

```
(ns my.service
  (:require [monger.core :as mg]
            [monger.collection :as mc]
            [monger.operators :refer :all]))

(let [conn (mg/connect)
      db   (mg/get-db conn "monger-test")
      coll "visits"]
  (mc/update db coll {:url "http://megacorp.com"} {$unset "unverified"}))
```

Using $push Operator

`$push` operator appends a single value to an array field (and allows for duplicates):

```
(ns my.service
  (:require [monger.core :as mg]
            [monger.collection :as mc]
            [monger.operators :refer :all]))

(let [conn (mg/connect)
      db   (mg/get-db conn "monger-test")
      coll "people"]
  ;; adds "überachievement" to the list of badges (which an array field)
  (mc/update db coll {:_id oid} {$push {:badges "überachievement"}}))
```

Using $pushAll Operator

`$pushAll` operator adds multiple values to an array field (and allows for duplicates):

```
(ns my.service
  (:require [monger.core :as mg]
            [monger.collection :as mc]
            [monger.operators :refer :all]))

(let [conn (mg/connect)
      db   (mg/get-db conn "monger-test")
      coll "people"]
  ;; adds multiple items to an array field
  (mc/update db coll {:_id oid} {$pushAll {:items ["Glass Star" "See No Evil"]}}))
```

Using $addToSet operator

`$addToSet` operator is similar to `$push` but will filter out duplicate entries (sets are collections that do not have duplicates):

```
(ns my.service
  (:require [monger.core :as mg]
            [monger.collection :as mc]
            [monger.operators :refer :all]))

(let [conn (mg/connect)
      db   (mg/get-db conn "monger-test")
      coll "accounts"]
  ;; adds a single value to an array field, ensuring there are no duplicates
  (mc/update coll {:_id oid} {$addToSet {:permissions ["write"]}}))
```

Using $pull Operator

```
(ns my.service
  (:require [monger.core :as mg]
            [monger.collection :as mc]
            [monger.operators :refer :all]))

(let [conn (mg/connect)
      db   (mg/get-db conn "monger-test")
      coll "accounts"]
  ;; removes a single value from an array field
  (mc/update coll {:_id oid} {$pull {:permissions "write"}}))
```

Using $pullAll Operator

```
(ns my.service
  (:require [monger.core :as mg]
            [monger.collection :as mc]
            [monger.operators :refer :all]))

(let [conn (mg/connect)
      db   (mg/get-db conn "monger-test")
      coll "people"]
  ;; removes multiple items from an array field
  (mc/update coll {:_id oid} {$pullAll {:items ["Glass Star" "See No Evil"]}}))
```

Using Save Operation

`monger.collection/save` function performs insert or update (replace) based on whether or not provided document already has an id (whether the `:_id` field is set).

With a new document, it works very much like `monger.collection/insert`:

```
(let [coll "people"
      doc  {:name "Joe" :age 30}]
    ;; here monger.collection/save works the same way as monger.collection/insert
    (mc/save db coll doc))
```

`monger.collection/save-and-return` with new documents works the same way as `monger.collection/insert-and-return`:

```
(let [coll "people"
      doc  {:name "Joe" :age 30}]
    ;; here monger.collection/save-and-return works the same way as monger.collection/insert-and-return
    (mc/save-and-return db coll doc))
```

With documents that are not new (already have the `:_id` field), `monger.collection/save` and `monger.collection/save-and-return` will first look up the existing document by id and replace it with the one provided:

```
(let [coll   "people"
      doc    (mc/insert-and-return coll {:name "Joe" :username "sadjoe" :age 30})
      doc-id (:_id document)]
    ;; finds and updates a document by _id because it is present
    (mc/save-and-return db coll { :_id doc-id :name "Joe" :age 30 :username "happyjoe" }))
```

What To Read Next

The documentation is organized as a number of guides, covering all kinds of topics.

We recommend that you read the following guides first, if possible, in this order:

- Deleting documents
- Indexing and other collection operations
- Integration with 3rd party libraries
- Map/Reduce
- GridFS support
- Using MongoDB Aggregation Framework
- Using MongoDB commands

Tell Us What You Think!

Please take a moment to tell us what you think about this guide on Twitter or the Monger mailing list

Let us know what was unclear or what has not been covered. Maybe you do not like the guide style or grammar or discover spelling mistakes. Reader feedback is key to making the documentation better.

comments powered by Disqus