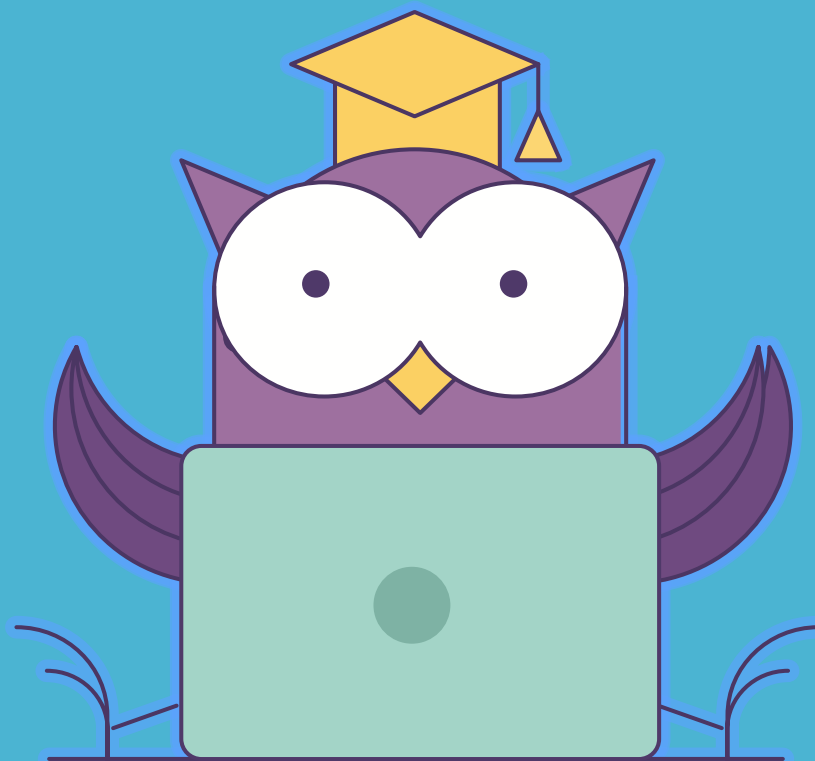




ОНЛАЙН-ОБРАЗОВАНИЕ

Как меня слышно и видно?



> Напишите в чат

+ если все хорошо

- если есть проблемы со звуком или с видео

!проверить запись!

NoSQL базы данных

Дмитрий Смаль



- Обзор NoSQL
- Встроенные базы данных, Bolt
- Key-Value базы данных, Redis
- Документные базы данных, MongoDB



Embedded база - просто библиотека, которую использует ваш проект (микросервис).

Примеры:

- BerkleyDB
- SQLite
- LevelDB (LSM tree) / RocksDB
- BoltDB (Go)

Плюсы:

- Быстрые, нет сетевых запросов
- Простота использования и развертывания

Минусы:

- Ограничение по объему данных - 1 сервер
- Ограничение по производительности - 1 сервер
- Время жизни данных = время жизни микросервиса

Область применения:

- Кеширование данных
- Временное хранение данных
- Для небольших сервисов под ручным управлением
- Для распределенных систем (peer-to-peer)

BoltDB позволяет хранить пары ключ/значени (byte[]), сгруппированные по бакетам.

Подключение BoltDB

```
import (  
    "log"  
  
    "github.com/boltdb/bolt"  
)  
  
func main() {  
    // Open the my.db data file in your current directory.  
    // It will be created if it doesn't exist.  
    db, err := bolt.Open("my.db", 0600, nil)  
    if err != nil {  
        log.Fatal(err)  
    }  
    defer db.Close()  
  
    ...  
}
```


Bolt позволяет 1 - RW транзакцию и N RO транзакций.

RW транзакция:

```
err := db.Update(func(tx *bolt.Tx) error {  
    ...  
    return nil // nil - commit, err - rollback  
})
```

RO транзакция:

```
err := db.View(func(tx *bolt.Tx) error {  
    ...  
    return nil  
})
```

```
db.Update(func(tx *bolt.Tx) error {
    b, err := tx.CreateBucket([]byte("MyBucket"))
    if err != nil {
        return fmt.Errorf("create bucket: %s", err)
    }
    return nil
})
```

Методы транзакции:

```
// получить бакет по имени
tx.Bucket(name []byte) *bolt.Bucket

// создать бакет
tx.CreateBucket(name []byte) (*bolt.Bucket, error)
tx.CreateBucketIfNotExists(name []byte) (*bolt.Bucket, error)

// удалить бакет
tx.DeleteBucket(name []byte) error

// перебрать все бакеты
tx.ForEach(fn func(name []byte, b *Bucket) error) error
```

Установка ключа:

```
db.Update(func(tx *bolt.Tx) error {  
    b := tx.Bucket([]byte("MyBucket"))  
    err := b.Put([]byte("answer"), []byte("42"))  
    return err  
})
```

Получение ключа:

```
var value []byte  
db.View(func(tx *bolt.Tx) error {  
    b := tx.Bucket([]byte("MyBucket"))  
    value = b.Get([]byte("answer"))  
    return nil  
})
```

Методы для работы с ключами:

```
b.Get(key []byte) []byte  
b.Put(key []byte, value []byte) error  
b.Delete(key []byte) error  
b.ForEach(fn func(k, v []byte) error) error
```

```
db.View(func(tx *bolt.Tx) error {
    b := tx.Bucket([]byte("MyBucket"))
    c := b.Cursor() // *sql.Cursor

    for k, v := c.First(); k != nil; k, v = c.Next() {
        fmt.Printf("key=%s, value=%s\n", k, v)
    }
    return nil
})
```

Методы курсора:

```
// перемещение к первому и последнему ключу в бакете
c.First() (key []byte, value []byte)
c.Last() (key []byte, value []byte)

// перемещение к следующему и предыдущему ключам
c.Next() (key []byte, value []byte)
c.Prev() (key []byte, value []byte)
```

```
db.View(func(tx *bolt.Tx) error {
    c := tx.Bucket([]byte("Events")).Cursor()

    // минимальное и максимальное значение ключа
    min := []byte("1990-01-01T00:00:00Z")
    max := []byte("2000-01-01T00:00:00Z")

    // прерывать итерацию нужно самостоятельно сравнивая ключи
    for k, v := c.Seek(min); k != nil && bytes.Compare(k, max) <= 0; k, v = c.Next() {
        fmt.Printf("%s: %s\n", k, v)
    }

    return nil
})
```

Метод `Seek` у курсора ищет ключ `key` или следующий за ним, если `key` не найден.

```
c.Seek(key []byte) (key []byte, value []byte)
```

- Используется mmap
- Управление памятью переложено на OS
- При превышении объема вероятен OOM

- in-memory
- key-value (rich value)
- client-server
- persistent



Плюсы:

- Скорость и низкая задержка
- Простота использования и развертывания

Минусы:

- Ограничение по объему - In-Memory
- Отсутствие Range Scan и сложных запросов

Область применения:

- Кеширование данных
- Write-heavy сервисы с простой моделью данных
- Вспомогательное хранилище для счетчиков и прочего

Основа Redis - большой словарь Key-Value.

На любой ключ можно установить TTL.

Value может быть сложным типом:

- String
- Hash
- List
- Set
- SortedSet
- HyperLogLog
- Streams
- Pub/Sub

<https://redis.io/commands>

```
import "github.com/go-redis/redis"

client := redis.NewClient(&redis.Options{
    Addr:      "your.redis.server.net:6379",
    Password: "",
})

resp, err := client.Ping().Result()
fmt.Println(resp, err)
```

<https://redis.io/commands#string>

```
// сохранение значения
err := client.Set("key", "value", 3*time.Second).Err()
if err != nil {
    log.Fatal(err)
}

// получение строки
val, err := client.Get("key").Result() // (string, error)
if err == redis.Nil {
    fmt.Println("key was not found")
} else if err != nil {
    log.Fatal(err)
}
fmt.Println("key", val)

// получение числа
ival, err := client.Get("nkey").Int() // (int, error)
```

<https://redis.io/commands#hash>

```
// сохранение хэша
err := client.HMSet("key", map[string]interface{}{
    "name": "Vasily",
    "age": 42,
}, 3*time.Second).Err()
if err != nil {
    log.Fatal(err)
}

// изменение отдельного значения
err := client.HSet("key", "age", 43).Result()
if err != nil {
    log.Fatal(err)
}

// получение ключа
age, err := client.HGet("key", "age").Int() // (int, error)

// получение всех ключей
obj, err := client.HGetAll("key").Result() // (map[string]string, error)
```

<https://redis.io/commands#pubsub>

Отправка сообщений

```
err := client.Publish("tasks", "some_data").Err()
```

Подписка и получение сообщений

```
sub := client.Subscribe("tasks", "another_chan")
_, err := sub.Receive() // подключиться к каналу redis
if err != nil {
    log.Fatal(err)
}

subCh := sub.Channel() // <-chan *Message
for message := range subCh {
    data := message.Payload // string
    // пример отписки
    if (data == "stop") {
        err := sub.Unsubscribe(message.Channel)
    }
}
```

HyperLogLog - структура данных, позволяющая с (требуемой) вероятностью и точностью определить кол-во элементов в множестве, не сохраняя сами элементы.

С вероятностью > 95% в этом множестве 15000 +/- 100 элементов

Пример: посчитать количество уникальных IP адресов за день/месяц/год.

Добавление элемента:

```
err := client.PFAdd("uniq_today", "8.8.8.8").Err()
```

Получение мощности множества:

```
n, err := client.PFCount("uniq_today").Result() // int, error
```

- single-threaded
- in-memory
- snapshots via fork (RDB)
- logs (AOF)

- Документная база данных
- Полно-функциональная
- Мощный язык запросов
- Масштабируемость



MongoDB хранит документы произвольной структуры, сгруппированные по коллекциям

```
{
  "_id": ObjectId("00000000000000000000000042"),
  "name": "Vasily",
  "age": 42,
  "car": {
    "mark": "tesla",
    "number": "123456y",
  }
}
```

MongoDB позволяет индексировать документы по произвольному полю и осуществлять опциональную валидацию.

BSON - надмножество JSON, с дополнительными типами данных и компактным бинарным форматом.

Типы данных в BSON

- `string` — строка
- `int` — целое число
- `double` — число с плавающей запятой
- `DateTime` — дата
- `byte[]` — массив байтов
- `bool` — булевы (True и False)
- `null` — NULL
- `BsonObject` — BSON-объект
- `BsonObject[]` — массив BSON-объектов

Установка:

```
$ sudo apt-get install mongodb
```

Подключение из консоли (по умолчанию - access control отключен)

```
$ mongo
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.3
>
```

По умолчанию mongodb работает без аутентификации.

Как включить: <https://docs.mongodb.com/manual/tutorial/enable-authentication/>

```
$ mongo
```

```
> show databases
```

```
admin    0.000GB
```

```
config   0.000GB
```

```
local    0.000GB
```

```
test     0.000GB
```

```
> use newdb
```

```
switched to db newdb
```

```
> db.createCollection("foo")
```

```
{ "ok" : 1 }
```

```
> use events
switched to db events

> db.createCollection("events", { "validator": {
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["owner", "title", "startdate"],
    "properties": {
      "owner": { "bsonType": "int" },
      "title": { "bsonType": "string" }
    }
  }
} })
{ "ok" : 1 }

> db.getCollection("events").createIndex({"owner": 1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```

```
import (
    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"
)

// создаем клиент
dsn := "mongodb://user1:pass1@localhost:27017"
options := options.Client().ApplyURI(uri)
client, err := mongo.NewClient(options)
if err != nil {
    // ...
}

// подключаемся
ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
defer cancel()
err = client.Connect(ctx)
if err != nil {
    // ...
}
```

```
type Event struct {
    Owner      int
    Title      string
    StartDate  time.Time
}

// документ для вставки
event := Event{
    Owner:      42,
    Title:      "new year",
    StartDate:  time.Now(),
}

// объект для работы с коллекцией events
coll := client.Database("db1").Collection("events")

// добавляем объект
res, err := coll.InsertOne(ctx, event)
if err != nil {
    // ...
}

// ID созданного объекта
fmt.Println(res.InsertedID)
```

```
import "go.mongodb.org/mongo-driver/bson"

// D - Document
update := bson.D{
    {"$inc", bson.D{
        {"members", 1},
    }},
}

// Один
filter = bson.D{{ "_id": "5d6e2f4ada476e65e5a5ba42" }} // res.InsertedID
updateResult, err := coll.UpdateOne(ctx, filter, update)
if err != nil {
    log.Fatal(err)
}

// Несколько
filter = bson.D{{ "owner", 42 }}
updateResult, err = coll.UpdateMany(ctx, filter, update)
if err != nil {
    log.Fatal(err)
}
```



```
var event Event

// предикат поиска
filter := bson.D{{ "_id": "5d6e2f4ada476e65e5a5ba42" }}

// выполняем запрос и сканируем в структуру
err := coll.FindOne(ctx, filter).Decode(&event)
if err != nil {
    log.Fatal(err)
}
fmt.Printf("%+v\n", event)
```

```
options := options.Find()
options.SetLimit(2)
filter = bson.D{{ "owner", 42 }}
var events []*Event

// выполняем запрос
cur, err := coll.Find(ctx, filter, options)
if err != nil {
    log.Fatal(err)
}
defer cur.Close(ctx)
// получаем документы
for cur.Next(ctx) {
    var event Event
    err := cur.Decode(&event)
    if err != nil {
        log.Fatal(err)
    }
    events = append(events, &event)
}
// проверяем ошибку получения
if err := cur.Err(); err != nil {
    log.Fatal(err)
}
```

Заполните пожалуйста опрос

<https://otus.ru/polls/5041/>



Спасибо за внимание!

