# 3 Grundlegende Konzepte

Nachdem Sie im letzten Kapitel die ersten praktischen Erfahrungen mit MongoDB und der Mongo Shell gesammelt haben, werden wir nun näher auf die dahinter liegenden Konzepte und Begriffe eingehen und die technischen Grundlagen beleuchten, auf denen MongoDB basiert.

Dieses Kapitel ist insbesondere für Architekten und Administratoren interessant, die die interne Arbeitsweise von MongoDB besser verstehen möchten. Als Entwickler können Sie beim ersten Lesen ggf. direkt zum Kapitel 6 springen, da ab dort das API von MongoDB beschrieben wird.

## 3.1 Konnektivität

Zunächst möchte ich Ihnen zeigen, welche grundsätzlichen Möglichkeiten es gibt, mit einem MongoDB-Server Verbindung aufzunehmen.

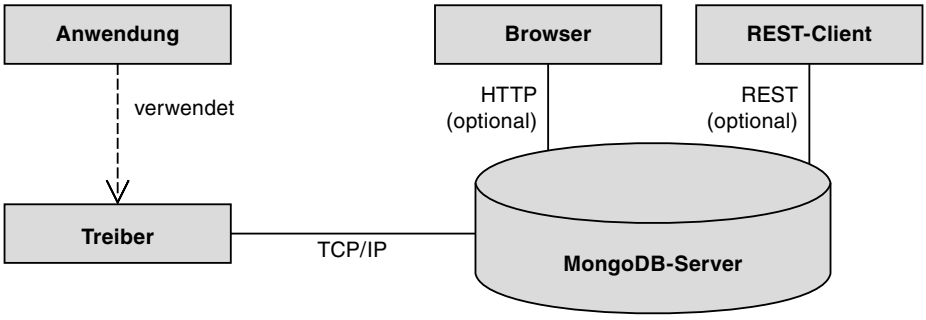


***Abb. 3–1*** *Schnittstellen eines MongoDB-Servers*

MongoDB arbeitet mit einem TCP/IP-basierten, binären Protokoll. Standardmäßig lauscht ein Server auf Port 27017. Soll ein anderer Port verwendet werden, kann dies durch die Kommandozeilenoption

```
$ mongod --port <Portnummer>
```

konfiguriert werden.

#### Treiber

Glücklicherweise müssen Sie das MongoDB-Protokoll nicht selbst implementieren, sondern können auf Treiber[1] zurückgreifen, die für eine Vielzahl von Programmiersprachen bereitstehen. Die Treiber konvertieren die jeweiligen, sprachspezifischen Datenstrukturen in das BSON-Format, das MongoDB zur Übertragung und auch zur Speicherung der Dokumente verwendet.

In Kapitel 11 stelle ich exemplarisch die Treiber für Java und Ruby genauer vor.

#### HTTP-Interface

Darüber hinaus gibt es eine HTTP-basierte Administrationsoberfläche, die wir in der Einleitung schon vorgestellt haben und im Standardfall unter `http://<host>:28017` zu erreichen ist. Diese Schnittstelle ist per Default deaktiviert. Wenn Sie Ihren Server mit dem HTTP-Interface betreiben wollen, können Sie dies beim Start auf der Kommandozeile wie folgt konfigurieren:

```
$ mongod --httpinterface
```

#### REST-Schnittstelle

Schließlich gibt es noch die sogenannte REST-Schnittstelle, die lesende Zugriffe auf Collections anbietet. Diese Schnittstelle muss zunächst mit

```
$ mongod --rest
```

aktiviert werden. Danach können auf dem Port, der auch vom HTTP-Interface genutzt wird, Dokumente unter

```
http://<host>:<port>/datenbank/collection/
```

abgerufen werden, z. B.

```
http://localhost:28017/twitter/tweets/
```

Ich werde auf die REST-Schnittstelle noch detailliert in Kapitel 10 eingehen.

#### Mongo Shell

Die sogenannte Mongo Shell ist eine Kommandozeilenanwendung, die sich wie jede andere Anwendung auch über TCP/IP mit dem MongoDB-Server verbindet. Wir haben sie in Kapitel 2 bereits ausgiebig verwendet. Die Mongo Shell ist Teil der Distribution und wird wie folgt aufgerufen:

```
$ mongo
```

---

1. *http://www.mongodb.org/display/DOCS/Drivers*

Ohne die Angabe von Parametern wird eine Verbindung zu dem Server `localhost:27017` zur Datenbank `test` aufgebaut. Sie können die Verbindungsparameter aber auch explizit angeben:

```
$ mongo localhost:27017/twitter
MongoDB shell version: 2.x
connecting to: localhost:27017/twitter
>
```

Ich werde die Mongo Shell im weiteren Verlauf ausgiebig verwenden und auf ihre Möglichkeiten dann jeweils im Detail eingehen.

## 3.2 Datenhaltung

Extrem wichtig für das Verständnis der Vor-, aber auch Nachteile, die MongoDB mit sich bringt, ist die interne Arbeitsweise eines einzelnen MongoDB-Knoten.

Wie schafft es MongoDB, der hohen Last bei Schreib- und Lesezugriffen zu begegnen? Die Antwort ist recht simpel: Die Daten werden erst mal im RAM abgelegt und von dort periodisch mittels des Betriebssystemdienstes *mmap* (s. Kasten) auf Disk synchronisiert (s. Abb. 3–2). Da die Zugriffszeiten auf das RAM sich im Bereich von Nanosekunden bewegen, sind die Daten schnell geschrieben. Dies ist ein Vorteil gegenüber dem Dateizugriff im Millisekundenbereich von ca. drei Größenordnungen.

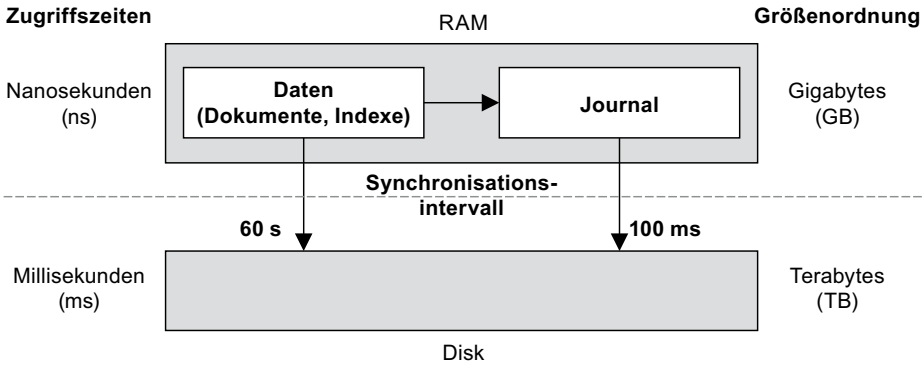


***Abb. 3–2*** *Interne Arbeitsweise eines MongoDB-Servers*

Nachteil ist allerdings, dass die Daten bis zur nächsten Synchronisation mit dem Dateisystem zunächst nur transient, also flüchtig sind. Stürzt der Server-Prozess jetzt ab, sind die Daten weg.

**mmap**

UNIX-Betriebssysteme (und auch Windows) bieten einen Systemdienst namens `mmap`[a] an, mit dem der Inhalt von Dateien transparent auf Speicherblöcke im RAM abgebildet werden kann. Man spricht dann von sogenannten *Memory Mapped Files*. Die Anwendung allokiert im Prinzip wie gewohnt Speicher, dessen Inhalt aber »automatisch« im Dateisystem persistiert wird.

a. *http://pubs.opengroup.org/onlinepubs/7908799/xsh/mmap.html*

Mit der Standardkonfiguration synchronisiert MongoDB den Speicher alle 60 Sekunden mit dem Dateisystem. Diese Einstellung können Sie mit der Kommandozeilenoption `--syncdelay <Sekunden>` beeinflussen. 60 Sekunden sind eine sehr lange Zeit, in der erstens jede Menge Daten anfallen und zweitens auch verloren gehen können.

Sie brauchen also zum einen möglichst viel RAM. Ein MongoDB-Server verwendet so viel Speicher, wie das Betriebssystem ihm anbietet. Daher sollten Sie produktive MongoDB-Systeme immer auf dedizierten Servern betreiben.

Um dem Datenverlust zu begegnen, also die Dauerhaftigkeit[2] der Schreiboperationen zu gewährleisten, verwendet MongoDB ein sogenanntes *Journaling*. Dabei werden Änderungen an Datenstrukturen zunächst sequenziell im Journal protokolliert, bevor sie intern wirksam werden. Standardmäßig ist das Journal auf 64-Bit-Systemen aktiv und wird alle 100 Millisekunden mit dem Dateisystem synchronisiert, sodass im Zweifelsfall nur Änderungen verloren gehen, die in diesem Zeitraum durchgeführt werden. Dieses Intervall kann mit der Kommandozeilenoption `--journalCommitInterval <Millisekunden>` überschrieben werden. Zulässig sind die Werte von 2 ms bis 300 ms. Kommt es zwischen zwei Synchronisationspunkten zu einem Absturz von MongoDB oder gar des ganzen Systems, werden beim nächsten Neustart die im Journal protokollierten Aktionen auf den Datenbestand angewandt, um die Datenkonsistenz wiederherzustellen.

Beim Starten des MongoDB-Servers sehen Sie in der Regel u.a. auch folgende Zeilen

```
Wed Jan 23 11:08:19 [initandlisten] journal dir=/data/journal
Wed Jan 23 11:08:19 [initandlisten] recover : no journal files present, no
recovery needed
```

die signalisieren, dass keine Wiederherstellung von Daten aus dem Journal notwendig ist.

Mit den beiden Synchronisationsintervallen stehen Ihnen zwei wesentliche Stellschrauben zur Verfügung, mit denen Sie die Balance zwischen Performance und Dauerhaftigkeit von Schreiboperationen beeinflussen können.

2. *http://de.wikipedia.org/wiki/ACID#Dauerhaftigkeit*

Nachdem ich Ihnen die allgemeine Arbeitsweise eines MongoDB-Servers vorgestellt habe, werde ich nun näher auf die Begriffe *Datenbanken*, *Collections*, *Dokumente, Felder* und *Indizes* eingehen.

## 3.3 Datenbanken

Ein MongoDB-Server (also ein laufender *mongod*-Prozess) kann mehrere Datenbanken verwalten. Eine Datenbank ist zunächst ein logischer Namensraum für die darin enthaltenen Collections. Zur Anzeige der Datenbanken starten wir die Mongo Shell und rufen folgenden Befehl auf:

```
$ mongo
MongoDB shell version: 2.x
connecting to: test
> show dbs
local   0.078125GB
test    0.203125GB
twitter 0.203125GB
```

Wir erkennen eine leere Datenbank namens `local` (mehr dazu später) sowie zwei weitere, die durch unsere ersten Tests bzw. den Datenimport entstanden sind. Im Dateisystem (im Verzeichnis `/data`) finden wir u.a. entsprechende Dateien vor:

```
07.12.2012  15:35    <DIR>          journal
07.12.2012  15:35                 5 mongod.lock
30.10.2012  07:37        67.108.864 test.0
30.10.2012  07:37       134.217.728 test.1
30.10.2012  07:37        16.777.216 test.ns
30.11.2012  15:10        67.108.864 twitter.0
30.11.2012  15:10       134.217.728 twitter.1
30.11.2012  15:10        16.777.216 twitter.ns
```

Pro Datenbank legt MongoDB eine Datei `<name>.ns` an, die als Default eine Größe von 16 MB hat. Darin werden die Namespaces gespeichert. Bei einer Größe von 16 MB können ca. 24.000 Namespaces verwendet werden. Die Größe der `.ns`-Datei kann bei Bedarf mit dem Parameter `--nssize` erhöht werden, was in der Praxis allerdings relativ selten auftreten sollte.

Die Dokumente einer Collection sowie Indexdaten werden in den durchnummerierten Daten `<name>.0`, `<name>.1` usw. gespeichert (s. Abb. 3–3). Diese haben feste Größen von 64 MB, 128 MB usw. bis hinauf zu 2 GB. Alle weiteren Dateien sind ebenfalls 2 GB groß. Die Idee dahinter ist u.a. eine Fragmentierung im Dateisystem zu verhindern. Sobald auf Datei `<name>.x` eine gewisse Menge an Daten gespeichert ist, wird im Hintergrund bereits die nächste Datei `<name>.x+1` angelegt. Diese Vorbelegung kann mit dem Kommandozeilenparameter `--noprealloc` unterbunden werden, wovon man in produktiven Systemen allerdings absehen sollte, da die Vorbelegung sonst erst genau dann erfolgt, wenn ein neuer Datensatz angelegt wird.

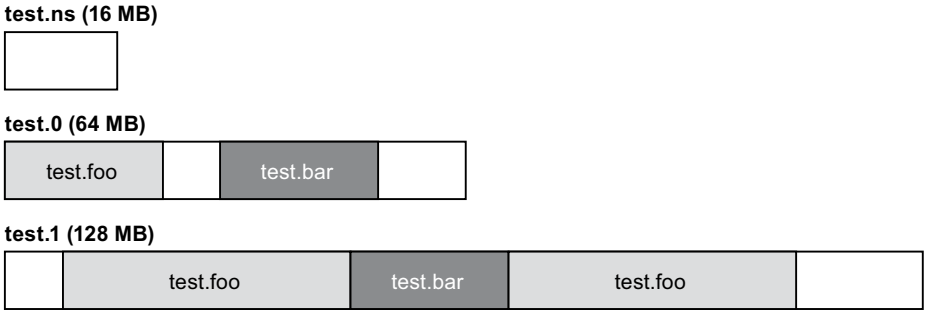


***Abb. 3–3*** *Datenbanken im Dateisystem*

Sie können MongoDB mit der Option `--directoryperdb` anweisen, pro Datenbank ein eigenes Unterverzeichnis anzulegen, in dem dann die einzelnen durchnummerierten Dateien abgelegt werden.

Da der Name einer Datenbank sich direkt auf das Dateisystem auswirkt, unterliegt er gewissen Einschränkungen. Er …

- darf keinen Punkt (.) enthalten.
- unterscheidet zwischen Groß- und Kleinschreibung.
- muss kürzer als 123[3] Zeichen sein.
- darf unter Windows keins der folgenden Zeichen enthalten: `/\."*<>:|?`

Darüber hinaus werden auf Ebene einer Datenbank folgende Aspekte konfiguriert bzw. behandelt:

- Sharding (s. Kap. 5)
- Authentifikation
- Profiling (s. Kap. 6)
- Locking

MongoDB verwendet folgende interne Datenbanken:

| Datenbank | Beschreibung |
|---|---|
| local | Diese Datenbank ist innerhalb eines Replica Sets (s. Kap. 4) von der Replication ausgenommen. |
| admin | Hier werden Authentifizierungs- und Autorisierungsinformationen gespeichert. |
| config | Enthält Meta-Information zum Sharding innerhalb eines *mongos*-Prozesses. |

***Tab. 3–1*** *Übersicht über interne Datenbanken*

3. *http://docs.mongodb.org/manual/reference/limits/*

Eine Übersicht der aktuellen Datenbank kann mit folgendem Befehl abgerufen werden:

```
> db.stats()
{
        "db" : "twitter",
        "collections" : 5,
        "objects" : 53682,
        "avgObjSize" : 1598.4299392720093,
        "dataSize" : 85806916,
        "storageSize" : 94015488,
        "numExtents" : 14,
        "indexes" : 2,
        "indexSize" : 1749664,
        "fileSize" : 469762048,
        "nsSizeMB" : 16,
        "ok" : 1
}
```

Weitere Kommandos sehen Sie nach der Eingabe von

```
> db.help()
```

Mit diesen Kommandos werden Sie in den folgenden Kapiteln nähere Bekanntschaft machen.

## 3.4 Collections

Eine Collection ist im Wesentlichen eine Sammlung von Dokumenten. Collections stellen technisch keine Anforderungen an die in ihnen enthaltenen Dokumente. Diesen Umstand nennt man *Schemafreiheit*. Anders als z.B. eine Tabelle eines relationalen Datenbanksystems definiert eine Collection keine Namen und Typen für einzelne Felder innerhalb eines einzelnen Datensatzes. Am folgenden Beispiel wird dies klarer:

```
> db.dokumente.insert( {a:1, b:2} )
> db.dokumente.insert( {b: "zwei", c: [3,4]} )
> db.dokumente.find( {}, {_id:0} )
{ "a" : 1, "b" : 2 }
{ "b" : "zwei", "c" : [ 3, 4 ] }
```

Die beiden Dokumente beinhalten zum einen unterschiedliche Felder, zum anderen variiert der Typ für das Feld `b` von Dokument zu Dokument.

> *Die Schemafreiheit ist ein technischer Aspekt. Inhaltlich muss sich der Entwickler durchaus Gedanken über sein Schema machen!*

Mit der Schemafreiheit könnte man auf den Gedanken kommen, einfach alle Daten seiner Anwendung in eine Collection zu packen. Überlegen Sie selbst,

warum das suboptimal sein könnte! Der Begriff Freiheit impliziert spontan oft die völlige Abgabe von Verantwortung und Verstand. Ich bevorzuge daher eher den Begriff von Flexibilität, möchte also eher von einem *flexiblen Schema* sprechen. In Kapitel 8 werde ich ausführlich auf dieses Thema eingehen.

Für die Benennung von Collections gelten folgende Einschränkungen. Der Name …

- sollte mit einem Unterstrich _ oder einem Buchstaben beginnen.
- darf nicht das Dollarzeichen `$` enthalten.
- darf kein Leerstring und nicht `null` sein.
- darf nicht mit dem reservierten Präfix `system.` beginnen.
- muss kürzer als $123^4$ Zeichen sein.

In der Praxis sind das keine allzu strengen Einschränkungen. Es gibt keine Hierarchie unter Collections, auch wenn der erlaubte Punkt (.) im Namen dies suggerieren könnte. Die folgenden Namen sind zulässig:

```
schaden.zahlungen
schaden.buchungen
```

Ansonsten steht es Ihnen frei, nach welchem Muster Sie Ihre Collections benennen. Einheitlichkeit hat allerdings noch nie geschadet.

Collections können implizit und explizit angelegt werden. Die Erzeugung passiert implizit beim ersten schreibenden Zugriff …

```
> db.dokumente.insert( {hello: "MongoDB"} )
```

… oder explizit über das folgende Datenbankkommando:

```
> db.createCollection("dokumente")
{ "ok" : 1 }
```

Den Weg der expliziten Anlage müssen Sie z.B. dann wählen, wenn Sie eine sog. *Capped Collection* (s. weiter hinten in diesem Kapitel) anlegen.

Auf der Ebene einer Collection kommen darüber hinaus folgende Aspekte zum Tragen:

- Definition von Indizesn (s. Kap. 6)
- Ausführen von Queries (s. Kap. 6)
- MapReduce bzw. Aggregation (s. Kap. 9)
- CRUD-Operationen (s. Kap. 7)
- Sharding (s. Kap. 5)

*Queries operieren immer nur auf genau einer Collection!*

4. *http://docs.mongodb.org/manual/reference/limits/*

Mit der Mongo Shell können Sie sich alle Collections der aktuellen Datenbank ausgeben lassen:

```
> use twitter
switched to db twitter
> show collections
dokumente
system.indexes
tweets
```

Eine einzelne Collection bietet eine Vielzahl von Kommandos an, die Sie sich mit

```
> db.twitter.help()
```

anzeigen lassen können. Im weiteren Verlauf werden wir uns diese Kommandos genauer ansehen.

#### Interne Speicherung

Intern werden die Daten einer Collection in sogenannten *Extents* abgelegt. Ein Extent ist ein zusammenhängender Speicherbereich, der die Daten von genau einem Namespace (also einer Collection oder einem Index) enthält. Ein Namespace kann allerdings mehrere Extents belegen. Wenn eine Collection oder ein Index im Laufe der Zeit wächst, werden sukzessive neue Extents allokiert. Durch das Löschen von Dokumenten werden Extents ggf. wieder frei und können anschließend wiederwendet werden. Mit

```
> db.tweets.validate()
{
        "ns" : "twitter.tweets",
        "firstExtent" : "0:2000 ns:twitter.tweets",
        "lastExtent" : "1:13f1000 ns:twitter.tweets",
        "extentCount" : 10,
        "datasize" : 85794956,
        "nrecords" : 53641,
        "lastExtentSize" : 28217344,
        "padding" : 1,
        "firstExtentDetails" : {
                "loc" : "0:2000",
                "xnext" : "0:13000",
                "xprev" : "null",
                "nsdiag" : "twitter.tweets",
                "size" : 24576,
                "firstRecord" : "0:20b0",
                "lastRecord" : "0:7b14"
        },
...
}
```

erhalten Sie eine Übersicht über die internen Informationen zu einer Collection. Das Feld `extentCount` lässt erkennen, dass die Collection insgesamt zehn Extents

belegt. Für den ersten Extent werden noch weitere Zusatzinformationen in Feld `firstExtentDetails` ausgegeben. Zur Anzeige aller Extents verwenden Sie folgendes Kommando:

```
> db.tweets.validate(true)
```

So erhalten Sie zusätzlich Informationen zu allen Extents im Feld `extents`. Interessant ist hier, dass die Größe der Extents jeweils zunimmt, was wir uns mit einer kleinen JavaScript-Funktion ausgeben lassen:

```
> function showExtentSizes(collection) {
      extents = db[collection].validate(true).extents;
      extents.forEach( function(extent) {
         print(extent.size)
      })
}
> showExtentSizes("tweets")
24576
98304
393216
1572864
6291456
8495104
11468800
15482880
20901888
28217344
```

#### Padding

Nach dem Einfügen ist ein Dokument an einer bestimmten Adresse innerhalb eines Extents abgespeichert. Solange ein Update das Dokument nicht vergrößert, ändert sich diese Adresse auch nicht. Wenn ein Update aber zu einem größeren Dokument führt, dann wird das Dokument an eine passende freie Stelle verschoben und alle Indizes, die das Dokument betreffen, müssen angepasst werden.

Um dieses relativ teure Verschieben zu vermeiden, setzt MongoDB automatisch ein *Padding* ein, d.h., für jedes Dokument wird etwas mehr Speicher reserviert als eigentlich benötigt. Der Anteil an zusätzlichem Speicherplatz wird durch den sogenannten Padding-Faktor bestimmt, der sich über die Zeit in Abhängigkeit vom Update-Verhalten auf den Dokumenten einer Collection ändern kann.

Den aktuellen Padding-Faktor können Sie (zusammen mit anderen interessanten Werten) mit:

```
> db.tweets.stats()
{
        "ns" : "twitter.tweets",
        "count" : 53641,
        "size" : 85794956,
        "avgObjSize" : 1599.4287205682222,
        "storageSize" : 92946432,
        "numExtents" : 10,
        "nindexes" : 1,
        "lastExtentSize" : 28217344,
        "paddingFactor" : 1,
        "systemFlags" : 1,
        "userFlags" : 0,
        "totalIndexSize" : 1741488,
        "indexSizes" : {
                "_id_" : 1741488
        },
        "ok" : 1
}
```

abfragen. Die Einheit der speicherbezogenen Werte ist standardmäßig ein Byte. Wenn Sie sich die Werte in KB oder MB anzeigen lassen wollen, rufen Sie alternativ `db.tweets.stats(1024)` bzw. `db.tweets.stats(1024 * 1024)` auf.

Wie Sie sehen, ist der Padding-Faktor hier 1 (=100 %), d.h., die Dokumente verbrauchen nicht mehr Speicher, als sie auch benötigen. Das liegt daran, dass wir die Tweets frisch importiert und noch keins der Dokumente verändert haben. Wenn Sie sehen möchten, wie sich der Padding-Faktor ändert, können Sie das folgende Beispiel aufrufen:

```
> db.padding.drop()
> for (var i=0; i<10000; i++) {
      var doc = {};
      doc["f" + i] = i;
      db.padding.update({_id: i%100},{$set:doc}, true, false)
}
> print( db.padding.stats().paddingFactor )
1.0350000000003639
```

Auf die Bedeutung der Parameter des Update-Kommandos gehen wir im Detail in Kapitel 7 ein. Interessant ist an dieser Stelle, dass die häufigen Updates den Padding-Faktor erhöht haben, sodass nun bei jedem zukünftigen Insert ca. 103,5 % Speicherplatz pro Dokument belegt werden. Wenn Dokumente kleiner werden, nimmt der Padding-Faktor allerdings auch wieder ab und nähert sich der 1 an:

```
> for(var i=0; i<10000; i++) {
      var doc = {};
      doc["f" + i] = i;
      db.padding.update({_id: i%100},{$unset:doc}, true, false)
}
> print( db.padding.stats().paddingFactor )
1.0000000000003677
```

Machen Sie sich also keine allzu große Sorgen um zu viel verbrauchten Speicherplatz, abschalten lässt sich das Padding ohnehin nicht.

Seit Version 2.2 können Sie allerdings auf einer Collection ein Flag setzen, das nur Dokumentgrößen zulässt, die eine 2er-Potenz (also 2, 4, 8, 16, 32 Bytes etc.) darstellen. Dies wirkt sich auch auf das Padding aus, wie Sie an folgendem Beispiel ausprobieren können:

```
> db.padding.drop()
true
> db.createCollection("padding")
{ "ok" : 1 }
> db.runCommand( {"collMod" : "padding" , "usePowerOf2Sizes" : true } )
{ "usePowerOf2Sizes_old" : false, "ok" : 1 }
```

Anschließend rufen Sie wieder das erste obige JavaScript auf, das Dokumente anlegt. Sie werden sehen, dass der Padding-Faktor jetzt geringer ist.

### 3.4.1 System Collections

MongoDB kennt ein Namensraumpräfix `system.` für Collections mit besonderer Bedeutung[5]. Im Einzelnen handelt es sich um die folgenden Collections:

| Name | Bedeutung |
|---|---|
| system.indexes | Nimmt alle Indizes der Datenbank auf.<br>Sie entsteht beim Anlegen des ersten Index (was implizit beim Anlegen der ersten Collection geschieht). |
| system.namespaces | Liste aller Namespaces (vollqualifizierter Name von Collections oder Indexe).<br>Diese Collection ist immer vorhanden, taucht aber in der Liste der Collections (s. `show collections`) nicht auf.<br>Lassen Sie sich alle bekannten Namespaces anzeigen mit<br>`> db.system.namespaces.find()` |
| | Bei aktiviertem Profiling (s. Abschnitt 6.5) werden in dieser Collection Profiling-Informationen abgelegt. |
| system.users | Bei aktivierter Authentifizierung werden in dieser Collection Informationen über Benutzer abgelegt. |
| system.roles | Nimmt die Definition von benutzerdefinierten Rollen auf. |
| system.js | Hier können JavaScript-Variablen und Funktionen zentral hinterlegt werden, die dann in `db.eval(…)` referenziert werden können. |

***Tab. 3–2*** *Überblick der System Collections*

5. *http://docs.mongodb.org/manual/reference/system-collections/*

### 3.4.2 Capped Collections

Die sogenannten *Capped Collections* sind eine besondere Art von Collections. Hierbei handelt es sich um eine in Größe in Bytes oder Anzahl an Dokumenten begrenzte Collection, aus der ältere Dokumente nach dem FIFO-Prinzip gelöscht werden, wenn beim Einfügen neuer Dokumente die jeweilige Grenze überschritten wird.

Eine Capped Collection kann nur explizit angelegt werden, da bei der Erzeugung Parameter für die maximale Größe bzw. Dokumentenanzahl mitgegeben werden müssen:

```
> db.createCollection(
      "capping",
      {capped: true, size: 1048576, max: 10}
)
{ "ok" : 1 }
```

Mit dem Flag `capped` wird gesteuert, ob die Collection ein Capping verwenden soll oder nicht. Der folgende Pflichtparameter `size` gibt die Maximalgröße der Collection in Bytes an. Mit dem optionalen Parameter `max` kann zusätzlich die Anzahl der Dokumente innerhalb der Collection begrenzt werden. Mit der Anlage einer Capped Collection wird automatisch der mit `size` angegebene Speicher direkt allokiert, was Sie mit

```
> db.capping.validate()
```

überprüfen können.

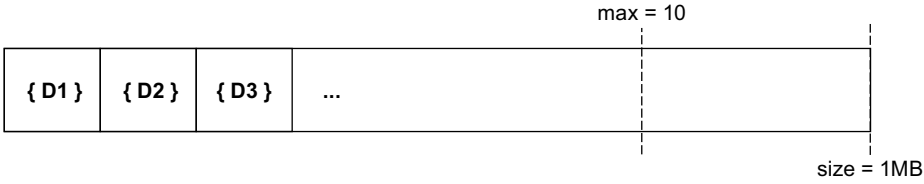


***Abb. 3–4*** *Capped Collection*

Das Capping orientiert sich also an bis zu zwei Schwellwerten. Wird beim Einfügen eines weiteren Dokuments einer dieser Schwellwerte überschritten, werden so lange »alte« Dokumente (im Sinne der Einfügereihenfolge) gelöscht, bis Platz für das »neue« Dokument ist. Im folgenden Beispiel fügen wir elf Dokumente ein und werfen anschließend einen Blick auf die Dokumente:

```
> for (i=1;i<12; i++) { db.capping.insert( {_id:i} ) }
> db.capping.find()
{ "_id" : 2 }
{ "_id" : 3 }
{ "_id" : 4 }
{ "_id" : 5 }
{ "_id" : 6 }
{ "_id" : 7 }
{ "_id" : 8 }
{ "_id" : 9 }
{ "_id" : 10 }
{ "_id" : 11 }
```

Das Insert des elften Dokuments mit `_id=11` hat dazu geführt, dass das erste Dokument mit `_id=1` aus der Collection gelöscht wurde.

Für Capped Collection gelten im Gegensatz zu normalen Collections einige Einschränkungen. Das Löschen von Dokumenten ist nicht erlaubt:

```
> db.capping.remove( {_id:3} )
can't remove from a capped collection
```

Updates dürfen nur dann ausgeführt werden, wenn das Dokument durch das Update nicht größer wird und damit seine ursprüngliche Einfügeposition beibehält (s. auch Abschnitt über das Padding). So ist

```
> db.capping.update( {_id:3}, {_id:3, mehr: "text"} )
failing update: objects in a capped ns cannot grow
```

beispielsweise nicht zulässig.

#### Use Cases

Eine Capped Collection bietet sich ganz allgemein in solchen Fällen an, in denen üblicherweise Ringpuffer bzw. Warteschlangen[6] verwendet werden.

MongoDB setzt Capped Collections auch selbst ein, und zwar beim sogenannten *Oplog* in Replica Sets (s. Kap. 4).

## 3.5 Dokumente

In der Einleitung haben wir gesehen, dass MongoDB in die Kategorie der sogenannten dokumentenorientierten NoSQL-Datenbanken fällt. Wir haben auch bereits mit ein paar Dokumenten gearbeitet. Für das weitere Verständnis ist es allerdings essenziell, dass wir den Begriff eines Dokuments etwas genauer fassen.

Zuallererst ist ein Dokument ein einzelner Datensatz in MongoDB. Schreibende Zugriffe auf Ebene eines Dokuments sind atomar, d.h., sie werden vollständig oder gar nicht ausgeführt. Dokumente werden aber auch zur Formulierung von Queries (s. Kap. 6), Updates und Sortierkriterien verwendet.

6. *http://de.wikipedia.org/wiki/Warteschlange_%28Datenstruktur%29*