-

https://pkg.go.dev/github.com/gogf/gf/v2/container/gmap

New

- NewAnyAnyMapsafemapfalse
-

```
New(safe ...bool) *Map
```

-

```
func ExampleNew() {
        m := gmap.New()

        // Add data.
        m.Set("key1", "val1")

        // Print size.
        fmt.Println(m.Size())

        addMap := make(map[interface{}]interface{})
        addMap["key2"] = "val2"
        addMap["key3"] = "val3"
        addMap[1] = 1

        fmt.Println(m.Values())

        // Batch add data.
        m.Sets(addMap)

        // Gets the value of the corresponding key.
        fmt.Println(m.Get("key3"))

        // Get the value by key, or set it with given key-value if
not exist.
        fmt.Println(m.GetOrSet("key4", "val4"))

        // Set key-value if the key does not exist, then return
true; or else return false.
        fmt.Println(m.SetIfNotExist("key3", "val3"))

        // Remove key
        m.Remove("key2")
        fmt.Println(m.Keys())

        // Batch remove keys.
        m.Removes([]interface{}{"key1", 1})
        fmt.Println(m.Keys())

        // Contains checks whether a key exists.
        fmt.Println(m.Contains("key3"))

        // Flip exchanges key-value of the map, it will change key-
value to value-key.
        m.Flip()
        fmt.Println(m.Map())

        // Clear deletes all data of the map.
        m.Clear()

        fmt.Println(m.Size())

        // May Output:
        // 1
        // [val1]
        // val3
        // val4
        // false
        // [key4 key1 key3 1]
        // [key4 key3]
        // true
        // map[val3:key3 val4:key4]
        // 0
}
```

NewFrom

- NewFrommapAnyAnyMap
- mapmapsafefalse
-

```
NewFrom(data map[interface{}]interface{}, safe ...bool) *Map
```

-

```
func ExampleNewFrom() {
        m := gmap.New()

        m.Set("key1", "val1")
        fmt.Println(m)

        n := gmap.NewFrom(m.MapCopy(), true)
        fmt.Println(n)

        // Output:
        // {"key1":"val1"}
        // {"key1":"val1"}
}
```

Iterator

- Iteratorfhashmapftruefalse
-

```
Iterator(f func(k interface{}, v interface{}) bool)
```

-

```
func ExampleAnyAnyMap_Iterator() {
        m := gmap.New()
        for i := 0; i < 10; i++ {
                m.Set(i, i*2)
        }

        var totalKey, totalValue int
        m.Iterator(func(k interface{}, v interface{}) bool {
                totalKey += k.(int)
                totalValue += v.(int)

                return totalKey < 10
        })

        fmt.Println("totalKey:", totalKey)
        fmt.Println("totalValue:", totalValue)

        // May Output:
        // totalKey: 11
        // totalValue: 22
}
```

Clone

- CloneAnyAnyMapmap
-

```
Clone(safe ...bool) *AnyAnyMap
```

-

```
func ExampleAnyAnyMap_Clone() {
        m := gmap.New()

        m.Set("key1", "val1")
        fmt.Println(m)

        n := m.Clone()
        fmt.Println(n)

        // Output:
        // {"key1":"val1"}
        // {"key1":"val1"}
}
```

Map

- Map map
-
-

```
Map() map[interface{}]interface{}
```

-

```
func ExampleAnyAnyMap_Map() {
        // non concurrent-safety, a pointer to the underlying data
        m1 := gmap.New()
        m1.Set("key1", "val1")
        fmt.Println("m1:", m1)

        n1 := m1.Map()
        fmt.Println("before n1:", n1)
        m1.Set("key1", "val2")
        fmt.Println("after n1:", n1)

        // concurrent-safety, copy of underlying data
        m2 := gmap.New(true)
        m2.Set("key1", "val1")
        fmt.Println("m1:", m2)

        n2 := m2.Map()
        fmt.Println("before n2:", n2)
        m2.Set("key1", "val2")
        fmt.Println("after n2:", n2)

        // Output:
        // m1: {"key1":"val1"}
        // before n1: map[key1:val1]
        // after n1: map[key1:val2]
        // m1: {"key1":"val1"}
        // before n2: map[key1:val1]
        // after n2: map[key1:val1]
}
```

MapCopy

- MapCopymap
-

```
MapCopy() map[interface{}]interface{}
```

-

```
func ExampleAnyAnyMap_MapCopy() {
        m := gmap.New()

        m.Set("key1", "val1")
        m.Set("key2", "val2")
        fmt.Println(m)

        n := m.MapCopy()
        fmt.Println(n)

        // Output:
        // {"key1":"val1","key2":"val2"}
        // map[key1:val1 key2:val2]
}
```

MapStrAny

- `MapStrAnymap[string]interface{}map`
-

```
MapStrAny() map[string]interface{}
```

-

```
func ExampleAnyAnyMap_MapStrAny() {
        m := gmap.New()
        m.Set(1001, "val1")
        m.Set(1002, "val2")

        n := m.MapStrAny()
        fmt.Println(n)

        // Output:
        // map[1001:val1 1002:val2]
}
```

FilterEmpty

- `FilterEmpty`: `0`, `nil`, `false`, `""`, `len(slice/map/chan) == 0`
-

```
FilterEmpty()
```

-

```
func ExampleAnyAnyMap_FilterEmpty() {
        m := gmap.NewFrom(g.MapAnyAny{
                "k1": "",
                "k2": nil,
                "k3": 0,
                "k4": 1,
        })
        m.FilterEmpty()
        fmt.Println(m.Map())

        // Output:
        // map[k4:1]
}
```

FilterNil

- FilterNilnil
-

```
FilterNil()
```

-

```
func ExampleAnyAnyMap_FilterNil() {
        m := gmap.NewFrom(g.MapAnyAny{
                "k1": "",
                "k2": nil,
                "k3": 0,
                "k4": 1,
        })
        m.FilterNil()
        fmt.Println(m.Map())

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

Set

- Setmapkey/value
-

```
Set(key interface{}, value interface{})
```

-

```
func ExampleAnyAnyMap_Set() {
        m := gmap.New()

        m.Set("key1", "val1")
        fmt.Println(m)

        // Output:
        // {"key1":"val1"}
}
```

Sets

- Setsmapkey/value
-

```
Sets(data map[interface{}]interface{})
```

-

```
func ExampleAnyAnyMap_Sets() {
        m := gmap.New()

        addMap := make(map[interface{}]interface{})
        addMap["key1"] = "val1"
        addMap["key2"] = "val2"
        addMap["key3"] = "val3"

        m.Sets(addMap)
        fmt.Println(m)

        // Output:
        // {"key1":"val1","key2":"val2","key3":"val3"}
}
```

Search

- Searchkeymapkeyfoundtruefalse
-

```
Search(key interface{}) (value interface{}, found bool)
```

-

```
func ExampleAnyAnyMap_Search() {
        m := gmap.New()

        m.Set("key1", "val1")

        value, found := m.Search("key1")
        if found {
                fmt.Println("find key1 value:", value)
        }

        value, found = m.Search("key2")
        if !found {
                fmt.Println("key2 not find")
        }

        // Output:
        // find key1 value: val1
        // key2 not find
}
```

Get

- GetkeyvaluekeyNil
-

```
Get(key interface{}) (value interface{})
```

-

```
func ExampleAnyAnyMap_Get() {
        m := gmap.New()

        m.Set("key1", "val1")

        fmt.Println("key1 value:", m.Get("key1"))
        fmt.Println("key2 value:", m.Get("key2"))

        // Output:
        // key1 value: val1
        // key2 value: <nil>
}
```

Pop

- Popmap
-

```
Pop() (key, value interface{})
```

-

```
func ExampleAnyAnyMap_Pop() {
        var m gmap.Map
        m.Sets(g.MapAnyAny{
                "k1": "v1",
                "k2": "v2",
                "k3": "v3",
                "k4": "v4",
        })

        fmt.Println(m.Pop())

        // May Output:
        // k1 v1
}
```

Pops

- Popsmapsizesize == -1
-

```
Pops(size int) map[interface{}]interface{}
```

-

```
func ExampleAnyAnyMap_Pops() {
        var m gmap.Map
        m.Sets(g.MapAnyAny{
                "k1": "v1",
                "k2": "v2",
                "k3": "v3",
                "k4": "v4",
        })
        fmt.Println(m.Pops(-1))
        fmt.Println("size:", m.Size())

        m.Sets(g.MapAnyAny{
                "k1": "v1",
                "k2": "v2",
                "k3": "v3",
                "k4": "v4",
        })
        fmt.Println(m.Pops(2))
        fmt.Println("size:", m.Size())

        // May Output:
        // map[k1:v1 k2:v2 k3:v3 k4:v4]
        // size: 0
        // map[k1:v1 k2:v2]
        // size: 2
}
```

GetOrSet

- GetOrSetkeyvaluekeykeyvaluemap
-

```
GetOrSet(key interface{}, value interface{}) interface{}
```

-

```
func ExampleAnyAnyMap_GetOrSet() {
        m := gmap.New()
        m.Set("key1", "val1")

        fmt.Println(m.GetOrSet("key1", "NotExistValue"))
        fmt.Println(m.GetOrSet("key2", "val2"))

        // Output:
        // val1
        // val2
}
```

GetOrSetFunc

- GetOrSetFunckeyvaluekeykeyfunc f map
-

```
GetOrSetFunc(key interface{}, f func() interface{}) interface{}
```

-

```
func ExampleAnyAnyMap_GetOrSetFunc() {
        m := gmap.New()
        m.Set("key1", "val1")

        fmt.Println(m.GetOrSetFunc("key1", func() interface{} {
                return "NotExistValue"
        }))
        fmt.Println(m.GetOrSetFunc("key2", func() interface{} {
                return "NotExistValue"
        }))

        // Output:
        // val1
        // NotExistValue
}
```

GetOrSetFuncLock

- GetOrSetFunckeyvaluekeykeyfunc f map
- GetOrSetFuncLockGetOrSetFuncf
-

```
GetOrSetFuncLock(key interface{}, f func() interface{}) interface{}
```

-

```
func ExampleAnyAnyMap_GetOrSetFuncLock() {
        m := gmap.New()
        m.Set("key1", "val1")

        fmt.Println(m.GetOrSetFuncLock("key1", func() interface{} {
                return "NotExistValue"
        }))
        fmt.Println(m.GetOrSetFuncLock("key2", func() interface{} {
                return "NotExistValue"
        }))

        // Output:
        // val1
        // NotExistValue
}
```

GetVar

- GetVarkey*gvar.Var
-

```
GetVar(key interface{}) *gvar.Var
```

-

```
func ExampleAnyAnyMap_GetVar() {
        m := gmap.New()
        m.Set("key1", "val1")

        fmt.Println(m.GetVar("key1"))
        fmt.Println(m.GetVar("key2").IsNil())

        // Output:
        // val1
        // true
}
```

GetVarOrSet

- GetVarOrSetkeyvalue/*gvar.Var
-

```
GetVarOrSet(key interface{}, value interface{}) *gvar.Var
```

-

```
func ExampleAnyAnyMap_GetVarOrSet() {
        m := gmap.New()
        m.Set("key1", "val1")

        fmt.Println(m.GetVarOrSet("key1", "NotExistValue"))
        fmt.Println(m.GetVarOrSet("key2", "val2"))

        // Output:
        // val1
        // val2
}
```

GetVarOrSetFunc

- GetVarOrSetFunckeyfunc f/*gvar.Var
-

```
GetVarOrSetFunc(key interface{}, f func() interface{}) *gvar.Var
```

-

```
func ExampleAnyAnyMap_GetVarOrSetFunc() {
        m := gmap.New()
        m.Set("key1", "val1")

        fmt.Println(m.GetVarOrSetFunc("key1", func() interface{} {
                return "NotExistValue"
        }))
        fmt.Println(m.GetVarOrSetFunc("key2", func() interface{} {
                return "NotExistValue"
        }))

        // Output:
        // val1
        // NotExistValue
}
```

GetVarOrSetFuncLock

- GetVarOrSetFuncLockkeyfunc f/*gvar.Var
- GetVarOrSetFuncLockGetVarOrSetFuncfgoroutinef
-

```
GetVarOrSetFuncLock(key interface{}, f func() interface{}) *gvar.Var
```

-

```
func ExampleAnyAnyMap_GetVarOrSetFuncLock() {
        m := gmap.New()
        m.Set("key1", "val1")

        fmt.Println(m.GetVarOrSetFuncLock("key1", func() interface{}
{
                return "NotExistValue"
        }))
        fmt.Println(m.GetVarOrSetFuncLock("key2", func() interface{}
{
                return "NotExistValue"
        }))

        // Output:
        // val1
        // NotExistValue
}
```

SetIfNotExist

- keySetIfNotExistmapkey/valuetruekeyfalsevalue
-

```
SetIfNotExist(key interface{}, value interface{}) bool
```

-

```
func ExampleAnyAnyMap_SetIfNotExist() {
        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]
}
```

SetIfNotExistFunc

- keySetIfNotExistFuncmapftruekeyfalsevalue
-

```
SetIfNotExistFunc(key interface{}, f func() interface{}) bool
```

-

```
func ExampleAnyAnyMap_SetIfNotExistFunc() {
        var m gmap.Map
        fmt.Println(m.SetIfNotExistFunc("k1", func() interface{} {
                return "v1"
        }))
        fmt.Println(m.SetIfNotExistFunc("k1", func() interface{} {
                return "v1"
        }))
        fmt.Println(m.Map())

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

SetIfNotExistFuncLock

- keySetIfNotExistFuncmapfunc ctruekeyfalsevalue
- SetIfNotExistFuncLockSetIfNotExistFuncmutex.Lockf
-

```
SetIfNotExistFuncLock(key interface{}, f func() interface{}) bool
```

-

```
func ExampleAnyAnyMap_SetIfNotExistFuncLock() {
        var m gmap.Map
        fmt.Println(m.SetIfNotExistFuncLock("k1", func() interface{}
{
                return "v1"
        }))
        fmt.Println(m.SetIfNotExistFuncLock("k1", func() interface{}
{
                return "v1"
        }))
        fmt.Println(m.Map())

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

Remove

- keymapvaluevalue
-

```
Remove(key interface{}) (value interface{})
```

-

```
func ExampleAnyAnyMap_Remove() {
        var m gmap.Map
        m.Set("k1", "v1")

        fmt.Println(m.Remove("k1"))
        fmt.Println(m.Remove("k2"))

        // Output:
        // v1
        // <nil>
}
```

Removes

- Removeskeymapvalue
-

```
Removes(keys []interface{})
```

-

```
func ExampleAnyAnyMap_Removes() {
        var m gmap.Map
        m.Sets(g.MapAnyAny{
                "k1": "v1",
                "k2": "v2",
                "k3": "v3",
                "k4": "v4",
        })

        removeList := make([]interface{}, 2)
        removeList = append(removeList, "k1")
        removeList = append(removeList, "k2")

        m.Removes(removeList)

        fmt.Println(m.Map())

        // Output:
        // map[k3:v3 k4:v4]
}
```

Keys

- Keysmapkeyslice
-

```
Keys() []interface{}
```

-

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

        // Output:
        // [k1 k2 k3 k4]
}
```

Values

- Valuesmapvalueslice
-

```
Values() []interface{}
```

-

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

        // May Output:
        // [v1 v2 v3 v4]
}
```

Contains

- Containskeykeytruefalse
- interface{}
-

```
Contains(key interface{}) bool
```

-

```
func ExampleAnyAnyMap_Contains() {
        var m gmap.Map
        m.Sets(g.MapAnyAny{
                "k1": "v1",
                "k2": "v2",
                "k3": "v3",
                "k4": "v4",
        })
        fmt.Println(m.Contains("k1"))
        fmt.Println(m.Contains("k5"))

        // Output:
        // true
        // false
}
```

Size

- Sizemap
-

```
Size() int
```

-

```
func ExampleAnyAnyMap_Size() {
        var m gmap.Map
        m.Sets(g.MapAnyAny{
                "k1": "v1",
                "k2": "v2",
                "k3": "v3",
                "k4": "v4",
        })

        fmt.Println(m.Size())

        // Output:
        // 4
}
```

IsEmpty

- IsEmptymapmaptruefalse
-

```
IsEmpty() bool
```

-

```
func ExampleAnyAnyMap_IsEmpty() {
        var m gmap.Map
        fmt.Println(m.IsEmpty())

        m.Set("k1", "v1")
        fmt.Println(m.IsEmpty())

        // Output:
        // true
        // false
}
```

Clear

- Clearmap
-

```
Clear()
```

-

```
func ExampleAnyAnyMap_Clear() {
        var m gmap.Map
        m.Sets(g.MapAnyAny{
                "k1": "v1",
                "k2": "v2",
                "k3": "v3",
                "k4": "v4",
        })

        m.Clear()

        fmt.Println(m.Map())

        // Output:
        // map[]
}
```

Replace

- Replacedatamapvalue
-

```
Replace(data map[interface{}]interface{})
```

-

```
func ExampleAnyAnyMap_Replace() {
        var m gmap.Map
        m.Sets(g.MapAnyAny{
                "k1": "v1",
        })

        var n gmap.Map
        n.Sets(g.MapAnyAny{
                "k2": "v2",
        })

        fmt.Println(m.Map())

        m.Replace(n.Map())
        fmt.Println(m.Map())

        n.Set("k2", "v1")
        fmt.Println(m.Map())

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

LockFunc

- LockFuncf
-

```
LockFunc(f func(m map[interface{}]interface{}))
```

-

```
func ExampleAnyAnyMap_LockFunc() {
        var m gmap.Map
        m.Sets(g.MapAnyAny{
                "k1": 1,
                "k2": 2,
                "k3": 3,
                "k4": 4,
        })

        m.LockFunc(func(m map[interface{}]interface{}) {
                totalValue := 0
                for _, v := range m {
                        totalValue += v.(int)
                }
                fmt.Println("totalValue:", totalValue)
        })

        // Output:
        // totalValue: 10
}
```

RLockFunc

- RLockFuncf
-

```
RLockFunc(f func(m map[interface{}]interface{}))
```

-

```
func ExampleAnyAnyMap_RLockFunc() {
        var m gmap.Map
        m.Sets(g.MapAnyAny{
                "k1": 1,
                "k2": 2,
                "k3": 3,
                "k4": 4,
        })

        m.RLockFunc(func(m map[interface{}]interface{}) {
                totalValue := 0
                for _, v := range m {
                        totalValue += v.(int)
                }
                fmt.Println("totalValue:", totalValue)
        })

        // Output:
        // totalValue: 10
}
```

Flip

- Flipmapkeyvalue
-

```
Flip()
```

-

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

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

Merge

- MergeAnyAnyMapmapmap
-

```
Merge(other *AnyAnyMap)
```

-

```
func ExampleAnyAnyMap_Merge() {
        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]
}
```

String

- Stringmap
-

```
String() string
```

-

```
func ExampleAnyAnyMap_String() {
        var m gmap.Map
        m.Sets(g.MapAnyAny{
                "k1": "v1",
        })

        fmt.Println(m.String())

        // Output:
        // {"k1":"v1"}
}
```

MarshalJSON

- MarshalJSONjson.Marshal
-

```
MarshalJSON() ([]byte, error)
```

-

```
func ExampleAnyAnyMap_MarshalJSON() {
        var m gmap.Map
        m.Sets(g.MapAnyAny{
                "k1": "v1",
                "k2": "v2",
                "k3": "v3",
                "k4": "v4",
        })

        bytes, err := m.MarshalJSON()
        if err == nil {
                fmt.Println(gconv.String(bytes))
        }

        // Output:
        // {"k1":"v1","k2":"v2","k3":"v3","k4":"v4"}
}
```

UnmarshalJSON

- UnmarshalJSONjson.Unmarshal
-

```
UnmarshalJSON(b []byte) error
```

-

```
func ExampleAnyAnyMap_UnmarshalJSON() {
        var m gmap.Map
        m.Sets(g.MapAnyAny{
                "k1": "v1",
                "k2": "v2",
                "k3": "v3",
                "k4": "v4",
        })

        var n gmap.Map

        err := n.UnmarshalJSON(gconv.Bytes(m.String()))
        if err == nil {
                fmt.Println(n.Map())
        }

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

UnmarshalValue

- UnmarshalValuemap
-

```
UnmarshalValue(value interface{}) (err error)
```

-

```
func ExampleAnyAnyMap_UnmarshalValue() {
        var m gmap.Map
        m.Sets(g.MapAnyAny{
                "k1": "v1",
                "k2": "v2",
                "k3": "v3",
                "k4": "v4",
        })

        var n gmap.Map
        err := n.UnmarshalValue(m.String())
        if err == nil {
                fmt.Println(n.Map())
        }
        // Output:
        // map[k1:v1 k2:v2 k3:v3 k4:v4]
}
```