-

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

New

- New value Var safe Var,false
-

```
func New(value interface{}, safe ...bool) *Var
```

-

```
// New
func ExampleVarNew() {
        v := gvar.New(400)
        g.Dump(v)

        // Output:
        // "400"
}
```

Clone

- Clone Var Var
-

```
func (v *Var) Clone() *Var
```

-

```
// Clone
func ExampleVar_Clone() {
        tmp := "fisrt hello"
        v := gvar.New(tmp)
        g.DumpWithType(v.Clone())
        fmt.Println(v == v.Clone())

        // Output:
        // *gvar.Var(11) "fisrt hello"
        // false
}
```

Set

- Set vvaluev
-

```
func (v *Var) Set(value interface{}) (old interface{})
```

-

Content Menu

```
// Set
func ExampleVar_Set() {
        var v = gvar.New(100.00)
        g.Dump(v.Set(200.00))
        g.Dump(v)

        // Output:
        // 100
        // "200"
}
```

Val

- `Val interface{}`
-

```
func (v *Var) Val() interface{}
```

-

```
// Val
func ExampleVar_Val() {
        var v = gvar.New(100.00)
        g.DumpWithType(v.Val())

        // Output:
        // float64(100)
}
```

Interface

- `Interface Val`
-

```
func (v *Var) Interface() interface{}
```

-

```
// Interface
func ExampleVar_Interface() {
        var v = gvar.New(100.00)
        g.DumpWithType(v.Interface())

        // Output:
        // float64(100)
}
```

Bytes

- `Bytesv`
-

```
func (v *Var) Bytes() []byte
```

-

```
// Bytes
func ExampleVar_Bytes() {
        var v = gvar.New("GoFrame")
        g.DumpWithType(v.Bytes())

        // Output:
        // []byte(7) "GoFrame"
}
```

String

- Stringv
-

```
func (v *Var) String() string
```

-

```
// String
func ExampleVar_String() {
        var v = gvar.New("GoFrame")
        g.DumpWithType(v.String())

        // Output:
        // string(7) "GoFrame"
}
```

Bool

- Boolv
-

```
func (v *Var) Bool() bool
```

-

```
// Bool
func ExampleVar_Bool() {
        var v = gvar.New(true)
        g.DumpWithType(v.Bool())

        // Output:
        // bool(true)
}
```

Int

- Intv
-

```
func (v *Var) Int() int
```

-

```
// Int
func ExampleVar_Int() {
        var v = gvar.New(-1000)
        g.DumpWithType(v.Int())

        // Output:
        // int(-1000)
}
```

Uint

- Uintv
-

```
func (v *Var) Uint() uint
```

-

```
// Uint
func ExampleVar_Uint() {
        var v = gvar.New(1000)
        g.DumpWithType(v.Uint())

        // Output:
        // uint(1000)
}
```

Float32

- Float32v32
-

```
func (v *Var) Float32() float32
```

-

```
// Float32
func ExampleVar_Float32() {
        var price = gvar.New(100.00)
        g.DumpWithType(price.Float32())

        // Output:
        // float32(100)
}
```

Float64

- Float64v64
-

```
func (v *Var) Float64() float64
```

-

```
// Float32
func ExampleVar_Float64() {
        var price = gvar.New(100.00)
        g.DumpWithType(price.Float64())

        // Output:
        // float64(100)
}
```

Time

- Timevtime.Timeformat gtime Y-m-d H:i:s
-

```
func (v *Var) Time(format ...string) time.Time
```

-

```
// Time
func ExampleVar_Time() {
        var v = gvar.New("2021-11-11 00:00:00")
        g.DumpWithType(v.Time())

        // Output:
        // time.Time(29) "2021-11-11 00:00:00 +0800 CST"
}
```

GTime

- GTimev*gtime.Timeformatgtime Y-m-d H:i:s
-

```
func (v *Var) GTime(format ...string) *gtime.Time
```

-

```
// GTime
func ExampleVar_GTime() {
        var v = gvar.New("2021-11-11 00:00:00")
        g.DumpWithType(v.GTime())

        // Output:
        // *gtime.Time(19) "2021-11-11 00:00:00"
}
```

Duration

- Durationvtime.Durationv time.ParseDuration
-

```
func (v *Var) Duration() time.Duration
```

-

```
// Duration
func ExampleVar_Duration() {
        var v = gvar.New("300s")
        g.DumpWithType(v.Duration())

        // Output:
        // time.Duration(4) "5m0s"
}
```

MarshalJSON

- MarshalJSON jsonMarshalJSON
-

```
func (v *Var) MarshalJSON() ([]byte, error)
```

-

```
// MarshalJSON
func ExampleVar_MarshalJSON() {
        testMap := g.Map{
                "code":  "0001",
                "name":  "Golang",
                "count": 10,
        }

        var v = gvar.New(testMap)
        res, err := json.Marshal(&v)
        if err != nil {
                panic(err)
        }
        g.DumpWithType(res)

        // Output:
        // []byte(42) "{"code":"0001","count":10,"name":"Golang"}"
}
```

UnmarshalJSON

- UnmarshalJSON json UnmarshalJSON
-

```
func (v *Var) UnmarshalJSON(b []byte) error
```

-

```
// UnmarshalJSON
func ExampleVar_UnmarshalJSON() {
        tmp := []byte(`{
             "Code":          "0003",
             "Name":          "Golang Book3",
             "Quantity":      3000,
             "Price":         300,
             "OnSale":        true
        }`)
        var v = gvar.New(map[string]interface{}{})
        if err := json.Unmarshal(tmp, &v); err != nil {
                panic(err)
        }

        g.Dump(v)

        // Output:
        // "{\"Code\":\"0003\",\"Name\":\"Golang Book3\",\"OnSale\":
true,\"Price\":300,\"Quantity\":3000}"
}
```

UnmarshalValue

- UnmarshalValue Var
-

```
func (v *Var) UnmarshalValue(value interface{}) error
```

-

```
// UnmarshalValue
func ExampleVar_UnmarshalValue() {
        tmp := g.Map{
                "code":  "00002",
                "name":  "GoFrame",
                "price": 100,
                "sale":  true,
        }

        var v = gvar.New(map[string]interface{}{})
        if err := v.UnmarshalValue(tmp); err != nil {
                panic(err)
        }
        g.Dump(v)

        // Output:
        // "{\"code\":\"00002\",\"name\":\"GoFrame\",\"price\":100,\"
sale\":true}"
}
```

IsNil

- IsNilvnilniltruefalse
-

```
func (v *Var) IsNil() bool
```

-

```
/// IsNil
func ExampleVar_IsNil() {
        g.Dump(gvar.New(0).IsNil())
        g.Dump(gvar.New(0.1).IsNil())
        // true
        g.Dump(gvar.New(nil).IsNil())
        g.Dump(gvar.New("").IsNil())

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

IsEmpty

- IsEmptyvtruefalse
-

```
func (v *Var) IsEmpty() bool
```

-

```
// IsEmpty
func ExampleVar_IsEmpty() {
        g.Dump(gvar.New(0).IsEmpty())
        g.Dump(gvar.New(nil).IsEmpty())
        g.Dump(gvar.New("").IsEmpty())
        g.Dump(gvar.New(g.Map{"k": "v"}).IsEmpty())

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

IsInt

- IsIntvintinttruefalse
-

```
func (v *Var) IsInt() bool
```

-

```
// IsInt
func ExampleVar_IsInt() {
        g.Dump(gvar.New(0).IsInt())
        g.Dump(gvar.New(0.1).IsInt())
        g.Dump(gvar.New(nil).IsInt())
        g.Dump(gvar.New("").IsInt())

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

IsUint

- IsUintvuintuinttruefalse
-

```
func (v *Var) IsUint() bool
```

-

```
// IsUint
func ExampleVar_IsUint() {
        g.Dump(gvar.New(0).IsUint())
        g.Dump(gvar.New(uint8(8)).IsUint())
        g.Dump(gvar.New(nil).IsUint())

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

IsFloat

- IsFloatvfloatfloattruefalse
-

```
func (v *Var) IsFloat() bool
```

-

```
// IsFloat
func ExampleVar_IsFloat() {
        g.Dump(g.NewVar(uint8(8)).IsFloat())
        g.Dump(g.NewVar(float64(8)).IsFloat())
        g.Dump(g.NewVar(0.1).IsFloat())

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

IsSlice

- IsSlicevslicetruefalse
-

```
func (v *Var) IsSlice() bool
```

-

```
// IsSlice
func ExampleVar_IsSlice() {
        g.Dump(g.NewVar(0).IsSlice())
        g.Dump(g.NewVar(g.Slice{0}).IsSlice())

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

IsMap

- IsMapvmaptruefalse
-

```
func (v *Var) IsMap() bool
```

-

```
// IsMap
func ExampleVar_IsMap() {
        g.Dump(g.NewVar(0).IsMap())
        g.Dump(g.NewVar(g.Map{"k": "v"}).IsMap())
        g.Dump(g.NewVar(g.Slice{}).IsMap())

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

IsStruct

- IsStructvstructtruefalse
-

```
func (v *Var) IsStruct() bool
```

-

```
// IsStruct
func ExampleVar_IsStruct() {
        g.Dump(g.NewVar(0).IsStruct())
        g.Dump(g.NewVar(g.Map{"k": "v"}).IsStruct())

        a := struct{}{}
        g.Dump(g.NewVar(a).IsStruct())
        g.Dump(g.NewVar(&a).IsStruct())

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

ListItemValues

- ListItemValues key/list map struct
-

```
func (v *Var) ListItemValues(key interface{}) (values []interface{})
```

-

```
// ListItemValues
func ExampleVar_ListItemValues() {
        var goods1 = g.List{
                g.Map{"id": 1, "price": 100.00},
                g.Map{"id": 2, "price": 0},
                g.Map{"id": 3, "price": nil},
        }
        var v = gvar.New(goods1)
        fmt.Println(v.ListItemValues("id"))
        fmt.Println(v.ListItemValues("price"))

        // Output:
        // [1 2 3]
        // [100 0 <nil>]
}
```

ListItemValuesUnique

- ListItemValuesUnique key struct/map list map struct
-

```
func (v *Var) ListItemValuesUnique(key string) []interface{}
```

-

```
// ListItemValuesUnique
func ExampleVar_ListItemValuesUnique() {
        var (
                goods1 = g.List{
                        g.Map{"id": 1, "price": 100.00},
                        g.Map{"id": 2, "price": 100.00},
                        g.Map{"id": 3, "price": nil},
                }
                v = gvar.New(goods1)
        )

        fmt.Println(v.ListItemValuesUnique("id"))
        fmt.Println(v.ListItemValuesUnique("price"))

        // Output:
        // [1 2 3]
        // [100 <nil>]
}
```

Struct

- Struct v“”pointermapping
-

```
func (v *Var) Struct(pointer interface{}, mapping ...map[string]
string) error
```

-

```
func ExampleVar_Struct() {
        params1 := g.Map{
                "uid":  1,
                "Name": "john",
        }
        v := gvar.New(params1)
        type tartget struct {
                Uid  int
                Name string
        }
        t := new(tartget)
        if err := v.Struct(&t); err != nil {
                panic(err)
        }
        g.Dump(t)

        // Output:
        // {
        //     Uid:  1,
        //     Name: "john",
        // }
}
```

Structs

- Structs vpointermapping
-

```
func (v *Var) Structs(pointer interface{}, mapping ...map[string]
string) error
```

-

```
func ExampleVar_Structs() {
        paramsArray := []g.Map{}
        params1 := g.Map{
                "uid":  1,
                "Name": "golang",
        }
        params2 := g.Map{
                "uid":  2,
                "Name": "java",
        }

        paramsArray = append(paramsArray, params1, params2)
        v := gvar.New(paramsArray)
        type tartget struct {
                Uid  int
                Name string
        }
        var t []tartget
        if err := v.Structs(&t); err != nil {
                panic(err)
        }
        g.DumpWithType(t)

        // Output:
        // []gvar_test.tartget(2) [
        //     gvar_test.tartget(2) {
        //         Uid:  int(1),
        //         Name: string(6) "golang",
        //     },
        //     gvar_test.tartget(2) {
        //         Uid:  int(2),
        //         Name: string(4) "java",
        //     },
        // ]
}
```

Ints

- Ints v []int
-

```
func (v *Var) Ints() []int
```

-

```
// Ints
func ExampleVar_Ints() {
        var (
                arr = []int{1, 2, 3, 4, 5}
                obj = gvar.New(arr)
        )

        fmt.Println(obj.Ints())

        // Output:
        // [1 2 3 4 5]
}
```

Int64s

- Int64s v []64int
-

```
func (v *Var) Int64s() []64int
```

-

```
// Int64s
func ExampleVar_Int64s() {
        var (
                arr = []int64{1, 2, 3, 4, 5}
                obj = gvar.New(arr)
        )

        fmt.Println(obj.Int64s())

        // Output:
        // [1 2 3 4 5]
}
```

Uints

- Uints v []uint
-

```
func (v *Var) Uints() []uint
```

-

```
// Uints
func ExampleVar_Uints() {
        var (
                arr = []uint{1, 2, 3, 4, 5}
                obj = gvar.New(arr)
        )
        fmt.Println(obj.Uints())

        // Output:
        // [1 2 3 4 5]
}
```

Uint64s

- Uint64s v []uint64
-

```
func (v *Var) Uint64s() []uint64
```

-

```
// Uint64s
func ExampleVar_Uint64s() {
        var (
                arr = []uint64{1, 2, 3, 4, 5}
                obj = gvar.New(arr)
        )

        fmt.Println(obj.Uint64s())

        // Output:
        // [1 2 3 4 5]
}
```

Floats

- Floats Float64s
-

```
func (v *Var) Floats() []float64
```

-

```
// Floats
func ExampleVar_Floats() {
        var (
                arr = []float64{1, 2, 3, 4, 5}
                obj = gvar.New(arr)
        )

        fmt.Println(obj.Floats())

        // Output:
        // [1 2 3 4 5]
}
```

Float64s

- Float64s v[]float64
-

```
func (v *Var) Float64s() []float64
```

-

```
// Float64s
func ExampleVar_Float64s() {
        var (
                arr = []float64{1, 2, 3, 4, 5}
                obj = gvar.New(arr)
        )

        fmt.Println(obj.Float64s())

        // Output:
        // [1 2 3 4 5]
}
```

Float32s

- Float32s v[]float32
-

```
func (v *Var) Float32s() []float32
```

-

```
// Float32s
func ExampleVar_Float32s() {
        var (
                arr = []float32{1, 2, 3, 4, 5}
                obj = gvar.New(arr)
        )

        fmt.Println(obj.Float32s())

        // Output:
        // [1 2 3 4 5]
}
```

Strings

- Strings v[]string
-

```
func (v *Var) Strings() []string
```

-

```
// Strings
func ExampleVar_Strings() {
        var (
                arr = []string{"GoFrame", "Golang"}
                obj = gvar.New(arr)
        )
        fmt.Println(obj.Strings())

        // Output:
        // [GoFrame Golang]
}
```

Interfaces

- Interfaces v[]interface{}
-

```
func (v *Var) Interfaces() []interface{}
```

-

```
// Interfaces
func ExampleVar_Interfaces() {
        var (
                arr = []string{"GoFrame", "Golang"}
                obj = gvar.New(arr)
        )

        fmt.Println(obj.Interfaces())

        // Output:
        // [GoFrame Golang]
}
```

Slice

- Slice Interfaces
-

```
func (v *Var) Slice() []interface{}
```

-

```
// Slice
func ExampleVar_Slice() {
        var (
                arr = []string{"GoFrame", "Golang"}
                obj = gvar.New(arr)
        )

        fmt.Println(obj.Slice())

        // Output:
        // [GoFrame Golang]
}
```

Array

- Array Interfaces
-

```
func (v *Var) Array() []interface{}
```

-

```
// Array
func ExampleVar_Array() {
        var (
                arr = []string{"GoFrame", "Golang"}
                obj = gvar.New(arr)
        )
        fmt.Println(obj.Array())

        // Output:
        // [GoFrame Golang]
}
```

Vars

- Vars v[]var
-

```
func (v *Var) Vars() []*Var
```

-

```
// Vars
func ExampleVar_Vars() {
        var (
                arr = []string{"GoFrame", "Golang"}
                obj = gvar.New(arr)
        )

        fmt.Println(obj.Vars())

        // Output:
        // [GoFrame Golang]
}
```

Map

- Map vmap[string]interface{}
-

```
func (v *Var) Map(tags ...string) map[string]interface{}
```

-

```
// Map
func ExampleVar_Map() {
        var (
                m   = g.Map{"id": 1, "price": 100.00}
                v   = gvar.New(m)
                res = v.Map()
        )

        fmt.Println(res["id"], res["price"])

        // Output:
        // 1 100
}
```

MapStrAny

- MapStrAny MapMapStrAny
-

```
func (v *Var) MapStrAny() map[string]interface{}
```

-

```
// MapStrAny
func ExampleVar_MapStrAny() {
        var (
                m1 = g.Map{"id": 1, "price": 100}
                v  = gvar.New(m1)
                v2 = v.MapStrAny()
        )

        fmt.Println(v2["price"], v2["id"])

        // Output:
        // 100 1
}
```

MapStrStr

- MapStrStr vmap[string]string
-

```
func (v *Var) MapStrStr(tags ...string) map[string]string
```

-

```
// MapStrStr
func ExampleVar_MapStrStr() {
        var (
                m1 = g.Map{"id": 1, "price": 100}
                v  = gvar.New(m1)
                v2 = v.MapStrStr()
        )

        fmt.Println(v2["price"] + "$")

        // Output:
        // 100$
}
```

MapStrVar

- `MapStrVar vmap[string]*Var`
-

```
func (v *Var) MapStrVar(tags ...string) map[string]*Var
```

-

```
// MapStrVar
func ExampleVar_MapStrVar() {
        var (
                m1 = g.Map{"id": 1, "price": 100}
                v  = gvar.New(m1)
                v2 = v.MapStrVar()
        )

        fmt.Println(v2["price"].Float64() * 100)

        // Output:
        // 10000
}
```

MapDeep

- `MapDeep vmap[string]interface{}`
-

```
func (v *Var) MapDeep(tags ...string) map[string]interface{}
```

-

```
// MapDeep
func ExampleVar_MapDeep() {
        var (
                m1 = g.Map{"id": 1, "price": 100}
                m2 = g.Map{"product": m1}
                v  = gvar.New(m2)
                v2 = v.MapDeep()
        )

        fmt.Println(v2["product"])

        // Output:
        // map[id:1 price:100]
}
```

MapStrStrDeep

- MapStrStrDeep vmap[string]string
-

```
func (v *Var) MapStrStrDeep(tags ...string) map[string]string
```

-

```
// MapStrStrDeep
func ExampleVar_MapStrStrDeep() {
        var (
                m1 = g.Map{"id": 1, "price": 100}
                m2 = g.Map{"product": m1}
                v  = gvar.New(m2)
                v2 = v.MapStrStrDeep()
        )

        fmt.Println(v2["product"])

        // Output:
        // {"id":1,"price":100}
}
```

MapStrVarDeep

- MapStrVarDeep vmap[string]*Var
-

```
func (v *Var) MapStrVarDeep(tags ...string) map[string]*Var
```

-

```
// MapStrVarDeep
func ExampleVar_MapStrVarDeep() {
        var (
                m1 = g.Map{"id": 1, "price": 100}
                m2 = g.Map{"product": m1}
                v  = gvar.New(m2)
                v2 = v.MapStrVarDeep()
        )

        fmt.Println(v2["product"])

        // Output:
        // {"id":1,"price":100}
}
```

Maps

- Maps vmap[string]interface{}
-

```
func (v *Var) Maps(tags ...string) []map[string]interface{}
```

-

```
// Maps
func ExampleVar_Maps() {
        var m = gvar.New(g.ListIntIntInt{g.MapIntIntInt{0: 100, 1: 200}, g.
MapIntIntInt{0: 300, 1: 400}})
        fmt.Printf("%#v", m.Maps())

        // Output:
        // []map[string]interface {}{map[string]interface {}{"0":
100, "1":200}, map[string]interface {}{"0":300, "1":400}}
}
```

MapsDeep

- MapsDeep v[]map[string]interface{}
-

```
func (v *Var) MapsDeep(tags ...string) []map[string]interface{}
```

-

```
// MapsDeep
func ExampleVar_MapsDeep() {
        var (
                p1 = g.MapStrAny{"product": g.Map{"id": 1, "price":
100}}
                p2 = g.MapStrAny{"product": g.Map{"id": 2, "price":
200}}
                v  = gvar.New(g.ListStrAny{p1, p2})
                v2 = v.MapsDeep()
        )

        fmt.Printf("%#v", v2)

        // Output:
        // []map[string]interface {}{map[string]interface {}
{"product":map[string]interface {}{"id":1, "price":100}}, map[string]
interface {}{"product":map[string]interface {}{"id":2, "price":200}}}
}
```

MapToMap

- MapToMap vpointermapmapping
-

```
func (v *Var) MapToMap(pointer interface{}, mapping ...map[string]
string) (err error)
```

-

```
// MapToMap
func ExampleVar_MapToMap() {
        var (
                m1 = gvar.New(g.MapIntInt{0: 100, 1: 200})
                m2 = g.MapStrStr{}
        )

        err := m1.MapToMap(&m2)
        if err != nil {
                panic(err)
        }

        fmt.Printf("%#v", m2)

        // Output:
        // map[string]string{"0":"100", "1":"200"}
}
```

MapToMaps

- MapToMaps v pointer mapmapping
-

```
func (v *Var) MapToMaps(pointer interface{}, mapping ...map[string]
string) (err error)
```

-

```
// MapToMaps
func ExampleVar_MapToMaps() {
        var (
                p1 = g.MapStrAny{"product": g.Map{"id": 1, "price":
100}}
                p2 = g.MapStrAny{"product": g.Map{"id": 2, "price":
200}}
                v  = gvar.New(g.ListStrAny{p1, p2})
                v2 []g.MapStrStr
        )

        err := v.MapToMaps(&v2)
        if err != nil {
                panic(err)
        }
        fmt.Printf("%#v", v2)

        // Output:
        // []map[string]string{map[string]string{"product":"{\"id\":
1,\"price\":100}"}, map[string]string{"product":"{\"id\":2,\"price\":
200}"}}
}
```

MapToMapsDeep

- MapToMaps vpointer mapmapping
-

```
func (v *Var) MapToMapsDeep(pointer interface{}, mapping ...map
[string]string) (err error)
```

-

```
// MapToMapDeep
func ExampleVar_MapToMapDeep() {
        var (
                p1 = gvar.New(g.MapStrAny{"product": g.Map{"id": 1,
"price": 100}})
                p2 = g.MapStrAny{}
        )

        err := p1.MapToMap(&p2)
        if err != nil {
                panic(err)
        }
        fmt.Printf("%#v", p2)

        // Output:
        // map[string]interface {}{"product":map[string]interface {}
{"id":1, "price":100}}
}
```

Scan

- Scan pointerparamspointerpointer

  *map *[]map *[]*map *struct **struct *[]struct *[]*struct
-

```
func (v *Var) Scan(pointer interface{}, mapping ...map[string]
string) error
```

-

```
// Scan
func ExampleVar_Scan() {
        type Student struct {
                Id     *g.Var
                Name   *g.Var
                Scores *g.Var
        }
        var (
                s Student
                m = g.Map{
                        "Id":     1,
                        "Name":   "john",
                        "Scores": []int{100, 99, 98},
                }
        )
        if err := gconv.Scan(m, &s); err != nil {
                panic(err)
        }

        g.DumpWithType(s)

        // Output:
        // gvar_test.Student(3) {
        //     Id:     *gvar.Var(1) "1",
        //     Name:   *gvar.Var(4) "john",
        //     Scores: *gvar.Var(11) "[100,99,98]",
        // }
}
```