

Если вы видите что-то необычное, просто сообщите мне.

Тестикулировать

- [Simple nats](#)
- [Neo4j + triplet](#)
- [Running list](#)
- [ponzu](#)

Simple nats

<https://nats.io/>

<https://github.com/nats-io/nats.go>

Neo4j + triplet

```
package main

import (
    "context"
    "github.com/neo4j/neo4j-go-driver/v5/neo4j"
    "log"
    "os"
    "fmt"
)

type Triplet struct {
    Object string
    Predicate string
    Subject string
}

func main() {
    ctx := context.TODO()
    uri := "bolt://localhost:7687"
    username := "neo4j"
    password := "neo4jneo4j"
    var driver neo4j.DriverWithContext
    var session neo4j.SessionWithContext
    var triplet Triplet
    triplet.Object = os.Args[1]
    triplet.Predicate = os.Args[2]
    triplet.Subject = os.Args[3]
    driver, err := createDriver(uri, username, password)
    if err != nil {
        log.Fatal(err)
    }
    session, err = createSession(ctx, driver)
    if err != nil {
        log.Fatal(err)
    }
}
```

```

    message, err := createTriplet(ctx, triplet, session)
    if err != nil {
        log.Fatal(err)
    }
    log.Println("message: ", message)
    defer driver.Close(ctx)
    defer session.Close(ctx)

}

func createDriver(uri, username, password string) (neo4j.DriverWithContext, error) {
    driver, err := neo4j.NewDriverWithContext(uri, neo4j.BasicAuth(username, password, ""))
    if err != nil {
        return nil, err
    }
    return driver, nil
}

func createSession(ctx context.Context, driver neo4j.DriverWithContext) (neo4j.SessionWithContext, error) {
    session := driver.NewSession(ctx, neo4j.SessionConfig{AccessMode: neo4j.AccessModeWrite})
    return session, nil
}

func createTriplet(ctx context.Context, t Triplet, session neo4j.SessionWithContext) (string, error) {
    var template = "MERGE (o:Node {name: \"%s\"}) " +
        "MERGE (s:Node {name: \"%s\"}) " +
        "MERGE (o)-[:%s]-(s)"

    request := fmt.Sprintf(template, t.Object, t.Subject, t.Predicate)

    _, err := session.ExecuteWrite(
        ctx,
        func(transaction neo4j.ManagedTransaction) (any, error) {
            result, err := transaction.Run(ctx, request, map[string]any{})
            if err != nil {
                return nil, err
            }

            if result.Next(ctx) {

```

```
    return result.Record().Values[0], nil
```

```
  }
```

```
  return nil, result.Err()
```

```
})
```

```
if err != nil {
```

```
  return "", err
```

```
}
```

```
return "", nil
```

```
}
```

Running list

```
//show.go

package main

import (
    "fmt"
    "time"
)

func PrintTasks(tasks []Task) {
    printCanvas()
    for _, t := range tasks {
        printTasks(t)
    }
}

func printCanvas() {
    day := time.Now()
    fmt.Printf("\tsun\tmon\ttue\twen\tthu\tfri\tSAT\t Weeknumber %d\n", (day.YearDay() / 7))
    fmt.Printf("\t#####\n")
}

func printTasks(t Task) {
    day := time.Now()
    week := [7]int{0, 0, 0, 0, 0, 0, 0}
    switch t.Done {
    case -1:
        for i := t.CreateDay; i < day.Weekday(); i++ {
            week[i] = 2
        }
    default:
        for i := t.CreateDay; i < t.Done; i++ {
            week[i] = 2
        }
        week[t.Done] = 1
    }
}
```

```
}
```

```
fmt.Printf(
    "%t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\n",
    printDays(week[0]),
    printDays(week[1]),
    printDays(week[2]),
    printDays(week[3]),
    printDays(week[4]),
    printDays(week[5]),
    printDays(week[6]),
    t.Description,
)
}
```

```
func printDays(i int) string {
    switch i {
    case 1:
        return "[X]"
    case 2:
        return "[ ]"
    default:
        return "  "
    }
}
```

```
//struct.go
package main
```

```
import "time"
```

```
type Task struct {
    Description string
    Done        time.Weekday
    CreateDay   time.Weekday
    Week        int
}
```

```
func (t *Task) NewTask(description string) {
    day := time.Now()
```

```
t.Description = description
t.Done = -1
t.CreateDay = day.Weekday()
t.Week = day.YearDay() / 7
}
```

```
//main.go
package main

import "time"

func main() {
    week := time.Now().YearDay()
    tasks := []Task{
        Task{
            Description: "Test 123",
            Done:        time.Wednesday,
            CreateDay:    time.Monday,
            Week:        week,
        },
        Task{
            Description: "Test 32",
            Done:        -1,
            CreateDay:    time.Monday,
            Week:        week,
        },
        Task{
            Description: "Test 32",
            Done:        time.Tuesday,
            CreateDay:    time.Tuesday,
            Week:        week,
        },
        Task{
            Description: "Test 32",
            Done:        time.Monday,
            CreateDay:    time.Sunday,
            Week:        week,
        },
    }
    PrintTasks(tasks)
}
```



```
}
```

```
package main
```

```
import (
```

```
    "log"
```

```
    "net"
```

```
)
```

```
func echoServer(c net.Conn) {
```

```
    for {
```

```
        buf := make([]byte, 512)
```

```
        nr, err := c.Read(buf)
```

```
        if err != nil {
```

```
            return
```

```
        }
```

```
        data := buf[0:nr]
```

```
        println("Server got:", string(data))
```

```
        _, err = c.Write(data)
```

```
        if err != nil {
```

```
            log.Fatal("Write: ", err)
```

```
        }
```

```
    }
```

```
}
```

```
func main() {
```

```
    l, err := net.Listen("unix", "/tmp/echo.sock")
```

```
    if err != nil {
```

```
        log.Fatal("listen error:", err)
```

```
    }
```

```
    for {
```

```
        fd, err := l.Accept()
```

```
        if err != nil {
```

```
            log.Fatal("accept error:", err)
```

```
        }
```

```
        go echoServer(fd)
```

```
}  
}
```

```
package main
```

```
import (  
    "io"  
    "log"  
    "net"  
    "time"  
)
```

```
func reader(r io.Reader) {  
    buf := make([]byte, 1024)  
    for {  
        n, err := r.Read(buf[:])  
        if err != nil {  
            return  
        }  
        println("Client got:", string(buf[0:n]))  
    }  
}
```

```
func main() {  
    c, err := net.Dial("unix", "/tmp/echo.sock")  
    if err != nil {  
        panic(err)  
    }  
    defer c.Close()  
  
    go reader(c)  
    for {  
        _, err := c.Write([]byte("hi"))  
        if err != nil {  
            log.Fatal("write error:", err)  
            break  
        }  
        time.Sleep(1e9)  
    }  
}
```

```
package main
```

```
import "github.com/martinlindhe/notify"
```

```
func main() {
```

```
    // show a notification
```

```
    notify.Notify("app name", "notice", "some text", "path/to/icon.png")
```

```
    // show a notification and play a alert sound
```

```
    notify.Alert("app name", "alert", "some text", "path/to/icon.png")
```

```
}
```

ponzu

<https://github.com/ponzu-cms/ponzu>