-

CPUns


```
package main

import (
        "fmt"
        "github.com/gogf/gf/v2/os/gcache"
        "github.com/gogf/gf/v2/os/gctx"
)

func main() {
        //
        // gcache
        var (
                ctx   = gctx.New()
                cache = gcache.New()
        )

        //
        err := cache.Set(ctx, "k1", "v1", 0)
        if err != nil {
                panic(err)
        }

        //
        value, err := cache.Get(ctx, "k1")
        if err != nil {
                panic(err)
        }
        fmt.Println(value)

        //
        size, err := cache.Size(ctx)
        if err != nil {
                panic(err)
        }
        fmt.Println(size)

        //
        b, err := cache.Contains(ctx, "k1")
        if err != nil {
                panic(err)
        }
        fmt.Println(b)

        //
        removedValue, err := cache.Remove(ctx, "k1")
        if err != nil {
                panic(err)
        }
        fmt.Println(removedValue)

        // GC
        if err = cache.Close(ctx); err != nil {
                panic(err)
        }
}
```

```
v1
1
true
v1
```

```
package main

import (
        "fmt"
        "github.com/gogf/gf/v2/os/gcache"
        "github.com/gogf/gf/v2/os/gctx"
        "time"
)

func main() {
        var (
                ctx = gctx.New()
        )
        // 1000
        _, err := gcache.SetIfNotExist(ctx, "k1", "v1", time.Second)
        if err != nil {
                panic(err)
        }

        //
        keys, err := gcache.Keys(ctx)
        if err != nil {
                panic(err)
        }
        fmt.Println(keys)

        //
        values, err := gcache.Values(ctx)
        if err != nil {
                panic(err)
        }
        fmt.Println(values)

        //
        value, err := gcache.GetOrSet(ctx, "k2", "v2", 0)
        if err != nil {
                panic(err)
        }
        fmt.Println(value)

        //
        data1, err := gcache.Data(ctx)
        if err != nil {
                panic(err)
        }
        fmt.Println(data1)

        // 1k1:v1
        time.Sleep(time.Second)

        // k1:v1k2:v2
        data2, err := gcache.Data(ctx)
        if err != nil {
                panic(err)
        }
        fmt.Println(data2)
}
```

```
[k1]
[v1]
v2
map[k1:v1 k2:v2]
map[k2:v2]
```

GetOrSetFunc*

GetOrSetFuncf func(context.Context) (interface{}, error)f

GetOrSetFuncffGetOrSetFuncff(f)GetOrSetFuncLockfff

```
package main

import (
        "fmt"
        "github.com/gogf/gf/v2/os/gcache"
        "github.com/gogf/gf/v2/os/gctx"
        "time"
)

func main() {
        var (
                ch    = make(chan struct{}, 0)
                ctx   = gctx.New()
                key   = `key`
                value = `value`
        )
        for i := 0; i < 10; i++ {
                go func(index int) {
                        <-ch
                        _, err := gcache.GetOrSetFuncLock(ctx, key, func
(ctx context.Context) (interface{}, error) {
                                fmt.Println(index, "entered")
                                return value, nil
                        }, 0)
                        if err != nil {
                                panic(err)
                        }
                }(i)
        }
        close(ch)
        time.Sleep(time.Second)
}
```

```
9 entered
```

goroutineGetOrSetFuncLockgoroutinegoroutine

LRU

```
package main

import (
        "fmt"
        "github.com/gogf/gf/v2/os/gcache"
        "github.com/gogf/gf/v2/os/gctx"
        "time"
)

func main() {

        var (
                ctx   = gctx.New()
                cache = gcache.New(2) // LRU
        )

        // 10
        for i := 0; i < 10; i++ {
                if err := cache.Set(ctx, i, i, 0); err != nil {
                        panic(err)
                }
        }
        size, err := cache.Size(ctx)
        if err != nil {
                panic(err)
        }
        fmt.Println(size)

        keys, err := cache.Keys(ctx)
        if err != nil {
                panic(err)
        }
        fmt.Println(keys)

        // 1
        value, err := cache.Get(ctx, 1)
        if err != nil {
                panic(err)
        }
        fmt.Println(value)

        // (1)
        //
        time.Sleep(3 * time.Second)
        size, err = cache.Size(ctx)
        if err != nil {
                panic(err)
        }
        fmt.Println(size)

        keys, err = cache.Keys(ctx)
        if err != nil {
                panic(err)
        }
        fmt.Println(keys)
}
```

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

```
CPU: Intel(R) Core(TM) i5-4460  CPU @ 3.20GHz
MEM: 8GB
SYS: Ubuntu 16.04 amd64
```

```
john@john-B85M:~/Workspace/Go/GOPATH/src/github.com/gogf/gf/v2/os/gcache$
go test *.go -bench=".*" -benchmem
goos: linux
goarch: amd64
Benchmark_CacheSet-4                       2000000        897 ns/op
249 B/op        4 allocs/op
Benchmark_CacheGet-4                       5000000        202 ns/op
49 B/op        1 allocs/op
Benchmark_CacheRemove-4                   50000000       35.7 ns/op
0 B/op        0 allocs/op
Benchmark_CacheLruSet-4                    2000000        880 ns/op
399 B/op        4 allocs/op
Benchmark_CacheLruGet-4                    3000000        212 ns/op
33 B/op        1 allocs/op
Benchmark_CacheLruRemove-4                50000000       35.9 ns/op
0 B/op        0 allocs/op
Benchmark_InterfaceMapWithLockSet-4        3000000        477 ns/op
73 B/op        2 allocs/op
Benchmark_InterfaceMapWithLockGet-4       10000000        149 ns/op
0 B/op        0 allocs/op
Benchmark_InterfaceMapWithLockRemove-4    50000000       39.8 ns/op
0 B/op        0 allocs/op
Benchmark_IntMapWithLockWithLockSet-4      5000000        304 ns/op
53 B/op        0 allocs/op
Benchmark_IntMapWithLockGet-4             20000000        164 ns/op
0 B/op        0 allocs/op
Benchmark_IntMapWithLockRemove-4          50000000       33.1 ns/op
0 B/op        0 allocs/op
PASS
ok   command-line-arguments 47.503s
```