-

gmapgmap(safetrue,)

```
m := gmap.New(true)
```

gmapgoframe


```
package main

import (
    "fmt"
    "github.com/gogf/gf/v2/container/gmap"
)

func main() {
    // gmap
    // gmap
    // true
    m := gmap.New()

    //
    for i := 0; i < 10; i++ {
        m.Set(i, i)
    }
    //
    fmt.Println(m.Size())
    // ()
    m.Sets(map[interface{}]interface{}{
        10 : 10,
        11 : 11,
    })
    fmt.Println(m.Size())

    //
    fmt.Println(m.Contains(1))

    //
    fmt.Println(m.Get(1))

    //
    m.Remove(9)
    fmt.Println(m.Size())

    //
    m.Removes([]interface{}{10, 11})
    fmt.Println(m.Size())

    // ()
    fmt.Println(m.Keys())
    // ()
    fmt.Println(m.Values())

    //
    fmt.Println(m.GetOrSet(100, 100))

    //
    fmt.Println(m.Remove(100))
```

```
    // map
    m.Iterator(func(k interface{}, v interface{}) bool {
        fmt.Printf("%v:%v ", k, v)
        return true
    })

    //
    m.LockFunc(func(m map[interface{}]interface{}) {
        m[99] = 99
    })

    //
    m.RLockFunc(func(m map[interface{}]interface{}) {
        fmt.Println(m[99])
    })

    // map
    m.Clear()

    // map
    fmt.Println(m.IsEmpty())
}
```

```
10
12
true
1
11
9
[0 1 2 4 6 7 3 5 8]
[3 5 8 0 1 2 4 6 7]
100
100
3:3 5:5 8:8 7:7 0:0 1:1 2:2 4:4 6:6 99
true
```

map

```
package main

import (
    "fmt"
    "github.com/gogf/gf/v2/frame/g"
    "github.com/gogf/gf/v2/container/gmap"
    "github.com/gogf/gf/v2/util/gutil"
)

func main() {
    array   := g.Slice{2, 3, 1, 5, 4, 6, 8, 7, 9}
    hashMap := gmap.New(true)
    listMap := gmap.NewListMap(true)
    treeMap := gmap.NewTreeMap(gutil.ComparatorInt, true)
    for _, v := range array {
        hashMap.Set(v, v)
    }
    for _, v := range array {
        listMap.Set(v, v)
    }
    for _, v := range array {
        treeMap.Set(v, v)
    }
    fmt.Println("HashMap   Keys:", hashMap.Keys())
    fmt.Println("HashMap Values:", hashMap.Values())
    fmt.Println("ListMap   Keys:", listMap.Keys())
    fmt.Println("ListMap Values:", listMap.Values())
    fmt.Println("TreeMap   Keys:", treeMap.Keys())
    fmt.Println("TreeMap Values:", treeMap.Values())
}
```

```
HashMap   Keys: [4 6 8 7 9 2 3 1 5]
HashMap Values: [6 8 4 3 1 5 7 9 2]
ListMap   Keys: [2 3 1 5 4 6 8 7 9]
ListMap Values: [2 3 1 5 4 6 8 7 9]
TreeMap   Keys: [1 2 3 4 5 6 7 8 9]
TreeMap Values: [1 2 3 4 5 6 7 8 9]
```

FilterEmpty/FilterNil

```
package main

import (
        "fmt"
        "github.com/gogf/gf/v2/container/gmap"
        "github.com/gogf/gf/v2/frame/g"
)

func main() {
        m1 := gmap.NewFrom(g.MapAnyAny{
                "k1": "",
                "k2": nil,
                "k3": 0,
                "k4": 1,
        })
        m2 := gmap.NewFrom(g.MapAnyAny{
                "k1": "",
                "k2": nil,
                "k3": 0,
                "k4": 1,
        })
        m1.FilterEmpty()
        m2.FilterNil()
        fmt.Println(m1.Map())
        fmt.Println(m2.Map())

        // Output:
        // map[k4:1]
        // map[k1: k3:0 k4:1]
}
```

Flip

```
package main

import (
        "fmt"
        "github.com/gogf/gf/v2/container/gmap"
        "github.com/gogf/gf/v2/frame/g"
)

func main() {
        var m gmap.Map
        m.Sets(g.MapAnyAny{
                "k1": "v1",
                "k2": "v2",
        })
        m.Flip()
        fmt.Println(m.Map())

        // May Output:
        // map[v1:k1 v2:k2]
}
```

Keys/Values/

```
package main

import (
        "fmt"
        "github.com/gogf/gf/v2/container/gmap"
        "github.com/gogf/gf/v2/frame/g"
)

func main() {
        var m gmap.Map
        m.Sets(g.MapAnyAny{
                "k1": "v1",
                "k2": "v2",
                "k3": "v3",
                "k4": "v4",
        })
        fmt.Println(m.Keys())
        fmt.Println(m.Values())

        // May Output:
        // [k1 k2 k3 k4]
        // [v2 v3 v4 v1]
}
```

Pop/Pops

```
package main

import (
        "fmt"
        "github.com/gogf/gf/v2/container/gmap"
        "github.com/gogf/gf/v2/frame/g"
)

func main() {
        var m gmap.Map
        m.Sets(g.MapAnyAny{
                "k1": "v1",
                "k2": "v2",
                "k3": "v3",
                "k4": "v4",
        })
        fmt.Println(m.Pop())
        fmt.Println(m.Pops(2))
        fmt.Println(m.Size())

        // May Output:
        // k1 v1
        // map[k2:v2 k4:v4]
        // 1
}
```

SetIfNotExist*

truefalse

- SetIfNotExist
- SetIfNotExistFunc
- SetIfNotExistFuncLock

```
package main

import (
        "fmt"
        "github.com/gogf/gf/v2/container/gmap"
)

func main() {
        var m gmap.Map
        fmt.Println(m.SetIfNotExist("k1", "v1"))
        fmt.Println(m.SetIfNotExist("k1", "v1"))
        fmt.Println(m.Map())

        // Output:
        // true
        // false
        // map[k1:v1]
}
```

Merge

```
package main

import (
        "fmt"
        "github.com/gogf/gf/v2/container/gmap"
)

func main() {
        var m1, m2 gmap.Map
        m1.Set("key1", "val1")
        m2.Set("key2", "val2")
        m1.Merge(&m2)
        fmt.Println(m1.Map())

        // May Output:
        // map[key1:val1 key2:val2]
}
```

JSON/

gmapjson/

1. Marshal

```
package main

import (
    "encoding/json"
    "fmt"
    "github.com/gogf/gf/v2/frame/g"

    "github.com/gogf/gf/v2/container/gmap"
)

func main() {
    m := gmap.New()
    m.Sets(g.MapAnyAny{
        "name":  "john",
        "score": 100,
    })
    b, _ := json.Marshal(m)
    fmt.Println(string(b))
}
```

```
{"name":"john","score":100}
```

2.Unmarshal

```
package main


import (
    "encoding/json"
    "fmt"
    "github.com/gogf/gf/v2/container/gmap"
)


func main() {
    m := gmap.Map{}
    s := []byte(`{"name":"john","score":100}`)
    json.Unmarshal(s, &m)
    fmt.Println(m.Map())
}
```

```
map[name:john score:100]
```