-

```go
package main

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


func main () {
    // int
    a := garray.NewIntArray(true)

    //
    for i := 0; i < 10; i++ {
        a.Append(i)
    }

    //
    fmt.Println(a.Len())

    //
    fmt.Println(a.Slice())

    //
    fmt.Println(a.Get(6))

    //
    a.InsertAfter(9, 11)
    //
    a.InsertBefore(10, 10)
    fmt.Println(a.Slice())

    //
    a.Set(0, 100)
    fmt.Println(a.Slice())

    //
    fmt.Println(a.Search(5))

    //
    a.Remove(0)
    fmt.Println(a.Slice())

    //
    a.LockFunc(func(array []int) {
        // 100
        array[len(array) - 1] = 100
    })

    //
    a.RLockFunc(func(array []int) {
        fmt.Println(array[len(array) - 1])
    })

    //
    fmt.Println(a.Slice())
    a.Clear()
    fmt.Println(a.Slice())
}
```

Content Menu

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

```go
package main

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


func main () {
    // (SortedIntArray)
    a := garray.NewSortedArray(func(v1, v2 interface{}) int {
        if v1.(int) < v2.(int) {
            return 1
        }
        if v1.(int) > v2.(int) {
            return -1
        }
        return 0
    })

    //
    a.Add(2)
    a.Add(3)
    a.Add(1)
    fmt.Println(a.Slice())

    //
    a.Add(3)
    fmt.Println(a.Slice())

    //
    // 0: ; <0:; >0:
    fmt.Println(a.Search(1))

    //
    a.SetUnique(true)
    fmt.Println(a.Slice())
    a.Add(1)
    fmt.Println(a.Slice())
}
```

```
[3 2 1]
[3 3 2 1]
3 0
[3 2 1]
[3 2 1]
```

Iterate*

```
package main

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

func main() {
        array := garray.NewStrArrayFrom(g.SliceStr{"a", "b", "c"})
        // Iterator is alias of IteratorAsc, which iterates the array
readonly in ascending order
        //  with given callback function <f>.
        // If <f> returns true, then it continues iterating; or false to
stop.
        array.Iterator(func(k int, v string) bool {
                fmt.Println(k, v)
                return true
        })
        // IteratorDesc iterates the array readonly in descending order
with given callback function <f>.
        // If <f> returns true, then it continues iterating; or false to
stop.
        array.IteratorDesc(func(k int, v string) bool {
                fmt.Println(k, v)
                return true
        })

        // Output:
        // 0 a
        // 1 b
        // 2 c
        // 2 c
        // 1 b
        // 0 a
}
```

Pop*

```
package main

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

func main() {
        array := garray.NewFrom([]interface{}{1, 2, 3, 4, 5, 6, 7, 8, 9})

        // Any Pop* functions pick, delete and return the item from array.

        fmt.Println(array.PopLeft())
        fmt.Println(array.PopLefts(2))
        fmt.Println(array.PopRight())
        fmt.Println(array.PopRights(2))

        // Output:
        // 1 true
        // [2 3]
        // 9 true
        // [7 8]
}
```

Rand/PopRand/

```
package main

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

func main() {
        array := garray.NewFrom(g.Slice{1, 2, 3, 4, 5, 6, 7, 8, 9})

        // Randomly retrieve and return 2 items from the array.
        // It does not delete the items from array.
        fmt.Println(array.Rands(2))

        // Randomly pick and return one item from the array.
        // It deletes the picked up item from array.
        fmt.Println(array.PopRand())
}
```

Contains/ContainsI

```
package main

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

func main() {
        var array garray.StrArray
        array.Append("a")
        fmt.Println(array.Contains("a"))
        fmt.Println(array.Contains("A"))
        fmt.Println(array.ContainsI("A"))

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

FilterEmpty/FilterNil

```
package main

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

func main() {
	array1 := garray.NewFrom(g.Slice{0, 1, 2, nil, "", g.Slice{},
"john"})
	array2 := garray.NewFrom(g.Slice{0, 1, 2, nil, "", g.Slice{},
"john"})
	fmt.Printf("%#v\n", array1.FilterNil().Slice())
	fmt.Printf("%#v\n", array2.FilterEmpty().Slice())

	// Output:
	// []interface {}{0, 1, 2, "", []interface {}{}, "john"}
	// []interface {}{1, 2, "john"}
}
```

Reverse

```
package main

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

func main() {
	array := garray.NewFrom(g.Slice{1, 2, 3, 4, 5, 6, 7, 8, 9})

	// Reverse makes array with elements in reverse order.
	fmt.Println(array.Reverse().Slice())

	// Output:
	// [9 8 7 6 5 4 3 2 1]
}
```

Shuffle

```
package main

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

func main() {
	array := garray.NewFrom(g.Slice{1, 2, 3, 4, 5, 6, 7, 8, 9})

	// Shuffle randomly shuffles the array.
	fmt.Println(array.Shuffle().Slice())
}
```

Walk

```go
package main

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

func main() {
	var array garray.StrArray
	tables := g.SliceStr{"user", "user_detail"}
	prefix := "gf_"
	array.Append(tables...)
	// Add prefix for given table names.
	array.Walk(func(value string) string {
		return prefix + value
	})
	fmt.Println(array.Slice())

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

Join

```go
package main

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

func main() {
	array := garray.NewFrom(g.Slice{"a", "b", "c", "d"})
	fmt.Println(array.Join(","))

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

Chunk

```go
package main

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

func main() {
	array := garray.NewFrom(g.Slice{1, 2, 3, 4, 5, 6, 7, 8, 9})

	// Chunk splits an array into multiple arrays,
	// the size of each array is determined by <size>.
	// The last chunk may contain less than size elements.
	fmt.Println(array.Chunk(2))

	// Output:
	// [[1 2] [3 4] [5 6] [7 8] [9]]
}
```

Merge

```
package main

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

func main() {
        array1 := garray.NewFrom(g.Slice{1, 2})
        array2 := garray.NewFrom(g.Slice{3, 4})
        slice1 := g.Slice{5, 6}
        slice2 := []int{7, 8}
        slice3 := []string{"9", "0"}
        fmt.Println(array1.Slice())
        array1.Merge(array1)
        array1.Merge(array2)
        array1.Merge(slice1)
        array1.Merge(slice2)
        array1.Merge(slice3)
        fmt.Println(array1.Slice())

        // Output:
        // [1 2]
        // [1 2 1 2 3 4 5 6 7 8 9 0]
}
```

JSON/

garrayjson/

1. Marshal

```
package main

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

func main() {
    type Student struct {
        Id     int
        Name   string
        Scores *garray.IntArray
    }
    s := Student{
        Id:     1,
        Name:   "john",
        Scores: garray.NewIntArrayFrom([]int{100, 99, 98}),
    }
    b, _ := json.Marshal(s)
    fmt.Println(string(b))
}
```

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

2. Unmarshal

```go
package main


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


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

```
{1 john [100,99,98]}
```