*Эта книга посвящается нашим семьям*
*в благодарность за то время, свободу действий и поддержку,*
*которую они предоставили, чтобы сделать нашу работу*
*над этой книгой возможной, и за их любовь.*

*Анне, Сигурни, Грэму и Беккету – Шэннон*
*И Джемме, Клоде и Броне – Йон*

# Оглавление

# Предисловие

## Как устроена эта книга

Эта книга разделена на шесть частей, в которых приводятся сведения о разработке, администрировании и развертывании.

## Начало работы с MongoDB

В главе 1 мы рассказываем о MongoDB: почему она была создана, какие цели пытается достичь и почему вы можете использовать ее для своего проекта. Более подробно мы рассмотрим главу 2, в которой представлены базовые понятия и словарь MongoDB. В главе 2 вы приступите к работе с базой данных и оболочкой. Следующие две главы посвящены основному материалу, который необходимо знать разработчикам для работы с MongoDB. В главе 3 мы опишем, как выполнять базовые операции записи, в том числе как делать это с различными уровнями безопасности и скорости. В главе 4 объясняется, как найти документы и создавать сложные запросы, а также рассказывается, как перебирать результаты, здесь приводятся варианты для ограничения, пропуска и сортировки результатов.

## Разработка с MongoDB

В главе 5 описано, что такое индексирование и как индексировать свои коллекции MongoDB. В главе 6 объясняется, как использовать несколько специальных типов индексов и коллекций. В главе 7 рассматривается ряд методов агрегирования данных с MongoDB, включая подсчет, поиск различных значений, группировку документов, фреймворк агрегации и запись этих результатов в коллекцию. Глава 8 знакомит вас с транзакциями: что это такое, как лучше всего использовать их для своего приложения и как настроить. Наконец, эта часть заканчивается главой о разработке вашего приложения: в главе 9 содержатся советы по написанию приложения, которое хорошо работает с MongoDB.

## Репликация

Часть, посвященная репликации, начинается с главы 10, в которой дается быстрый способ настроить набор реплик локально, и охватывает многие из доступных параметров конфигурирования. Затем в главе 11 рассматриваются различные концепции, связанные с репликацией. В главе 12 показано, как репликация взаимодействует с вашим приложением, а в главе 13 разбираются административные аспекты запуска набора реплик.

## Шардинг

Часть, посвященная шардингу, начинается с главы 14, где дается описание быстрой локальной настройки. Затем в главе 15 приводится обзор компонентов кластера и рассказывается, как их настроить. Глава 16 содержит советы по выбору ключа шардинга для различных приложений. Наконец, глава 17 посвящена администрированию разделенного кластера.

## Администрирование приложений

В следующих двух главах рассматривается множество аспектов администрирования MongoDB с точки зрения вашего приложения. В главе 18 обсуждается, как проанализировать то, что делает MongoDB. Глава 19 посвящена безопасности в MongoDb, настройке аутентификации и авторизации для вашего развертывания. В главе 20 объясняется, как MongoDB надежно хранит данные.

## Администрирование сервера

Последняя часть посвящена администрированию сервера. В главе 21 описываются общие параметры при запуске и остановке MongoDB. В главе 22 обсуждается, что искать и как читать статистику во время мониторинга. В главе 23 описано, как сделать резервные копии и провести восстановление для каждого типа развертывания. Наконец, в главе 24 обсуждается ряд системных настроек, которые следует учитывать при развертывании MongoDB.

## Приложения

В приложении A объясняется схема управления версиями в MongoDB и ее установка в Windows, OS X и Linux. Приложение B подробно описывает внутреннюю работу MongoDB: механизм хранения, формат данных и проводной протокол.

## Обозначения, принятые в этой книге

В этой книге используются следующие типографские обозначения.

*Курсив*

Используется для обозначений новых терминов, URL-адресов, адресов электронной почты, имен коллекций, баз данных, файлов и расширений файлов.

`Моноширинный шрифт`

Используется в листингах программ, а также в абзацах для ссылки на элементы программы, такие как имена переменных или функ-

ций, утилиты командной строки, переменные среды, операторы и ключевые слова.

**`Моноширинный полужирный шрифт`**

Показывает команды или другой текст, который должен быть набран пользователем буквально.

*`Моноширинный курсив`*

Показывает текст, который должен быть заменен предоставленными пользователем значениями или значениями, определенными контекстом.

Этот элемент означает подсказку или предложение.

Этот элемент означает общее примечание.

Этот элемент указывает на предупреждение или предостережение.

## Использование примеров кода

Дополнительный материал (примеры кода, упражнения и т. д.) можно загрузить по адресу https://github.com/mongodb-the-definitive-guide-3e/mongodb-the-definitive-guide-3e.

Если у вас есть вопрос технического характера или возникла проблема, связанная с примерами кода, отправьте письмо на адрес bookquestions@oreilly.com.

Данная книга призвана помочь вам выполнить свою работу. В общем, вы можете использовать код из этой книги в своих программах и документации. Вам не нужно обращаться к нам за разрешением, если вы не воспроизводите значительную часть кода. Например, для написания программы, в которой используется несколько фрагментов кода из этой книги, оно не требуется. Продажа или распространение CD-ROM с примерами из книг

O'Reilly требует разрешения. Чтобы ответить на вопрос, сославшись на эту книгу и приведя пример кода, разрешение не требуется. Включение значительного количества примеров кода из этой книги в документацию вашего продукта требует разрешения.

Атрибуция желательна, но не является обязательной. Обычно она включает в себя название книги, автора, издателя и ISBN. Например: «Книга рецептов R, 2-е изд., Дж. Д. Лонг и Пол Титор. Copyright 2019 Дж. Лонг и Пол Титор, 978-1-492-04068-2».

Если вы считаете, что использование примеров кода выходит за рамки добросовестного применения или только что описанного разрешения, свяжитесь с нами по адресу permissions@oreilly.com.

## Обучение в режиме онлайн

На протяжении почти 40 лет O'Reilly Media (https://www.oreilly.com) предоставляет технологии и бизнес-тренинги, знания и анализ, чтобы помочь компаниям добиваться успеха.

Наша уникальная сеть экспертов и новаторов делится своими знаниями и опытом через книги, статьи, конференции и нашу онлайн-платформу обучения. Платформа онлайн-обучения O'Reilly предоставляет доступ по требованию к курсам обучения в режиме реального времени, углубленным способам обучения, интерактивным средам кодирования и обширной коллекции текстов и видео от O'Reilly и свыше 200 других издательств. Для получения дополнительной информации, пожалуйста, посетите сайт http://oreilly.com.

# Предисловие от издательства

## Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте **www.dmkpress.com**, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу **dmkpress@gmail.com**, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу **http://dmkpress.com/authors/publish_book/** или напишите в издательство по адресу **dmkpress@gmail.com**.

## Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу **dmkpress@gmail.com**, и мы исправим это в следующих тиражах.

## Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и O'Reilly очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу электронной почты **dmkpress@gmail.com** со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую предоставлять вам качественные материалы.

# Часть I

# Введение в MongoDB

# Глава 1

# Введение

MongoDB – это мощная, гибкая и масштабируемая система управления базами данных (СУБД) общего назначения. Она сочетает в себе возможность масштабирования с такими функциями, как вторичные индексы, запросы по диапазону, сортировка, агрегирование и геопространственные индексы. В этой главе рассматриваются основные проектные решения, которые сделали MongoDB тем, чем она является.

## Простота использования

MongoDB – это не реляционная, а *документоориентированная* система управления базами данных. Основной причиной отказа от реляционной модели является упрощение горизонтального масштабирования, но есть и другие преимущества.

Документоориентированная СУБД заменяет концепцию «строки» более гибкой моделью, «документом». Позволяя использовать вложенные документы и массивы, документоориентированный подход дает возможность представлять сложные иерархические отношения с помощью одной записи. Это естественным образом вписывается в то, как разработчики, работающие с современными объектно-ориентированными языками, рассматривают свои данные.

Также нет предопределенных схем: ключи и значения документа не имеют фиксированных типов или размеров. Когда нет фиксированной схемы, добавлять или удалять поля по мере необходимости становится проще. Как правило, это ускоряет разработку, поскольку разработчики могут быстро выполнять итерации. Экспериментировать также проще. Разработчики могут опробовать десятки моделей для данных, а затем выбрать лучшую.

## Разработана для масштабирования

Размеры наборов данных для приложений растут невероятными темпами. Увеличение доступной пропускной способности и дешевые хранилища создали среду, в которой даже небольшим приложениям необходимо

хранить больше данных, чем способны обработать многие базы данных. Терабайт данных, некогда неслыханный объем информации, теперь стал обычным явлением.

По мере роста объема данных, которые необходимо хранить разработчикам, последние сталкиваются с трудным решением: как масштабировать свои базы данных? Масштабирование базы данных сводится к выбору между вертикальным масштабированием (получение более крупной машины) и горизонтальным масштабированием (партиционирование данных на нескольких машинах). Вертикальное масштабирование часто является путем наименьшего сопротивления, но у него имеются свои недостатки: большие машины нередко очень дороги, и в конечном итоге достигается физический предел, когда более мощную машину нельзя купить любой ценой. Альтернативой является горизонтальное масштабирование: добавить место для хранения или увеличить пропускную способность для операций чтения и записи, приобрести дополнительные серверы и добавить их в свой кластер. Это и дешевле, и более масштабируемо; однако администрировать тысячу машин сложнее, чем заботиться об одной.

MongoDB была разработана для горизонтального масштабирования. Документоориентированная модель данных облегчает распределение данных между несколькими серверами. MongoDB автоматически заботится о балансировке данных и нагрузки в кластере, автоматически перераспределяя документы и направляя операции чтения и записи в нужные машины, как показано на рис. 1.1.

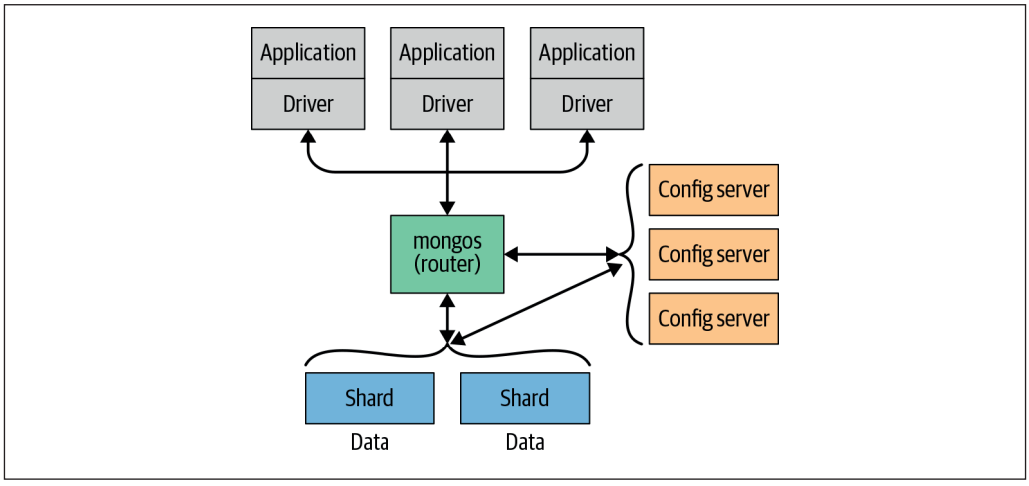


**Рис. 1.1.** Масштабирование MongoDB с использованием шардинга на нескольких серверах

Топология кластера MongoDB, или же это фактически кластер, а не один узел на другом конце соединения с базой данных, прозрачна для прило-

жения. Это позволяет разработчикам сосредоточиться на программировании приложения, а не на его масштабировании.

Аналогичным образом, если необходимо изменить топологию существующего развертывания, например для масштабирования, чтобы поддержать большую нагрузку, логика приложения может оставаться прежней.

## Богатство функций...

MongoDB – СУБД общего назначения, поэтому помимо создания, чтения, обновления и удаления данных она предоставляет большинство тех функций, которые можно ожидать от системы управления базами данных, и многие другие, которые выделяют ее. Среди них:

*Индексирование*

MongoDB поддерживает общие вторичные индексы и предоставляет уникальное, составное, геопространственное и полнотекстовое индексирование. Также поддерживаются вторичные индексы в иерархических структурах, таких как вложенные документы и массивы, которые позволяют разработчикам в полной мере использовать возможность моделирования таким способом, который наиболее подходит для их приложений.

*Агрегация*

MongoDB предоставляет фреймворк для агрегации на базе концепции конвейеров обработки данных. Конвейеры агрегации позволяют создавать сложные аналитические механизмы, обрабатывая данные через ряд относительно простых этапов на стороне сервера, используя все преимущества оптимизации базы данных.

*Специальные типы коллекций и индексов*

MongoDB поддерживает коллекции данных TTL (time-to-live), срок действия которых должен истечь в определенное время, такие как сеансы и коллекции фиксированного размера, для хранения недавно полученных данных, например журналов. MongoDB также поддерживает частичные индексы, ограниченные только теми документами, которые соответствуют фильтру критериев, чтобы повысить эффективность и уменьшить необходимый объем дискового пространства.

*Файловое хранилище*

MongoDB поддерживает простой в использовании протокол для хранения больших файлов и метаданных файлов.

Некоторые функции, распространенные в реляционных СУБД, в MongoDB отсутствуют, особенно сложные соединения. MongoDB поддержива-

ет соединения очень ограниченным образом посредством использования оператора агрегации `$lookup`, который появился в выпуске 3.2. В версии 3.6 более сложные соединения возможны с использованием нескольких условий объединения, а также несвязанных подзапросов. Обработка соединений в MongoDB была архитектурным решением, обеспечивающим бóльшую масштабируемость, поскольку обе эти функции сложно эффективно реализовать в распределенной системе.

## ...Без ущерба для скорости

Производительность является основной целью MongoDB и во многом сформировала ее дизайн. Она использует уступающую блокировку в своей подсистеме хранения WiredTiger, чтобы максимизировать параллелизм и пропускную способность. Она применяет столько оперативной памяти, сколько может, как и свой кеш, и пытается автоматически выбирать правильные индексы для запросов. Говоря кратко, почти каждый аспект MongoDB был разработан для поддержания высокой производительности.

Хотя MongoDB является мощным средством, включающим в себя множество функций реляционных систем, она не предназначена для выполнения всего того, что делает реляционная СУБД. В случае с некоторыми функциями сервер базы данных переносит обработку и логику на клиентскую сторону (обработка осуществляется либо драйверами, либо кодом приложения пользователя). Поддержание этого обтекаемого дизайна является одной из причин, по которой MongoDB может достигать такой высокой производительности.

## Философия

В этой книге мы уделим время тому, чтобы отметить причины или мотивы, стоящие за конкретными решениями, которые принимались при разработке MongoDB. С помощью этого мы надеемся поделиться философией MongoDB. Однако лучший способ подвести итоги проекта MongoDB – обозначить его главную цель – создание полноценного хранилища данных, которое является масштабируемым, гибким и быстрым.

# Глава 2

# Начало работы

MongoDB – мощный инструмент, с которым легко начать работу. В этой главе мы познакомимся с некоторыми основными понятиями MongoDB:

- *документ* представляет собой основную единицу данных в MongoDB и приблизительно эквивалентен строке в реляционной системе управления базами данных (но гораздо более выразителен);
- аналогично *коллекцию* можно рассматривать как таблицу с динамической схемой;
- один экземпляр MongoDB может содержать несколько независимых *баз данных*, каждая из которых содержит свои собственные коллекции;
- у каждого документа есть специальный ключ `"_id"`, который является уникальным в рамках коллекции;
- MongoDB распространяется с помощью простого, но мощного инструмента под названием оболочка *mongo*. Оболочка *mongo shell* предоставляет встроенную поддержку для администрирования экземпляров MongoDB и манипулирования данными с использованием языка запросов MongoDB. Это также полнофункциональный интерпретатор JavaScript, который позволяет пользователям создавать и загружать собственные сценарии для различных целей.

## Документы

В основе MongoDB лежит *документ*: упорядоченный набор ключей со связанными значениями. Представление документа зависит от языка программирования, но большинство языков имеют естественную структуру данных, например ассоциативный массив, хеш или словарь. Например, в JavaScript документы представлены в виде объектов:

```
{"greeting" : "Hello, world!"}
```

Этот простой документ содержит единственный ключ `"greeting"` со значением `"Hello, world!"`. Большинство документов будут более сложными

по сравнению с этим и часто будут содержать несколько пар типа «ключ/значение»:

```
{"greeting" : "Hello, world!", "views" : 3}
```

Как видно, значения в документах – это не просто «BLOB-объекты». Они могут относиться к одному из нескольких типов данных (или даже ко всему вложенному документу – *см.* раздел «Вложенные документы»). В этом примере значение `"greeting"` является строкой, тогда как значение `"views"` – это целое число.

Ключи в документе – это строки. В ключе допустимо использование любого символа в кодировке UTF-8, за несколькими заметными исключениями:

- ключи не должны содержать символ \ *0* (нулевой символ). Он используется для обозначения конца ключа;
- символы . и $ обладают некоторыми специальными свойствами и должны использоваться только при определенных обстоятельствах, о чем будет рассказано в последующих главах. В целом они должны считаться зарезервированными, и драйверы будут жаловаться, если эти символы будут использоваться не по назначению.

MongoDB чувствительна к типу и регистру. Например, эти документы отличаются:

```
{"count" : 5}
{"count" : "5"}
```

равно как и эти:

```
{"count" : 5}
{"Count" : 5}
```

И еще одна важная вещь: документы в MongoDB не могут содержать дубликаты ключей. Например, приведенный ниже документ не является допустимым:

```
{"greeting" : "Hello, world!", "greeting" : "Hello, MongoDB!"}
```

## Коллекции

*Коллекция* представляет собой группу документов. Если в MongoDB документ является аналогом строки в реляционной СУБД, то коллекцию можно рассматривать как аналог таблицы.

## Динамические схемы

Коллекции имеют *динамические схемы*. Это означает, что документы в одной коллекции могут иметь любое число различных «фигур». Например, оба приведенных ниже документа могут храниться в одной коллекции:

```
{"greeting" : "Hello, world!", "views": 3}
{"signoff": "Good night, and good luck"}
```

Обратите внимание на то, что предыдущие документы имеют разные ключи, разное количество ключей и значения разных типов. Поскольку любой документ можно поместить в любую коллекцию, часто возникает вопрос: «Зачем вообще нужны отдельные коллекции?» Если нет необходимости в отдельных схемах для разных видов документов, зачем использовать дополнительные коллекции? На то есть ряд веских причин:

- хранение разных видов документов в одной коллекции может стать кошмаром для разработчиков и администраторов. Разработчики должны убедиться, что каждый запрос возвращает только документы, привязанные к определенной схеме, или что код приложения, выполняющий запрос, может обрабатывать документы различной формы. Если мы запрашиваем посты в блоге, очень сложно отсеять документы, содержащие данные об авторах;
- получить список коллекций намного быстрее, чем извлечь список типов документов в коллекции. Например, если бы в каждом документе было поле `"type"`, в котором указывалось, был ли это документ «skim», «whole» или «chunky monkey», было бы намного медленнее искать эти три значения в одной коллекции, чем иметь три отдельные коллекции и запросить правильную;
- группировка документов одного и того же вида в одной коллекции допускает локальность данных. Получение нескольких постов в блоге из коллекции, содержащей только посты, вероятно, потребует меньше операций поиска на диске, нежели получение тех же постов из коллекции, где содержатся посты и данные об авторах;
- мы начинаем навязывать своим документам некую структуру при создании индексов. (Это особенно верно в случае с уникальными индексами.) Эти индексы определяются для каждой коллекции. Помещая в одну коллекцию только документы одного типа, можно более эффективно индексировать свои коллекции.

Существуют веские причины для создания схемы и группировки связанных типов документов. Хотя этого и не требуется по умолчанию, определение схем для вашего приложения является хорошей практикой и может быть реализовано с помощью функций проверки документации MongoDB

и библиотек объектно-документного отображения, доступных для множества языков программирования.

## Именование

Коллекция идентифицируется по имени. Имена коллекций могут быть любой строкой в кодировке UTF-8 с некоторыми ограничениями:

- пустая строка (“”) не является допустимым именем коллекции;
- имена коллекций не могут содержать символ \ *0* (нулевой символ), поскольку он обозначает конец имени коллекции;
- не следует создавать коллекции с именами, начинающимися со слова *system*. Этот префикс зарезервирован для внутренних коллекций. Например, коллекция *system.users* содержит пользователей базы данных, а коллекция *system.namespaces* содержит информацию обо всех коллекциях базы данных;
- созданные пользователем коллекции не должны содержать зарезервированный символ *$* в своих именах. Различные драйверы, доступные для базы данных, все же поддерживают применение этого символа в именах коллекций, потому что он содержится в некоторых сгенерированных системой коллекциях, но вы не должны использовать *$* в имени, только если вы не выполняете доступ к одной из этих коллекций.

### Вложенные коллекции

Одно из соглашений для организации коллекций состоит в том, чтобы использовать вложенные коллекции пространства имен, разделенные символом «.». Например, приложение, содержащее блог, может иметь коллекцию с именем *blog.posts* и отдельную коллекцию с именем *blog.authors*. Это служит только организационным целям – нет никакой связи между коллекцией *blog* (она даже не должна существовать) и ее «потомками».

Хотя вложенные коллекции не имеют каких-либо специальных свойств, они полезны и включены во многие инструменты MongoDB. Например:

- GridFS, протокол для хранения больших файлов, использует вложенные коллекции для хранения метаданных файлов отдельно от блоков содержимого (дополнительную информацию о GridFS *см.* в главе 6);
- большинство драйверов предоставляют некий «синтаксический сахар» для доступа к вложенной коллекции. Например, в оболочке базы данных `db.blog` предоставит вам коллекцию *blog*, а `db.blog.posts` – коллекцию *blog.posts*.

Вложенные коллекции – хороший способ организовать данные в MongoDB для множества случаев использования.

## Базы данных

В дополнение к группированию документов по коллекции MongoDB группирует коллекции в *базы данных*. Один экземпляр MongoDB может содержать несколько баз данных, каждая из которых объединяет ноль или более коллекций. Есть хорошее практическое правило – хранить все данные одного приложения в одной и той же базе данных. Отдельные базы данных полезны при хранении данных для нескольких приложений или пользователей на одном сервере MongoDB.

Как и коллекции, базы данных идентифицируются по имени. Имена баз данных могут быть любой строкой в формате UTF-8 со следующими ограничениями:

- пустая строка (“”) не является допустимым именем базы данных;
- имя базы данных не может содержать следующие символы: /, \, ., ”, *, <, >, :, |, ?, $, (один пробел) или \ *0* (нулевой символ). В основном придерживаться буквенно-цифровой таблицы ASCII;
- имена баз данных нечувствительны к регистру;
- имена баз данных ограничены максимум 64 байтами.

Традиционно, до использования подсистемы хранения WiredTiger, имена баз данных становились файлами в вашей файловой системе. Теперь этого больше нет. Это объясняет, почему многие из предыдущих ограничений вообще существуют.

Есть также некоторые зарезервированные имена баз данных, к которым можно получить доступ, но которые имеют особую семантику. Вот они:

*admin*

: База данных *admin* играет роль в аутентификации и авторизации. Кроме того, доступ к этой базе данных необходим для ряда административных операций. *См.* главу 19 для получения дополнительной информации о базе данных *admin*.

*local*

: В этой базе данных хранятся данные, относящиеся к одному серверу. В наборах реплик в базе *local* хранятся данные, используемые в процессе репликации. Сама база данных *local* никогда не реплицируется. (*См.* главу 10 для получения дополнительной информации о репликации и локальной базе данных.)

*config*

Разделенные (сегментированные) кластеры MongoDB (*см.* главу 14) используют базу данных *config* для хранения информации о каждом шарде.

Объединяя имя базы данных с коллекцией в этой базе данных, вы можете получить полное имя коллекции, которое называется *пространством имен*. Например, если вы используете коллекцию *blog.posts* в базе данных *cms*, пространство имен этой коллекции будет таким: *cms.blog.posts*. Длина пространств имен ограничена 120 байтами, а на практике должна быть менее 100 байт. Подробнее о пространствах имен и внутреннем представлении коллекций в MongoDB *см.* приложение B.

# Начало работы с MongoDB

Чтобы запустить сервер, выполните исполняемый файл *mongod* в выбранной вами среде с интерфейсом командной строки Unix:

```
$ mongod
2016-04-27T22:15:55.871-0400 I CONTROL [initandlisten] MongoDB starting :
pid=8680 port=27017 dbpath=/data/db 64-bit host=morty
2016-04-27T22:15:55.872-0400 I CONTROL [initandlisten] db version v4.2.0
2016-04-27T22:15:55.872-0400 I CONTROL [initandlisten] git version:
34e65e5383f7ea1726332cb175b73077ec4a1b02
2016-04-27T22:15:55.872-0400 I CONTROL [initandlisten] allocator: system
2016-04-27T22:15:55.872-0400 I CONTROL [initandlisten] modules: none
2016-04-27T22:15:55.872-0400 I CONTROL [initandlisten] build environment:
2016-04-27T22:15:55.872-0400 I CONTROL [initandlisten] distarch: x86_64
2016-04-27T22:15:55.872-0400 I CONTROL [initandlisten] target_arch: x86_64
2016-04-27T22:15:55.872-0400 I CONTROL [initandlisten] options: {}
2016-04-27T22:15:55.889-0400 I JOURNAL [initandlisten]
journal dir=/data/db/journal
2016-04-27T22:15:55.889-0400 I JOURNAL [initandlisten] recover :
no journal files
present, no recovery needed
2016-04-27T22:15:55.909-0400 I JOURNAL [durability] Durability thread started
2016-04-27T22:15:55.909-0400 I JOURNAL [journal writer] Journal writer thread
started
2016-04-27T22:15:55.909-0400 I CONTROL [initandlisten]
2016-04-27T22:15:56.777-0400 I NETWORK [HostnameCanonicalizationWorker]
Starting hostname canonicalization worker
2016-04-27T22:15:56.778-0400 I FTDC [initandlisten] Initializing full-time
diagnostic data capture with directory '/data/db/diagnostic.data'
2016-04-27T22:15:56.779-0400 I NETWORK [initandlisten] waiting for connections
on port 27017
```

Если вы работаете в Windows, запустите это:

```
> mongod.exe
```

Для получения подробной информации об установке MongoDB в вашей системе см. приложение A или соответствующее руководство по установке (https://oreil.ly/5WP5e) в документации MongoDB.

При запуске без аргументов файл *mongod* будет использовать каталог данных по умолчанию, */data/db/* (или *\data\db* на текущем томе в Windows). Если каталог данных отсутствует или недоступен для записи, сервер не запустится. Важно создать каталог данных (например, `mkdir -p /data/db/`) и убедиться, что у вашего пользователя есть права на запись в каталог перед запуском MongoDB.

При запуске сервер выведет информацию о версии и системе, а затем начнет ждать подключения. По умолчанию MongoDB прослушивает подключения к сокету на порту 27017. Сервер не сможет запуститься, если этот порт недоступен, – наиболее частой причиной этого является еще один экземпляр MongoDB, который уже запущен.

Всегда следует обезопасить свои экземпляры `mongod`. См. главу 19 для получения дополнительной информации.

Можно безопасно остановить *mongod*, набрав сочетание клавиш **Ctrl-C** в окружении с интерфейсом командной строки, из которой вы запускали сервер.

Для получения дополнительной информации о запуске или остановке MongoDB см. главу 21.

# Знакомство с оболочкой MongoDB

MongoDB поставляется с оболочкой JavaScript, которая позволяет взаимодействовать с экземпляром MongoDB из командной строки. Эта оболочка

полезна для выполнения административных функций, проверки работающего экземпляра или просто изучения MongoDB. Оболочка *mongo* является важным инструментом для использования MongoDB. Мы будем широко использовать ее на протяжении всей книги.

## Запуск оболочки

Чтобы запустить оболочку, запустите исполняемый файл *mongo*:

```
$ mongo
MongoDB shell version: 4.2.0
connecting to: test
>
```

Оболочка автоматически пытается подключиться к серверу MongoDB, работающему на локальной машине, при запуске, поэтому убедитесь, что вы запускаете *mongod* перед запуском оболочки.

Оболочка представляет собой полнофункциональный интерпретатор JavaScript, способный запускать произвольные программы на языке JavaScript. В качестве иллюстрации давайте выполним базовые математические вычисления:

```
> x = 200;
200
> x / 5;
40
```

Мы также можем использовать все стандартные библиотеки JavaScript:

```
> Math.sin(Math.PI / 2);
1
> new Date("20109/1/1");
ISODate("2019-01-01T05:00:00Z")
> "Hello, World!".replace("World", "MongoDB");
Hello, MongoDB!
```

И даже можем определять и вызывать функции JavaScript:

```
> function factorial (n) {
... if (n <= 1) return 1;
... return n * factorial(n - 1);
... }
> factorial(5);
120
```

Обратите внимание, что вы можете создавать многострочные команды. Оболочка будет определять, завершен ли оператор JavaScript, когда

вы нажимаете **Enter**. Если оператор не завершен, оболочка позволит вам продолжить запись в следующей строке. Нажав **Enter** три раза подряд, вы отмените наполовину сформированную команду и снова увидите приглашение к вводу >.

## Клиент MongoDB

Хотя возможность выполнения произвольного кода JavaScript полезна, реальная сила оболочки заключается в том, что она также является автономным клиентом MongoDB. При запуске оболочка подключается к базе данных *test* на сервере MongoDB и присваивает это подключение глобальной переменной `db`. Эта переменная является основной точкой доступа к вашему серверу MongoDB через оболочку.

Чтобы увидеть базу данных, которой присвоена переменная `db`, наберите `db` и нажмите **Enter**:

```
> db
test
```

Оболочка содержит некоторые дополнения, которые не являются допустимым синтаксисом в JavaScript, но они были реализованы по причине того, что они знакомы пользователям оболочек SQL. Эти дополнения не предоставляют никакой дополнительной функциональности, но являются хорошим синтаксическим сахаром. Например, одна из наиболее важных операций – выбор базы данных, которая будет использоваться:

```
> use video
switched to db video
```

Теперь если вы посмотрите на переменную `db`, то увидите, что она относится к базе данных *video*:

```
> db
Video
```

Поскольку это оболочка JavaScript, при вводе имени переменной имя будет оцениваться как выражение. После будет выведено значение (в данном случае имя базы данных).

Вы можете получить доступ к коллекциям из переменной `db`. Например:

```
> db.movies
```

возвращает коллекцию *movies* из текущей базы данных. Теперь, когда мы можем получить доступ к коллекции в оболочке, мы можем выполнить практически любую операцию с базой данных.