-

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

StrSet

NewStrSet

- NewStrSetsafefalse
-

```
func NewStrSet(safe ...bool) *StrSet
```

-

```
func ExampleNewStrSet() {
        strSet := gset.NewStrSet(true)
        strSet.Add([]string{"str1", "str2", "str3"}...)
        fmt.Println(strSet.Slice())

        // May Output:
        // [str3 str1 str2]
}
```

NewStrSetFrom

- NewStrSetFromsafefalse
-

```
func NewStrSetFrom(items []string, safe ...bool) *StrSet
```

-

```
func ExampleNewStrSetFrom() {
        strSet := gset.NewStrSetFrom([]string{"str1", "str2",
"str3"}, true)
        fmt.Println(strSet.Slice())

        // May Output:
        // [str1 str2 str3]
}
```

Add

- Add
-

```
func (set *StrSet) Add(item ...string)
```

-

```
func ExampleStrSet_Add() {
        strSet := gset.NewStrSetFrom([]string{"str1", "str2",
"str3"}, true)
        strSet.Add("str")
        fmt.Println(strSet.Slice())
        fmt.Println(strSet.AddIfNotExist("str"))

        // Mya Output:
        // [str str1 str2 str3]
        // false
}
```

AddIfNotExist

- Addifnotexistitemitemtruefalse
-

```
func (set *StrSet) AddIfNotExist(item string) bool
```

-

```
func ExampleStrSet_AddIfNotExist() {
        strSet := gset.NewStrSetFrom([]string{"str1", "str2",
"str3"}, true)
        strSet.Add("str")
        fmt.Println(strSet.Slice())
        fmt.Println(strSet.AddIfNotExist("str"))

        // Mya Output:
        // [str str1 str2 str3]
        // false
}
```

AddIfNotExistFunc

- AddIfNotExistFuncitemftrueitemtruefalse
-

```
func (set *StrSet) AddIfNotExistFunc(item string, f func() bool) bool
```

-

```
func ExampleStrSet_AddIfNotExistFunc() {
        strSet := gset.NewStrSetFrom([]string{"str1", "str2",
"str3"}, true)
        strSet.Add("str")
        fmt.Println(strSet.Slice())
        fmt.Println(strSet.AddIfNotExistFunc("str5", func() bool {
                return true
        }))

        // May Output:
        // [str1 str2 str3 str]
        // true
}
```

AddIfNotExistFuncLock

- AddifnotExistFuncLockAddIfNotExistFuncgoroutineAddifnotExistFuncLockgoroutinesafetrueAddIfNotExistFunc
-
```
func (set *StrSet) AddIfNotExistFuncLock(item string, f func() bool)
bool
```
-
```
func ExampleStrSet_AddIfNotExistFuncLock() {
        strSet := gset.NewStrSetFrom([]string{"str1", "str2",
"str3"}, true)
        strSet.Add("str")
        fmt.Println(strSet.Slice())
        fmt.Println(strSet.AddIfNotExistFuncLock("str4", func() bool
{
                return true
        }))

        // May Output:
        // [str1 str2 str3 str]
        // true
}
```

Clear

- Clear
-
```
func (set *StrSet) Clear()
```
-
```
func ExampleStrSet_Clear() {
        strSet := gset.NewStrSetFrom([]string{"str1", "str2",
"str3"}, true)
        fmt.Println(strSet.Size())
        strSet.Clear()
        fmt.Println(strSet.Size())

        // Output:
        // 3
        // 0
}
```

Intersect

- Intrersect setothersnewSetnewSetsetothers
-
```
func (set *StrSet) Intersect(others ...*StrSet) (newSet *StrSet)
```
-

```
func ExampleStrSet_Intersect() {
        s1 := gset.NewStrSet(true)
        s1.Add([]string{"a", "b", "c"}...)
        var s2 gset.StrSet
        s2.Add([]string{"a", "b", "c", "d"}...)
        fmt.Println(s2.Intersect(s1).Slice())

        // May Output:
        // [c a b]
}
```

Diff

- Diff setothersnewSetnewSetsetothersothers
-

```
func (set *StrSet) Diff(others ...*StrSet) (newSet *StrSet)
```

-

```
func ExampleStrSet_Diff() {
        s1 := gset.NewStrSetFrom([]string{"a", "b", "c"}, true)
        s2 := gset.NewStrSetFrom([]string{"a", "b", "c", "d"}, true)
        fmt.Println(s2.Diff(s1).Slice())

        // Output:
        // [d]
}
```

Union

- unionsetothersnewSet
-

```
func (set *StrSet) Union(others ...*StrSet) (newSet *StrSet)
```

-

```
func ExampleStrSet_Union() {
        s1 := gset.NewStrSet(true)
        s1.Add([]string{"a", "b", "c", "d"}...)
        s2 := gset.NewStrSet(true)
        s2.Add([]string{"a", "b", "d"}...)
        fmt.Println(s1.Union(s2).Slice())

        // May Output:
        // [a b c d]
}
```

Complement

- Complement setfullnewSet
-

```
func (set *StrSet) Complement(full *StrSet) (newSet *StrSet)
```

-

```
func ExampleStrSet_Complement() {
        strSet := gset.NewStrSetFrom([]string{"str1", "str2",
"str3", "str4", "str5"}, true)
        s := gset.NewStrSetFrom([]string{"str1", "str2", "str3"},
true)
        fmt.Println(s.Complement(strSet).Slice())

        // May Output:
        // [str4 str5]
}
```

Contains

- `Contains` `item`
-

```
func (set *StrSet) Contains(item string) bool
```

-

```
func ExampleStrSet_Contains() {
        var set gset.StrSet
        set.Add("a")
        fmt.Println(set.Contains("a"))
        fmt.Println(set.Contains("A"))

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

ContainsI

- `ContainsIContains`
-

```
func (set *StrSet) ContainsI(item string) bool
```

-

```
func ExampleStrSet_ContainsI() {
        var set gset.StrSet
        set.Add("a")
        fmt.Println(set.ContainsI("a"))
        fmt.Println(set.ContainsI("A"))

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

Equal

- `Equal`
-

```
func (set *StrSet) Equal(other *StrSet) bool
```

-

```
func ExampleStrSet_Equal() {
        s1 := gset.NewStrSetFrom([]string{"a", "b", "c"}, true)
        s2 := gset.NewStrSetFrom([]string{"a", "b", "c", "d"}, true)
        fmt.Println(s2.Equal(s1))

        s3 := gset.NewStrSetFrom([]string{"a", "b", "c"}, true)
        s4 := gset.NewStrSetFrom([]string{"a", "b", "c"}, true)
        fmt.Println(s3.Equal(s4))

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

IsSubSetOf

- IsSumSetOfsetother
-

```
func (set *StrSet) IsSubsetOf(other *StrSet) bool
```

-

```
func ExampleStrSet_IsSubsetOf() {
        s1 := gset.NewStrSet(true)
        s1.Add([]string{"a", "b", "c", "d"}...)
        var s2 gset.StrSet
        s2.Add([]string{"a", "b", "d"}...)
        fmt.Println(s2.IsSubsetOf(s1))

        // Output:
        // true
}
```

Iterator

- Iterator fsetftrue
-

```
func (set *StrSet) Iterator(f func(v string) bool)
```

-

```
func ExampleStrSet_Iterator() {
        s1 := gset.NewStrSet(true)
        s1.Add([]string{"a", "b", "c", "d"}...)
        s1.Iterator(func(v string) bool {
                fmt.Println("Iterator", v)
                return true
        })

        // May Output:
        // Iterator a
        // Iterator b
        // Iterator c
        // Iterator d
}
```

Join

- Join glue
-
```
func (set *StrSet) Join(glue string) string
```
-
```
func ExampleStrSet_Join() {
        s1 := gset.NewStrSet(true)
        s1.Add([]string{"a", "b", "c", "d"}...)
        fmt.Println(s1.Join(","))

        // May Output:
        // b,c,d,a
}
```

LockFunc

- LockFunc setf
-
```
func (set *StrSet) LockFunc(f func(m map[string]struct{}))
```
-
```
func ExampleStrSet_LockFunc() {
        s1 := gset.NewStrSet(true)
        s1.Add([]string{"1", "2"}...)
        s1.LockFunc(func(m map[string]struct{}) {
                m["3"] = struct{}{}
        })
        fmt.Println(s1.Slice())

        // May Output
        // [2 3 1]

}
```

RLockFunc

- RLockFunc setf

-

```
func (set *StrSet) RLockFunc(f func(m map[string]struct{}))
```

-

```
func ExampleStrSet_RLockFunc() {
        s1 := gset.NewStrSet(true)
        s1.Add([]string{"a", "b", "c", "d"}...)
        s1.RLockFunc(func(m map[string]struct{}) {
                fmt.Println(m)
        })

        // Output:
        // map[a:{} b:{} c:{} d:{}]
}
```

Merge

- Mergeothersset
-

```
func (set *StrSet) Merge(others ...*StrSet) *StrSet
```

-

```
func ExampleStrSet_Merge() {
        s1 := gset.NewStrSet(true)
        s1.Add([]string{"a", "b", "c", "d"}...)

        s2 := gset.NewStrSet(true)
        fmt.Println(s1.Merge(s2).Slice())

        // May Output:
        // [d a b c]
}
```

Pop

- Pop
-

```
func (set *StrSet) Pop() string
```

-

```
func ExampleStrSet_Pop() {
        s1 := gset.NewStrSet(true)
        s1.Add([]string{"a", "b", "c", "d"}...)

        fmt.Println(s1.Pop())

        // May Output:
        // a
}
```

Pops

- Pops `sizesize == -1`
-

```
func (set *StrSet) Pops(size int) []string
```

-

```
func ExampleStrSet_Pops() {
        s1 := gset.NewStrSet(true)
        s1.Add([]string{"a", "b", "c", "d"}...)
        for _, v := range s1.Pops(2) {
                fmt.Println(v)
        }

        // May Output:
        // a
        // b
}
```

Remove

- Removeitem
-

```
func (set *StrSet) Remove(item string)
```

-

```
func ExampleStrSet_Remove() {
        s1 := gset.NewStrSet(true)
        s1.Add([]string{"a", "b", "c", "d"}...)
        s1.Remove("a")
        fmt.Println(s1.Slice())

        // May Output:
        // [b c d]
}
```

Size

- Size
-

```
func (set *StrSet) Size() int
```

-

```
func ExampleStrSet_Size() {
        s1 := gset.NewStrSet(true)
        s1.Add([]string{"a", "b", "c", "d"}...)
        fmt.Println(s1.Size())

        // Output:
        // 4
}
```

Silce

- Sliceslice
-

```
func (set *StrSet) Slice() []string
```

-

```
func ExampleStrSet_Slice() {
        s1 := gset.NewStrSet(true)
        s1.Add([]string{"a", "b", "c", "d"}...)
        fmt.Println(s1.Slice())

        // May Output:
        // [a,b,c,d]
}
```

String

- String
-

```
func (set *StrSet) String() string
```

-

```
func ExampleStrSet_String() {
        s1 := gset.NewStrSet(true)
        s1.Add([]string{"a", "b", "c", "d"}...)
        fmt.Println(s1.String())

        // May Output:
        // "a","b","c","d"
}
```

Sum

- Sum
-

```
func (set *StrSet) Sum() (sum int)
```

-

```
func ExampleStrSet_Sum() {
        s1 := gset.NewStrSet(true)
        s1.Add([]string{"1", "2", "3", "4"}...)
        fmt.Println(s1.Sum())

        // Output:
        // 10
}
```

Walk

- Walkff
-

```
func (set *StrSet) Walk(f func(item string) string) *StrSet
```

-

```
func ExampleStrSet_Walk() {
        var (
                set    gset.StrSet
                names  = g.SliceStr{"user", "user_detail"}
                prefix = "gf_"
        )
        set.Add(names...)
        // Add prefix for given table names.
        set.Walk(func(item string) string {
                return prefix + item
        })
        fmt.Println(set.Slice())

        // May Output:
        // [gf_user gf_user_detail]
}
```

MarshalJSON

- MarshalJSONjson.MarshalMarshalJSON
-

```
func (set *StrSet) MarshalJSON() ([]byte, error)
```

-

```
func ExampleStrSet_MarshalJSON() {
        type Student struct {
                Id     int
                Name   string
                Scores *gset.StrSet
        }
        s := Student{
                Id:     1,
                Name:   "john",
                Scores: gset.NewStrSetFrom([]string{"100", "99",
"98"}, true),
        }
        b, _ := json.Marshal(s)
        fmt.Println(string(b))

        // May Output:
        // {"Id":1,"Name":"john","Scores":["100","99","98"]}
}
```

UnmarshalJSON

- UnmarshalJSONjson.UnmarshalUnmarshalJSON
-

```
func (set *StrSet) UnmarshalJSON(b []byte) error
```

-

```
func ExampleStrSet_UnmarshalJSON() {
        b := []byte(`{"Id":1,"Name":"john","Scores":["100","99","
98"]}`)
        type Student struct {
                Id     int
                Name   string
                Scores *gset.StrSet
        }
        s := Student{}
        json.Unmarshal(b, &s)
        fmt.Println(s)

        // May Output:
        // {1 john "99","98","100"}
}
```

UnmarshalValue

- UnfarshalValuegoframeinterface{}interface{}
-

```
func (set *StrSet) UnmarshalValue(value interface{}) (err error)
```

-

```
func ExampleStrSet_UnmarshalValue() {
        b := []byte(`{"Id":1,"Name":"john","Scores":["100","99","
98"]}`)
        type Student struct {
                Id     int
                Name   string
                Scores *gset.StrSet
        }
        s := Student{}
        json.Unmarshal(b, &s)
        fmt.Println(s)

        // May Output:
        // {1 john "99","98","100"}
}
```