**Введение в MongoDB**

**Что такое MongoDB**

**Последнее обновление: 30.10.2015**

MongoDB реализует новый подход к построению баз данных, где нет таблиц, схем, запросов SQL, внешних ключей и многих других вещей, которые присущи объектно-реляционным базам данных.

Со времен динозавров было обычным делом хранить все данные в реляционных базах данных (MS SQL, MySQL, Oracle, PostgresSQL). При этом было не столь важно, а подходят ли реляционные базы данных для хранения данного типа данных или нет.

В отличие от реляционных баз данных MongoDB предлагает документо-ориентированную модель данных, благодаря чему MongoDB работает быстрее, обладает лучшей масштабируемостью, ее легче использовать.

Но, даже учитывая все недостатки традиционных баз данных и достоинства MongoDB, важно понимать, что задачи бывают разные и методы их решения бывают разные. В какой-то ситуации MongoDB действительно улучшит производительность вашего приложения, например, если надо хранить сложные по структуре данные. В другой же ситуации лучше будет использовать традиционные реляционные базы данных. Кроме того, можно использовать смешенный подход: хранить один тип данных в MongoDB, а другой тип данных - в традиционных БД.

Вся система MongoDB может представлять не только одну базу данных, находящуюся на одном физическом сервере. Функциональность MongoDB позволяет расположить несколько баз данных на нескольких физических серверах, и эти базы данных смогут легко обмениваться данными и сохранять целостность.

**Формат данных в MongoDB**

Одним из популярных стандартов обмена данными и их хранения является JSON (JavaScript Object Notation). JSON эффективно описывает сложные по структуре данные. Способ хранения данных в MongoDB в этом плане похож на JSON, хотя формально JSON не используется. Для хранения в MongoDB применяется формат, который называется **BSON** (БиСон) или сокращение от binary JSON.

BSON позволяет работать с данными быстрее: быстрее выполняется поиск и обработка. Хотя надо отметить, что BSON в отличие от хранения данных в формате JSON имеет небольшой недостаток: в целом данные в JSON-формате занимают меньше места, чем в формате BSON, с другой стороны, данный недостаток с лихвой окупается скоростью.

**Кроссплатформенность**

MongoDB написана на C++, поэтому ее легко портировать на самые разные платформы. MongoDB может быть развернута на платформах Windows, Linux, MacOS, Solaris. Можно также загрузить исходный код и самому скомпилировать MongoDB, но рекомендуется использовать библиотеки с офсайта.

**Документы вместо строк**

Если реляционные базы данных хранят строки, то MongoDB хранит документы. В отличие от строк документы могут хранить сложную по структуре информацию. Документ можно представить как хранилище ключей и значений.

**Ключ** представляет простую метку, с которым ассоциировано определенный кусок данных.

Однако при всех различиях есть одна особенность, которая сближает MongoDB и реляционные базы данных. В реляционных СУБД встречается такое понятие как **первичный ключ**. Это понятие описывает некий столбец, который имеет уникальные значения. В MongoDB для каждого документа имеется уникальный идентификатор, который называется \_id. И если явным образом не указать его значение, то MongoDB автоматически сгенерирует для него значение.

Каждому ключу сопоставляется определенное значение. Но здесь также надо учитывать одну особенность: если в реляционных базах есть четко очерченная структура, где есть поля, и если какое-то поле не имеет значение, ему (в зависимости от настроек конкретной бд) можно присвоить значение NULL. В MongoDB все иначе. Если какому-то ключу не сопоставлено значение, то этот ключ просто опускается в документе и не употребляется.

**Коллекции**

Если в традиционном мире SQL есть таблицы, то в мире MongoDB есть коллекции. И если в реляционных БД таблицы хранят однотипные жестко структурированные объекты, то в коллекции могут содержать самые разные объекты, имеющие различную структуру и различный набор свойств.

**Репликация**

Система хранения данных в MongoDB представляет набор реплик. В этом наборе есть основной узел, а также может быть набор вторичных узлов. Все вторичные узлы сохраняют целостность и автоматически обновляются вместе с обновлением главного узла. И если основной узел по каким-то причинам выходит из строя, то один из вторичных узлов становится главным.

**Простота в использовании**

Отсутствие жесткой схемы базы данных и в связи с этим потребности при малейшем изменении концепции хранения данных пересоздавать эту схему значительно облегчают работу с базами данных MongoDB и дальнейшим их масштабированием. Кроме того, экономится время разработчиков. Им больше не надо думать о пересоздании базы данных и тратить время на построение сложных запросов.

**GridFS**

Одной из проблем при работе с любыми системами баз данных является сохранение данных большого размера. Можно сохранять данные в файлах, используя различные языки программирования. Некоторые СУБД предлагают специальные типы данных для хранения бинарных данных в БД (например, BLOB в MySQL).

В отличие от реляционных СУБД MongoDB позволяет сохранять различные документы с различным набором данных, однако при этом размер документа ограничивается 16 мб. Но MongoDB предлагает решение - специальную технологию **GridFS**, которая позволяет хранить данные по размеру больше, чем 16 мб.

Система GridFS состоит из двух коллекций. В первой коллекции, которая называется files, хранятся имена файлов, а также их метаданные, например, размер. А в другой коллекции, которая называется chunks, в виде небольших сегментов хранятся данные файлов, обычно сегментами по 256 кб.

Для тестирования GridFS можно использовать специальную утилиту **mongofiles**, которая идет в пакете mongodb.

**Установка и начало работы с MongoDB на Windows**

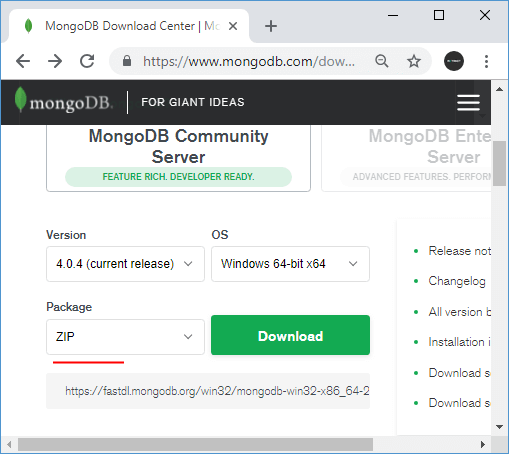
**Последнее обновление: 14.11.2018**

Для установки MongoDB загрузим один распространяемых пакетов с официального сайта <https://www.mongodb.com/download-center/community>.

Официальный сайт предоставляет пакеты дистрибутивов для различных платформ: Windows, Linux, MacOS, Solaris. И каждой платформы доступно несколько дистрибутивов. Причем есть два вида серверов - Community и Enterprise. В данном случае надо установить версию Community. Хотя Enterprise-версия обладает несколько большими возможностями, но она доступна только в триальном режиме или по подписке.

На момент написания данного материала последней версией платформы была версия **4.0.4**. Использование конкретной версии может несколько отличаться от применения иных версий платформы MongoDB.

Для загрузки нобходиомго функционала выберем нужную операционную систему и подходящий тип пакета:



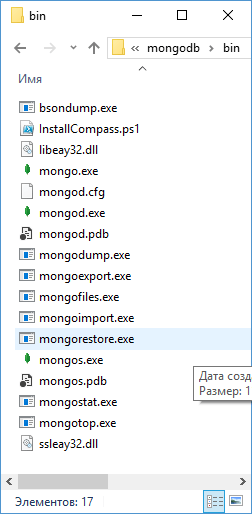
Для ОС Windows можно выбрать тип пакета "ZIP", то есть загрузить сервер в виде архива.

Если до установки уже была установлена более рання версия MongoDB, то ее необходимо удалить и также необходимо удалить все равне созданные базы данных.

После загрузки архивного пакета распакуем его в папку C:\mongodb.

**Содержимое пакета MongoDB**

Если после установки мы откроем папку *C:\mongodb\bin*, то сможем найти там кучу приложений, которые выполняют определенную роль. Вкратце рассмотрим их.



* **bsondump**: считывает содержимое BSON-файлов и преобразует их в читабельный формат, например, в JSON
* **mongo**: представляет консольный интерфейс для взаимодействия с базами данных, своего рода консольный клиент
* **mongod**: сервер баз данных MongoDB. Он обрабатывает запросы, управляет форматом данных и выполняет различные операции в фоновом режиме по управлению базами данных
* **mongodump**: утилита создания бэкапа баз данных
* **mongoexport**: утилита для экспорта данных в форматы JSON, TSV или CSV
* **mongofiles**: утилита, позволяющая управлять файлами в системе GridFS
* **mongoimport**: утилита, импорирующая данных в форматах JSON, TSV или CSV в базу данных MongoDB
* **mongorestore**: позволяет записывать данные из дампа, созданного mongodump, в новую или существующую базу данных
* **mongos**: служба маршрутизации MongoDB, которая помогает обрабатывать запросы и определять местоположение данных в кластере MongoDB
* **mongorestat**: представляет счетчики операций с бд
* **mongotop**: предоставляет способ подсчета времени, затраченного на операции чтения-записи в бд

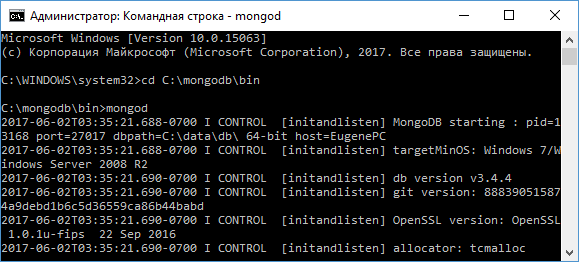
**Создание каталога для БД и запуск MongoDB**

После установки надо создать на жестком диске каталог, в котором будут находиться базы данных MongoDB.

В ОС Windows по умолчанию MongoDB хранит базы данных по пути *C:\data\db*, поэтому, если вы используете Windows, вам надо создать соответствующий каталог. В ОС Linux и MacOS каталогом по умолчанию будет */data/db*.

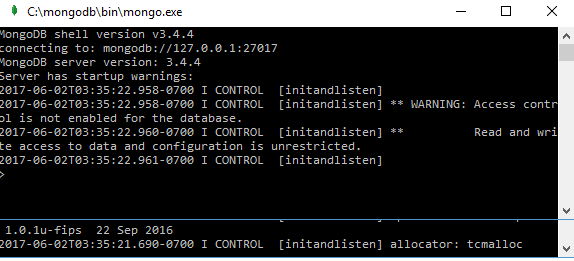
Если же возникла необходимость использовать какой-то другой путь к файлам, то его можно передать при запуске MongoDB во флаге --dbpath.

Итак, после создания каталога для хранения БД можно запустить сервер MongoDB. Сервер представляет приложение **mongod**, которое находится в папке bin. Для этого запустим командную строку (в Windows) или консоль в Linux и там введем соответствующие команды. Для ОС Windows это будет выглядеть так:



Командная строка отобразит нам ряд служебной информации, например, что сервер запускается на localhost на порту 27017.

И после удачного запуска сервера мы сможем производить операции с бд через оболочку **mongo**. Эта оболочка представляет файл **mongo.exe**, который располагается в выше рассмотренной папке установки. Запустим этот файл:



Это консольная оболочка для взаимодействия с сервером, через которую можно управлять данными. Второй строкой эта оболочка говорит о подключении к серверу mongod.

Теперь произведем какие-либо простейшие действия. Введем в mongo последовательно следующие команды и после каждой команды нажмем на Enter:

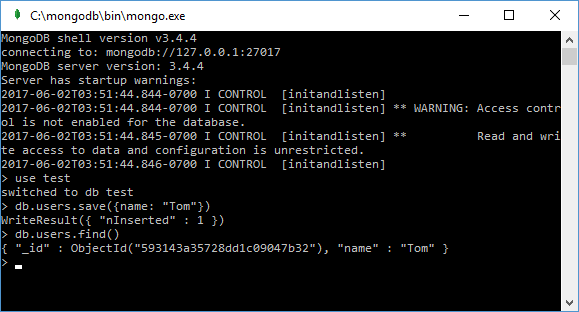
|  |  |
| --- | --- |
| 1  2  3 | use test  db.users.save( { name: "Tom" } )  db.users.find() |

Первая команда **use test** устанавливает в качестве используемой базу данных test. Даже если такой бд нет, то она создается автоматически. И далее db будет представлять текущую базу данных - то есть базу данных test. После db идет users - это коллекция, в которую затем мы добавляем новый объект. Если в SQL нам надо создавать таблицы заранее, то коллекции MongoDB создает самостоятельно при их отсутствии.

С помощью метода **db.users.save()** в коллекцию users базы данных test добавляется объект { name: "Tom" }. Описание добавляемого объекта определяется в формате, с которым вы возможно знакомы, если имели дело с форматом JSON. То есть в данном случае у объекта определен один ключ "name", которому сопоставляется значение "Tom". То есть мы добавляем пользователя с именем Tom.

Если объект был успешно добавлен, то консоль выведет результа в виде выражения WriteResult({ "nInserted" : 1 }).

А третья команда **db.users.find()** выводит на экран все объекты из бд test.



Из вывода вы можете увидеть, что к начальным значениям объекта было добавлено какое-то непонятно поле **ObjectId**. Как вы помните, MongoDB в качестве уникальных идентификаторов документа использует поле \_id. И в данном случае ObjectId как раз и представляет значение для идентификатора \_id.

**Установка драйверов MongoDB**

Конечно, мы можем работать и через консоль mongo, добавляя и отображая объекты в бд. Но нам также было бы неплохо, если бы mongoDB взаимодействовала бы с нашими приложениями, написанными на PHP, C++, C# и других языках программирования. И для этой цели нам потребуются специальные драйверы.

На офсайте на странице <https://docs.mongodb.com/ecosystem/drivers/> можно найти драйвера для таких языков программирования, как PHP, C++, C#, Java, Python, Perl, Ruby, Scala и др.

Далее уже, рассматривая взаимодействие отдельных языков программирования с MongoDB, мы подробнее рассмотрим установку и драйвера и всю необходимую конфигурацию для определенных языков программирования.

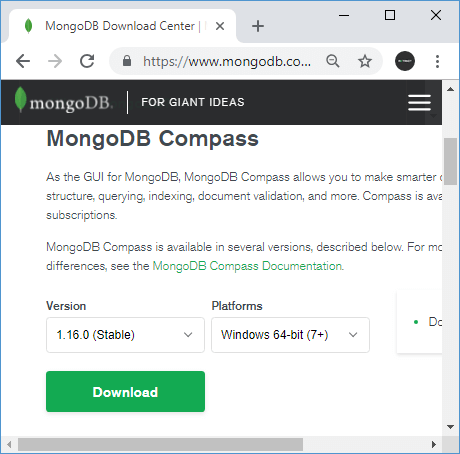
**Графический клиент Compass**

**Последнее обновление: 14.11.2018**

Для работы с MongoDB можно использовать официальный графический клиент **Compass**.

Если вдруг на этапе установки MongoDB, Mongo Compass не был установлен, то для его загрузки перейдем по адресу <https://www.mongodb.com/download-center/compass>.

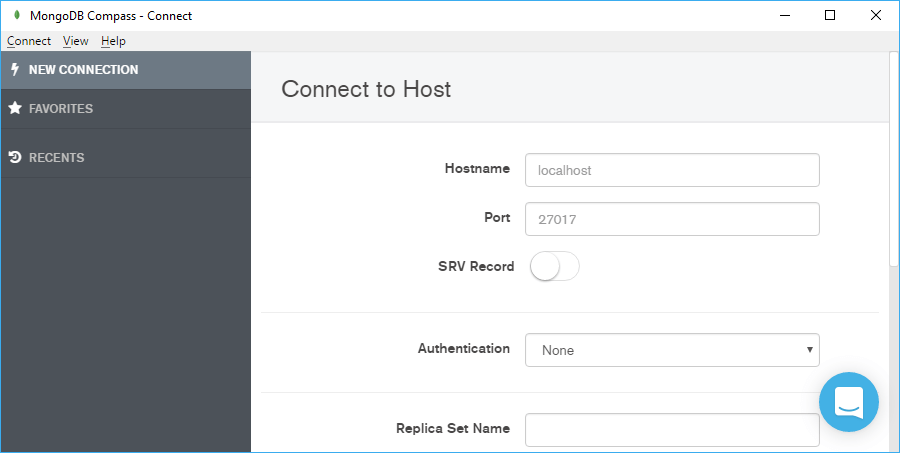
Укажем опции для загрузки - версию Compass и целевую операционную систему:

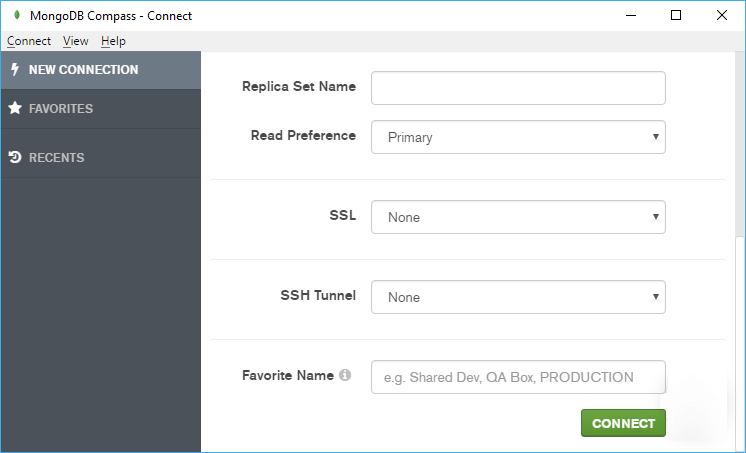


После загрузки программы установки запустим ее. Нам отобразится следующее окно, которое инфомирует о том, что произодится установка:



Установка производится очень быстро, и после ее окончания на рабочем столе и в меню Пуск (если наша ОС - Windows), отобразится соответствующая иконка. Кроме того, автоматически запустится сам Compass:



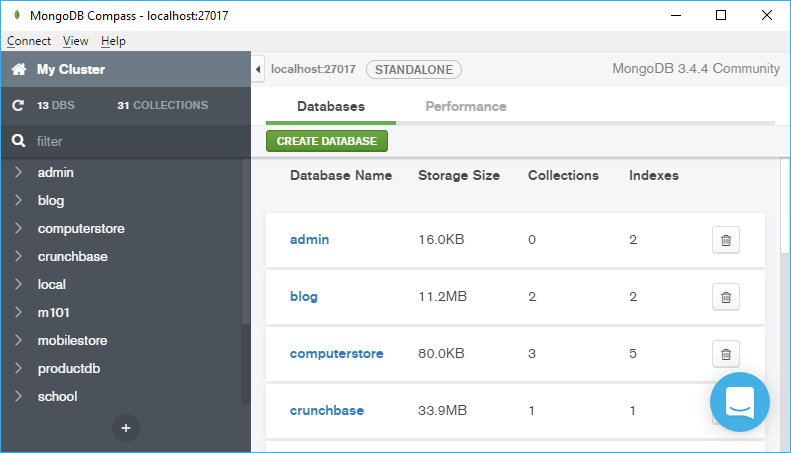


При запуске программы нам надо ввести ряд данных, чтобы подключиться к серверу.

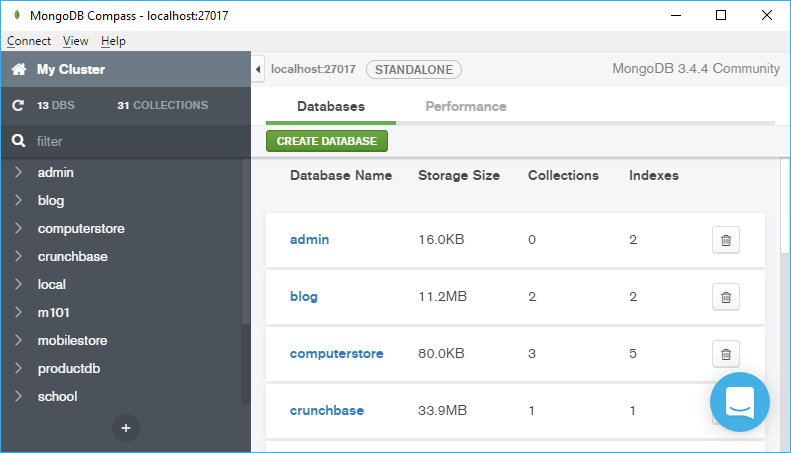
* **Hostname**: хост сервера
* **Port**: порт, по которому запущен сервер
* **SRV Record**: спецификация, которая определяет расположение сервера (например, адрес и порт).
* **Authentication**: тип применяемой аутентификации. Здеь есть следующие опции:
  + **None**
  + **Username / Password**
  + **X.509**
  + **Kerberos**
  + **LDAP**
* **Replica Set Name**: название реплики MongoDB, к которой происходит подключение
* **Read Preference**: определяет, как Compass управляет операциями чтения. Может принимать следующие опции: Primary, Primary Preferred, Secondary, Secondary Preferred и Nearest.
* **SSL**: указывает, будет ли использоваться защищенное подключение
* **SSH tunnel**: следует ли подключаться к кластеру MongoDB через туннель SSH
* **Favorite Name**: устанавливает имя подключения

Например, подключимся к локальному серверу. Для этого вначале запустим сам сервер MongoDB. В Compass оставим все настройки подключения по умолчанию (они как раз нацелены на локальный сервер, который запускается по адресу localhost:27017) и для подключения нажмем на кнопку Connect.

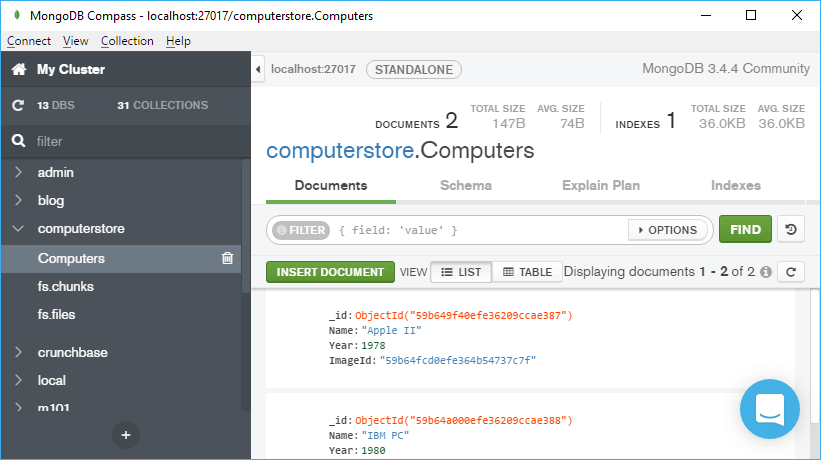
После этого нам откроется список баз данных, которые есть на сервере:



Мы можем выбрать определенную базу данных и получить по нему информацию, в частности, увидеть набор коллекций в бд, сколько они занимают данных.



Нажав на определенную коллекцию, можно увидеть графически все данные, которые есть в коллекции:



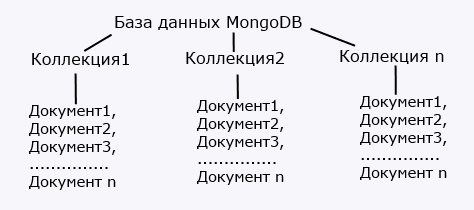
Используя графический интерфейс программы Compass, мы можем управлять этими данными, добавлять, изменять, удалять их.

**Работа с базой данных**

**Устройство базы данных. Документы**

**Последнее обновление: 25.03.2018**

Всю модель устройства базы данных в MongoDB можно представить следующим образом:



Если в реляционных бд содержимое составляют таблицы, то в mongodb база данных состоит из **коллекций**.

Каждая коллекция имеет свое уникальное имя - произвольный идентификатор, состоящий из не более чем 128 различных алфавитно-цифровых символов и знака подчеркивания.

В отличие от реляционных баз данных MongoDB не использует табличное устройство с четко заданным количеством столбцов и типов данных. MongoDB является документо-ориентированной системой, в которой центральным понятием является **документ**.

Документ можно представить как объект, хранящий некоторую информацию. В некотором смысле он подобен строкам в реляционных субд, где строки хранят информацию об отдельном элементе. Например, типичный документ:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | {      "name": "Bill",      "surname": "Gates",      "age": "48",      "company": {          "name" : "microsoft",          "year" : "1974",          "price" : "300000"          }  } |

Документ представляет набор пар ключ-значение. Например, в выражении "name": "Bill" name представляет ключ, а Bill - значение.

Ключи представляют строки. Значения же могут различаться по типу данных. В данном случае у нас почти все значения также представляют строковый тип, и лишь один ключ (company) ссылается на отдельный объект. Всего имеется следующие типы значений:

* **String**: строковый тип данных, как в приведенном выше примере (для строк используется кодировка UTF-8)
* **Array (массив)**: тип данных для хранения массивов элементов
* **Binary data (двоичные данные)**: тип для хранения данных в бинарном формате
* **Boolean**: булевый тип данных, хранящий логические значения TRUE или FALSE, например, {"married": FALSE}
* **Date**: хранит дату в формате времени Unix
* **Double**: числовой тип данных для хранения чисел с плавающей точкой
* **Integer**: используется для хранения целочисленных значений, например, {"age": 29}
* **JavaScript**: тип данных для хранения кода javascript
* **Min key/Max key**: используются для сравнения значений с наименьшим/наибольшим элементов BSON
* **Null**: тип данных для хранения значения Null
* **Object**: строковый тип данных, как в приведенном выше примере
* **ObjectID**: тип данных для хранения id документа
* **Regular expression**: применяется для хранения регулярных выражений
* **Symbol**: тип данных, идентичный строковому. Используется преимущественно для тех языков, в которых есть специальные символы.
* **Timestamp**: применяется для хранения времени

В отличие от строк документы могут содержать разнородную информацию. Так, рядом с документом, описанным выше, в одной коллекции может находиться другой объект, например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | {      "name": "Tom",      "birthday": "1985.06.28",      "place" : "Berlin",      "languages" :[          "english",          "german",          "spanish"      ]  } |

Казалось бы разные объекты за исключением отдельных свойств, но все они могут находиться в одной коллекции.

Еще пара важных замечаний: в MongoDB запросы обладают регистрозависимостью и строгой типизацией. То есть следующие два документа не будут идентичны:

|  |  |
| --- | --- |
| 1  2 | {"age" : "28"}  {"age" : 28} |

Если в первом случае для ключа age определена в качестве значения строка, то во втором случае значением является число.

**Идентификатор документа**

Для каждого документа в MongoDB определен уникальный идентификатор, который называется \_id. При добавлении документа в коллекцию данный идентификатор создается автоматически. Однако разработчик может сам явным образом задать идентификатор, а не полагаться на автоматически генерируемые, указав соответствующий ключ и его значение в документе.

Данное поле должно иметь уникальное значение в рамках коллекции. И если мы попробуем добавить в коллекцию два документа с одинаковым идентификатором, то добавится только один из них, а при добавлении второго мы получим ошибку.

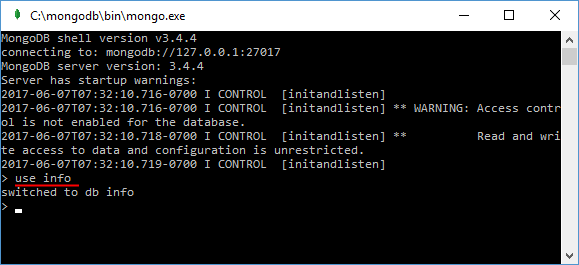
Если идентификатор не задан явно, то MongoDB создает специальное бинарное значение размером 12 байт. Это значение состоит из нескольких сегментов: значение типа timestamp размером 4 байта, идентификатор машины из 3 байт, идентификатор процесса из 2 байт и счетчик из 3 байт. Таким образом, первые 9 байт гарантируют уникальность среди других машин, на которых могут быть реплики базы данных. А следующие 3 байта гарантируют уникальность в течение одной секунды для одного процесса. Такая модель построения идентификатора гарантирует с высокой долей вероятности, что он будет иметь уникальное значение, ведь она позволяет создавать до 16 777 216 уникальных объектов ObjectId в секунду для одного процесса.

**Установка и администрирование базы данных**

**Последнее обновление: 25.03.2018**

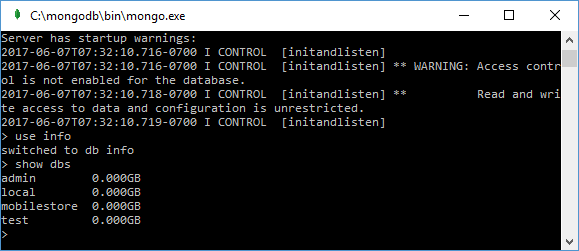
Начиная работать с MongoDB, первым делом надо установить нужную нам базу данных в качестве текущей, чтобы затем ее использовать. Для этого надо использовать команду use, после которой идет название базы данных. При этом не важно, существует ли такая бд или нет. Если ее нет, то MongoDB автоматически создаст ее при добавлении в нее данных. Например, запустим mongo.exe и введем там следующую команду:

|  |  |
| --- | --- |
| 1 | > use info |



Теперь в качестве текущей будет установлена БД info.

Если вы вдруг не уверены, а существует ли уже база данных с таким названием, то с помощью команды show dbs можно вывести названия всех имеющихся бд на консоль:



Для базы данных можно задать любое имя, однако есть некоторые ограничения. Например, в имени не должно быть символов /, \, ., ", \*, <, >, :, |, ?, $. Кроме того, имена баз данных ограничены 64 байтами.

Также есть зарезервированные имена, которые нельзя использовать: local, admin, config.

Причем как вы видите, бд info в данном списке нет, так как я в нее еще не добавил данные.

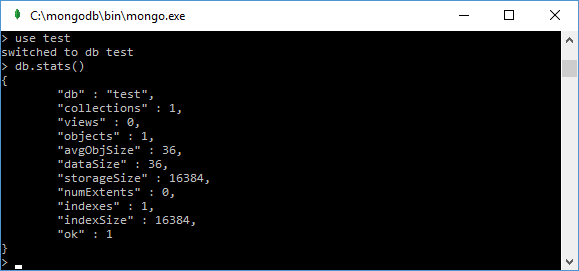
Если мы хотим узнать, какая бд используется в текущей момент, то мы можем воспользоваться командой db:

|  |  |
| --- | --- |
| 1  2 | > db  info |

Кроме баз данных мы можем просмотреть список всех коллекций в текущей бд с помощью команды show collections. Но так как бд info, которая указана текущей, еще не существует, то и коллекций она пока не содержит.

**Получение статистики**

Используя команду db.stats(), можно получить статистику по текущей базе данных. Например, у нас в качестве текущей установлена база данных test:



Похожим образом мы можем узнать всю статистику по отдельной коллекции. Например, узнаем статистику по коллекции users: db.users.stats()

**Добавление данных**

**Последнее обновление: 25.03.2018**

Установив бд, теперь мы можем добавить в нее данные. Все данные хранятся в бд в формате BSON, который близок к JSON, поэтому нам надо также вводить данные в этом формате. И хотя у нас, возможно, на данный момент нет ни одной коллекции, но при добавлении в нее данных она автоматически создается.

Как ранее говорилось, имя коллекции - произвольный идентификатор, состоящий из не более чем 128 различных алфавитно-цифровых символов и знака подчеркивания. В то же время имя коллекции не должно начинаться с префикса system., так как он зарезервирован для внутренних коллекций (например, коллекция system.users содержит всех пользователей базы данных). И также имя не должно содержать знака доллара - $.

Для добавления в коллекцию могут использоваться три ее метода:

* **insertOne()**: добавляет один документ
* **insertMany()**: добавляет несколько документов
* **insert()**: может добавлять как один, так и несколько документов

Итак, добавим один документ:

|  |  |
| --- | --- |
| 1 | > db.users.insertOne({"name": "Tom", "age": 28, languages: ["english",  "spanish"]}) |

Документ представляет набор пар ключ-значение. В данном случае добавляемый документ имеет три ключа: name, age, languages, и каждому из них сопоставляет определенное значение. Например, ключу languages в качесте значения сопоставляется массив.

Некоторые ограничения при использовании имен ключей:

* Символ $ не может быть первым символом в имени ключа
* Имя ключа не может содержать символ точки .

При добавлении данных, если мы явным образом не предоставили значение для поля "\_id" (то есть уникального идентификатора документа), то оно генерируется автоматически. Но в принципе мы можем сами установить этот идентификатор при добавлении данных:

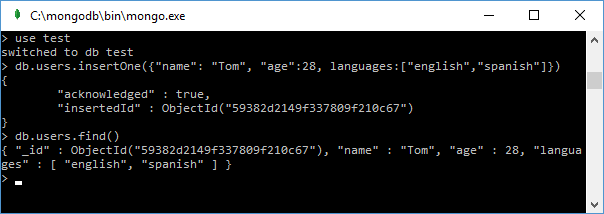
|  |  |
| --- | --- |
| 1 | > db.users.insertOne({"\_id": 123457, "name": "Tom", "age": 28,  languages: ["english", "spanish"]}) |
|  |  |

Стоит отметить, что названия ключей могут использоваться в кавычках, а могут и без кавычек.

В случае удачного добавления на консоль будет выведен идентификатор добавленного документа.

И чтобы убедиться, что документ в бд, мы его выводим функцией find.

|  |  |
| --- | --- |
| 1 | > db.users.find() |



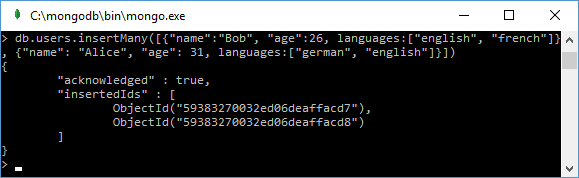
Чтобы вывести в более читабельном виде добавим метод pretty():

|  |  |
| --- | --- |
| 1 | > db.users.find().pretty() |

Если надо добавить ряд документов, то мы можем воспользоваться методом **insertMany()**:

|  |  |
| --- | --- |
| 1  2 | > db.users.insertMany([{"name": "Bob", "age": 26, languages: ["english",  "frensh"]},  {"name": "Alice", "age": 31, languages:["german", "english"]}]) |

После добавления консоль выводит идентификаторы добавленных документов:



И третий метод - **insert()** демонстрирует более универсальный способ добавления документов. При его вызове в него также передается добавляемый документ:

|  |  |
| --- | --- |
| 1 | > db.users.insert({"name": "Tom", "age": 28, languages: ["english",  "spanish"]}) |

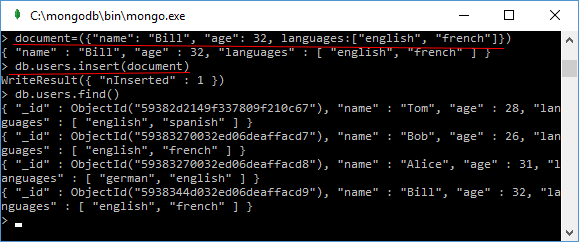
После его вызова на консоль выводится количество добавленных записей:

|  |  |
| --- | --- |
| 1 | WriteResult({ "nInserted" : 1 }) |

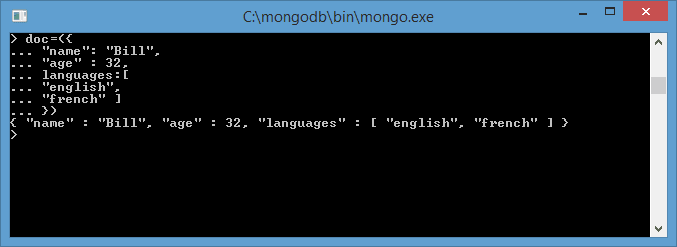
Есть еще один способ добавления в бд документа, который включает два этапа: определение документа (document = ( { ... } )) и собственно его добавление:

|  |  |
| --- | --- |
| 1  2 | > document=({"name": "Bill", "age": 32, languages: ["english", "french"]})  > db.users.insert(document) |

При желании опять же можно с помощью функции db.users.find() убедиться, что документ попал в бд.



Возможно, не всем будет удобно вводить в одну строчку все пары ключей и свойств. Но интеллектуальный интерпретатор MongoDB на основе javascript позволяет также вводить и многострочные команды. Если выражение не закончено (с точки зрения языка JavaScript), и вы нажимаете Enter, то ввод следующей части выражения автоматически переносится на следующую строку:



**Загрузка данных из файла**

Данные для базы данных mongodb можно определять в обычном текстовом файле, что довольно удобно, поскольку мы можем переносить или пересылать этот файл независимо от базы данных mongodb. Например, определим где-нибудь на жестком диске файл **users.js** со следующим содержимым:

|  |  |
| --- | --- |
| 1  2  3  4  5 | db.users.insertMany([  {"name": "Alice", "age": 31, languages: ["english", "french"]},  {"name": "Lene", "age": 29, languages: ["english", "spanish"]},  {"name": "Kate", "age": 30, languages: ["german", "russian"]}  ]) |

То есть здесь с помощью метода insertMany добавляются три документа в коллекцию users.

Для загузки файла в текущую базу данных применяется функция **load()**, в которую в качестве параметра передается путь к файлу:

|  |  |
| --- | --- |
| 1 | > load("D:/users.js") |

В данном случае предполагается, что файл располагается по пути "D:/users.js".