Useful libreries in Go

Index

- cli
- gjson
- resty
- errors
- gin
- gorm
- go-redis
- go kit
- decimal
- go.uuid
- logrus
- mock
- testify

Cli

cli is a simple, fast, and fun package for building command line apps in Go.

The usage is simple:

```
package main

import (
    "fmt"
    "os"

    "github.com/urfave/cli"
)

func main() {
    app := cli.NewApp()
    app.Name = "greet"
    app.Usage = "fight the loneliness!"
    app.Action = func(c *cli.Context) error {
        fmt.Println("Hello friend!")
        return nil
    }

    app.Run(os.Args)
}
```

Parameters for the app:

- Name: The name of the app
- Usage: A short description of what the app does
- Action: The function to call when the app is run
- Flags: A list of flags to parse
- Commands: A list of commands to run
- Version: The version of the app
- Authors: A list of authors of the app
- HelpName: The name of the help command
- UsageText: The text to display in the help output
- HideHelp: A boolean to hide the help command
- HideVersion: A boolean to hide the version flag

An example of a command:

```
package main

import (
    "fmt"
    "os"

    "github.com/urfave/cli"
)

func main() {
    app := cli.NewApp()
    app.Flags = []cli.Flag {
        cli.StringFlag {
            Name: "lang",
            Value: "english",
            Usage: "language for the greeting",
        },
    }
    app.Action = func(c *cli.Context) error {
        name := "someone"
        if c.NArg() > 0 {
            name = c.Args()[0]
        }
        if c.String("lang") == "spanish" {
            fmt.Println("Hola", name)
        } else {
            fmt.Println("Hello", name)
        }
        return nil
    }

    app.Run(os.Args)
}
```

Gjson

gjson is a fast and simple way to get values from a JSON document. It supports JSONPath syntax.

```
package main

import (
    "fmt"

    "github.com/tidwall/gjson"
)

func main() {
    json := `{"name":{"first":"Janet","last":"Prichard"},"age":47}`
    value := gjson.Get(json, "name.last")
    fmt.Println(value.String())
}
```

Methods for gjson:

- Get: Get the value of a path
- GetMany: Get the values of many paths
- Set: Set the value of a path
- Exists: Check if a path exists
- Delete: Delete a path
- Type: Get the type of a path
- String: Get the string value of a path
- Int: Get the int value of a path
- Float: Get the float value of a path
- Bool: Get the bool value of a path
- Array: Get the array value of a path
- Map: Get the map value of a path
- Value: Get the value of a path
- Raw: Get the raw value of a path
- Search: Search for a path
- Iter: Iterate over all paths
- Parse: Parse a JSON document
- Valid: Check if a JSON document is valid

Generate a JSON document from a map:

```
package main

import (
    "fmt"

    "github.com/tidwall/gjson"
)

func main() {
    json := gjson.Parse(`{"name":{"first":"Janet","last":"Prichard"},"age":47}`)
    fmt.Println(json.Get("name.last").String())
}
```

Resty

resty is a HTTP client library for Go, which is inspired by Ruby's rest-client.

```
package main

import (
    "fmt"
    "log"

    "github.com/go-resty/resty/v2"
)

func main() {
    client := resty.New()

    resp, err := client.R().
        SetHeader("Accept", "application/json").
        Get("https://httpbin.org/get")
    if err != nil {
        log.Fatal(err)
    }

    fmt.Println(resp)

}
```

An example of a POST request:

```
package main

import (
    "fmt"
    "log"
    "github.com/go-resty/resty/v2"
)

func main() {
    client := resty.New()

    resp, err := client.R().
        SetHeader("Accept", "application/json").
        SetBody(map[string]string{
            "name": "John",
            "age":  "25",
        }).
        Post("https://httpbin.org/post")
    if err != nil {
        log.Fatal(err)
    }

    fmt.Println(resp)

}
```

An example of a PUT request:

```
package main

import (
    "fmt"
    "log"
    "github.com/go-resty/resty/v2"
)

func main() {
    client := resty.New()

    resp, err := client.R().
        SetHeader("Accept", "application/json").
        SetBody(map[string]string{
            "name": "John",
            "age":  "25",
        }).
        Put("https://httpbin.org/put")
    if err != nil {
        log.Fatal(err)
    }

    fmt.Println(resp)

}
```

An example of a DELETE request:

```
package main

import (
    "fmt"
    "log"
    "github.com/go-resty/resty/v2"
)

func main() {
    client := resty.New()

    resp, err := client.R().
        SetHeader("Accept", "application/json").
        Delete("https://httpbin.org/delete")
    if err != nil {
        log.Fatal(err)
    }

    fmt.Println(resp)

}
```

Gorm

GORM is the most used ORM library for Golang, aims to be developer friendly. It's well documented, full-featured, and has first-class support for PostgreSQL, MySQL, SQLite, SQL Server and Oracle.

An example of a SQLite database:

```
package main

import (
    "fmt"
    "log"

    "gorm.io/driver/sqlite"
    "gorm.io/gorm"
)

type User struct {
    gorm.Model
    Name string
    Age  uint
}

func main() {
    db, err := gorm.Open(sqlite.Open("test.db"), &gorm.Config{})
    if err != nil {
        log.Fatal(err)
    }

    db.AutoMigrate(&User{})

    db.Create(&User{Name: "Jinzhu", Age: 18})
    db.Create(&User{Name: "Jinzhu 2", Age: 20})

    var user User
    db.First(&user, 1) // find product with integer primary key
    db.First(&user, "name = ?", "Jinzhu 2") // find product with code D42

    fmt.Println(user)
}
```

Gorm with Postgres

```
package main

import (
    "fmt"
    "log"

    "gorm.io/driver/postgres"
    "gorm.io/gorm"
)

type User struct {
    gorm.Model
    Name string
    Age  uint
}

func main() {
    dsn := "host=localhost user=gorm password=gorm dbname=gorm port=9920 sslmode=disable
TimeZone=Asia/Shanghai"
    db, err := gorm.Open(postgres.Open(dsn), &gorm.Config{})
    if err != nil {
        log.Fatal(err)
    }

    db.AutoMigrate(&User{})

    db.Create(&User{Name: "Jinzhu", Age: 18})
    db.Create(&User{Name: "Jinzhu 2", Age: 20})

    var user User
    db.First(&user, 1) // find product with integer primary key
    db.First(&user, "name = ?", "Jinzhu 2") // find product with code D42

    fmt.Println(user)
}
```

Main features

- Auto Migrations
- Associations
- Transactions
- Hooks
- Preload
- Query
- Raw SQL
- Logger
- Callbacks
-

Eager Loading

- Soft Delete
- Composite Primary Key
- Composite Foreign Key

Auto Migrations

Auto Migrations will automatically create/update/delete tables to match your struct definitions.

```
package main

import (
    "fmt"
    "log"

    "gorm.io/driver/sqlite"
    "gorm.io/gorm"
)

type User struct {
    gorm.Model
    Name string
    Age  uint
}

func main() {
    db, err := gorm.Open(sqlite.Open("test.db"), &gorm.Config{})
    if err != nil {
        log.Fatal(err)
    }

    db.AutoMigrate(&User{})
}
```

Associations

GORM supports all the common associations: has one, has many, belongs to, many to many.

```
package main

import (
    "fmt"
    "log"

    "gorm.io/driver/sqlite"
    "gorm.io/gorm"
)

type User struct {
    gorm.Model
    Name string
    Age  uint
    Pets []Pet
}

type Pet struct {
    gorm.Model
    Name string
    Age  uint
    User uint
}

func main() {
    db, err := gorm.Open(sqlite.Open("test.db"), &gorm.Config{})
    if err != nil {
        log.Fatal(err)
    }

    db.AutoMigrate(&User{}, &Pet{})

    user := User{
        Name: "Jinzhu",
        Age:  18,
        Pets: []Pet{
            {Name: "Galeone", Age: 3},
            {Name: "Naruto", Age: 1},
        },
    }
    db.Create(&user)
}
```

Transactions

Transactions allow you to run multiple operations as a single atomic operation. If any operation fails, all operations will be rolled back.

```
package main

import (
    "fmt"
    "log"

    "gorm.io/driver/sqlite"
    "gorm.io/gorm"
)

type User struct {
    gorm.Model
    Name string
    Age  uint
}

func main() {
    db, err := gorm.Open(sqlite.Open("test.db"), &gorm.Config{})
    if err != nil {
        log.Fatal(err)
    }

    db.AutoMigrate(&User{})

    tx := db.Begin()
    if tx.Error != nil {
        log.Fatal(tx.Error)
    }

    if err := tx.Create(&User{Name: "Jinzhu", Age: 18}).Error; err != nil {
        tx.Rollback()
        log.Fatal(err)
    }

    if err := tx.Create(&User{Name: "Jinzhu 2", Age: 20}).Error; err != nil {
        tx.Rollback()
        log.Fatal(err)
    }

    tx.Commit()
}
```

Hooks

Hooks allow you to execute code before or after certain events.

```
package main

import (
    "fmt"
    "log"

    "gorm.io/driver/sqlite"
    "gorm.io/gorm"
)

type User struct {
    gorm.Model
    Name string
    Age  uint
}

func main() {
    db, err := gorm.Open(sqlite.Open("test.db"), &gorm.Config{})
    if err != nil {
        log.Fatal(err)
    }

    db.AutoMigrate(&User{})

    db.Callback().Create().Before("gorm:create").Register("my_create:before", func(db
*gorm.DB) {
        fmt.Println("before create")
    })

    db.Callback().Create().After("gorm:create").Register("my_create:after", func(db *gorm.DB)
{
        fmt.Println("after create")
    })

    db.Create(&User{Name: "Jinzhu", Age: 18})
}
```