-

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

NewBTree

- NewBTreem()BTreesafetreefalse
- m3panic
-

```
NewBTree(m int, comparator func(v1, v2 interface{}) int, safe ...
bool) *BTree
```

-

```
func ExampleNewBTree() {
        bTree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                bTree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }
        fmt.Println(bTree.Map())

        // Output:
        // map[key0:val0 key1:val1 key2:val2 key3:val3 key4:val4
key5:val5]
}
```

NewBTreeFrom

- NewBTreeFromm()map[interface{}]interface{}dataBTreesafetreefalse
- m3panic
-

```
NewBTreeFrom(m int, comparator func(v1, v2 interface{}) int, data map
[interface{}]interface{}, safe ...bool) *BTree
```

-

```
func ExampleNewBTreeFrom() {
        bTree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                bTree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        otherBTree := gtree.NewBTreeFrom(3, gutil.ComparatorString,
bTree.Map())
        fmt.Println(otherBTree.Map())

        // Output:
        // map[key0:val0 key1:val1 key2:val2 key3:val3 key4:val4
key5:val5]
}
```

Clone

- clonetreeBTree

-

```
Clone() *BTree
```

-

```
func ExampleBTree_Clone() {
        b := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                b.Set("key"+gconv.String(i), "val"+gconv.String(i))
        }

        tree := b.Clone()

        fmt.Println(tree.Map())
        fmt.Println(tree.Size())

        // Output:
        // map[key0:val0 key1:val1 key2:val2 key3:val3 key4:val4
key5:val5]
        // 6
}
```

Set

- Settreekey/value
-

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

-

```
func ExampleBTree_Set() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.Map())
        fmt.Println(tree.Size())

        // Output:
        // map[key0:val0 key1:val1 key2:val2 key3:val3 key4:val4
key5:val5]
        // 6
}
```

Sets

- Setstreekey/value
-

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

-

```
func ExampleBTree_Sets() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)

        tree.Sets(map[interface{}]interface{}{
                "key1": "val1",
                "key2": "val2",
        })

        fmt.Println(tree.Map())
        fmt.Println(tree.Size())

        // Output:
        // map[key1:val1 key2:val2]
        // 2
}
```

Get

- GetkeyvaluekeyNil
-
```
Get(key interface{}) (value interface{})
```
-
```
func ExampleBTree_Get() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.Get("key1"))
        fmt.Println(tree.Get("key10"))

        // Output:
        // val1
        // <nil>
}
```

GetOrSet

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

```
func ExampleBTree_GetOrSet() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.GetOrSet("key1", "newVal1"))
        fmt.Println(tree.GetOrSet("key6", "val6"))

        // Output:
        // val1
        // val6
}
```

GetOrSetFunc

- GetOrSetFunckeyvaluekeykeyfunc f
-

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

-

```
func ExampleBTree_GetOrSetFunc() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.GetOrSetFunc("key1", func() interface{} {
                return "newVal1"
        }))
        fmt.Println(tree.GetOrSetFunc("key6", func() interface{} {
                return "val6"
        }))

        // Output:
        // val1
        // val6
}
```

GetOrSetFuncLock

- GetOrSetFunckeyvaluekeykeyfunc f
- GetOrSetFuncLockGetOrSetFuncf
-

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

-

```
func ExampleBTree_GetOrSetFuncLock() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.GetOrSetFuncLock("key1", func() interface{}
{
                return "newVal1"
        }))
        fmt.Println(tree.GetOrSetFuncLock("key6", func() interface{}
{
                return "val6"
        }))

        // Output:
        // val1
        // val6
}
```

GetVar

- GetVarkey*gvar.Var
- gvar.Var
-

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

-

```
func ExampleBTree_GetVar() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.GetVar("key1").String())

        // Output:
        // val1
}
```

GetVarOrSet

- GetVarOrSetGetOrSet*gvar.Var
- gvar.Var
-

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

-

```
func ExampleBTree_GetVarOrSet() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.GetVarOrSet("key1", "newVal1"))
        fmt.Println(tree.GetVarOrSet("key6", "val6"))

        // Output:
        // val1
        // val6
}
```

GetVarOrSetFunc

- GetVarOrSetFuncGetOrSetFunc*gvar.Var
- gvar.Var
-

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

-

```
func ExampleBTree_GetVarOrSetFunc() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.GetVarOrSetFunc("key1", func() interface{} {
                return "newVal1"
        }))
        fmt.Println(tree.GetVarOrSetFunc("key6", func() interface{} {
                return "val6"
        }))

        // Output:
        // val1
        // val6
}
```

GetVarOrSetFuncLock

- GetVarOrSetFuncLockGetOrSetFuncLock*gvar.Var
- gvar.Var
-

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

-

```
func ExampleBTree_GetVarOrSetFuncLock() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.GetVarOrSetFuncLock("key1", func() interface
{} {
                return "newVal1"
        }))
        fmt.Println(tree.GetVarOrSetFuncLock("key6", func() interface
{} {
                return "val6"
        }))

        // Output:
        // val1
        // val6
}
```

SetIfNotExist

- keySetIfNotExistmapkey/valuetruekeyfalsevalue
-

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

-

```
func ExampleBTree_SetIfNotExist() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.SetIfNotExist("key1", "newVal1"))
        fmt.Println(tree.SetIfNotExist("key6", "val6"))

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

SetIfNotExistFunc

- keySetIfNotExistFuncftruekeyfalsevalue
-

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

-

```
func ExampleBTree_SetIfNotExistFunc() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.SetIfNotExistFunc("key1", func() interface
{} {
                return "newVal1"
        }))
        fmt.Println(tree.SetIfNotExistFunc("key6", func() interface
{} {
                return "val6"
        }))

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

SetIfNotExistFuncLock

- keySetIfNotExistFuncfunc ctruekeyfalsevalue
- SetIfNotExistFuncLockSetIfNotExistFuncmutex.Lockf
-

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

-

```
func ExampleBTree_SetIfNotExistFuncLock() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.SetIfNotExistFuncLock("key1", func()
interface{} {
                return "newVal1"
        }))
        fmt.Println(tree.SetIfNotExistFuncLock("key6", func()
interface{} {
                return "val6"
        }))

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

Contains

- Containskeytreekeytruefalse
-

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

-

```
func ExampleBTree_Contains() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.Contains("key1"))
        fmt.Println(tree.Contains("key6"))

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

Remove

- keytreevaluevalue
-

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

-

```
func ExampleBTree_Remove() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.Remove("key1"))
        fmt.Println(tree.Remove("key6"))
        fmt.Println(tree.Map())

        // Output:
        // val1
        // <nil>
        // map[key0:val0 key2:val2 key3:val3 key4:val4 key5:val5]
}
```

Removes

- Removeskeytreevalue
-

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

-

```
func ExampleBTree_Removes() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        removeKeys := make([]interface{}, 2)
        removeKeys = append(removeKeys, "key1")
        removeKeys = append(removeKeys, "key6")

        tree.Removes(removeKeys)

        fmt.Println(tree.Map())

        // Output:
        // map[key0:val0 key2:val2 key3:val3 key4:val4 key5:val5]
}
```

IsEmpty

- IsEmptytreetreetruefalse
-

```
IsEmpty() bool
```

-

```
func ExampleBTree_IsEmpty() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)

        fmt.Println(tree.IsEmpty())

        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.IsEmpty())

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

Size

- Sizetree
-

```
Size() int
```

-

```
func ExampleBTree_Size() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)

        fmt.Println(tree.Size())

        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.Size())

        // Output:
        // 0
        // 6
}
```

Keys

- Keyskey
-

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

-

```
func ExampleBTree_Keys() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 6; i > 0; i-- {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.Keys())

        // Output:
        // [key1 key2 key3 key4 key5 key6]
}
```

Values

- Valueskeyvalue
-

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

-

```
func ExampleBTree_Values() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 6; i > 0; i-- {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.Values())

        // Output:
        // [val1 val2 val3 val4 val5 val6]
}
```

Map

- Mapmapkey/value
-

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

-

```
func ExampleBTree_Map() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.Map())

        // Output:
        // map[key0:val0 key1:val1 key2:val2 key3:val3 key4:val4
key5:val5]
}
```

MapStrAny

- MapStrAnymap[string]interface{}key/value
-

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

-

```
func ExampleBTree_MapStrAny() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set(1000+i, "val"+gconv.String(i))
        }

        fmt.Println(tree.MapStrAny())

        // Output:
        // map[1000:val0 1001:val1 1002:val2 1003:val3 1004:val4
1005:val5]
}
```

Clear

- Cleartree
-

```
Clear()
```

-

```
func ExampleBTree_Clear() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set(1000+i, "val"+gconv.String(i))
        }
        fmt.Println(tree.Size())

        tree.Clear()
        fmt.Println(tree.Size())

        // Output:
        // 6
        // 0
}
```

Replace

- Replacemap[interface{}]interface{}datatreekey/value
-

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

-

```
func ExampleBTree_Replace() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.Map())

        data := map[interface{}]interface{}{
                "newKey0": "newVal0",
                "newKey1": "newVal1",
                "newKey2": "newVal2",
        }

        tree.Replace(data)

        fmt.Println(tree.Map())

        // Output:
        // map[key0:val0 key1:val1 key2:val2 key3:val3 key4:val4
key5:val5]
        // map[newKey0:newVal0 newKey1:newVal1 newKey2:newVal2]
}
```

Height

- Heighttree
-

```
Height() int
```

-

```
func ExampleBTree_Height() {
        tree := gtree.NewBTree(3, gutil.ComparatorInt)
        for i := 0; i < 100; i++ {
                tree.Set(i, i)
        }
        fmt.Println(tree.Height())

        // Output:
        // 6
}
```

Left

- `Left *BTreeEntrynodetree nil`
-

```
Left() *BTreeEntry
```

-

```
func ExampleBTree_Left() {
        tree := gtree.NewBTree(3, gutil.ComparatorInt)
        for i := 1; i < 100; i++ {
                tree.Set(i, i)
        }
        fmt.Println(tree.Left().Key, tree.Left().Value)

        emptyTree := gtree.NewBTree(3, gutil.ComparatorInt)
        fmt.Println(emptyTree.Left())

        // Output:
        // 1 1
        // <nil>
}
```

Right

- `Left *BTreeEntrynodetree nil`
-

```
Right() *BTreeEntry
```

-

```
func ExampleBTree_Right() {
        tree := gtree.NewBTree(3, gutil.ComparatorInt)
        for i := 1; i < 100; i++ {
                tree.Set(i, i)
        }
        fmt.Println(tree.Right().Key, tree.Right().Value)

        emptyTree := gtree.NewBTree(3, gutil.ComparatorInt)
        fmt.Println(emptyTree.Left())

        // Output:
        // 99 99
        // <nil>
}
```

String

- Stringtreenode
-

```
String() string
```

-

```
func ExampleBTree_String() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.String())

        // Output:
        // key0
        // key1
        //     key2
        // key3
        //     key4
        //     key5
}
```

Search

- Searchkeytreekeyfoundtruefalse
-

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

-

```
func ExampleBTree_Search() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        fmt.Println(tree.Search("key0"))
        fmt.Println(tree.Search("key6"))

        // Output:
        // val0 true
        // <nil> false
}
```

Print

- Printtree
-

```
Print()
```

-

```
func ExampleBTree_Print() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        tree.Print()

        // Output:
        // key0
        // key1
        //     key2
        // key3
        //     key4
        //     key5
}
```

Iterator

- Iterator IteratorAsc
-

```
Iterator(f func(key, value interface{}) bool)
```

-

```
func ExampleBTree_Iterator() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 10; i++ {
                tree.Set(i, 10-i)
        }

        var totalKey, totalValue int
        tree.Iterator(func(key, value interface{}) bool {
                totalKey += key.(int)
                totalValue += value.(int)

                return totalValue < 20
        })

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

        // Output:
        // totalKey: 3
        // totalValue: 27
}
```

IteratorFrom

- IteratorFrom IteratorAscFrom
-

```
IteratorFrom(key interface{}, match bool, f func(key, value interface
{}) bool)
```

-

```
func ExampleBTree_IteratorFrom() {
        m := make(map[interface{}]interface{})
        for i := 1; i <= 5; i++ {
                m[i] = i * 10
        }
        tree := gtree.NewBTreeFrom(3, gutil.ComparatorInt, m)

        tree.IteratorFrom(1, true, func(key, value interface{}) bool
{
                fmt.Println("key:", key, ", value:", value)
                return true
        })

        // Output:
        // key: 1 , value: 10
        // key: 2 , value: 20
        // key: 3 , value: 30
        // key: 4 , value: 40
        // key: 5 , value: 50
}
```

IteratorAsc

- IteratorAscftreeftruefalse
-

```
IteratorAsc(f func(key, value interface{}) bool)
```

-

```
func ExampleBTree_IteratorAsc() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 10; i++ {
                tree.Set(i, 10-i)
        }

        tree.IteratorAsc(func(key, value interface{}) bool {
                fmt.Println("key:", key, ", value:", value)
                return true
        })

        // Output:
        // key: 0 , value: 10
        // key: 1 , value: 9
        // key: 2 , value: 8
        // key: 3 , value: 7
        // key: 4 , value: 6
        // key: 5 , value: 5
        // key: 6 , value: 4
        // key: 7 , value: 3
        // key: 8 , value: 2
        // key: 9 , value: 1
}
```

IteratorAscFrom

- IteratorAscFromftree key keymatchtruekeyftruefalse
-

```
IteratorAscFrom(key interface{}, match bool, f func(key, value
interface{}) bool)
```

-

```
func ExampleBTree_IteratorAscFrom_Normal() {
        m := make(map[interface{}]interface{})
        for i := 1; i <= 5; i++ {
                m[i] = i * 10
        }
        tree := gtree.NewBTreeFrom(3, gutil.ComparatorInt, m)

        tree.IteratorAscFrom(1, true, func(key, value interface{})
bool {
                fmt.Println("key:", key, ", value:", value)
                return true
        })

        // Output:
        // key: 1 , value: 10
        // key: 2 , value: 20
        // key: 3 , value: 30
        // key: 4 , value: 40
        // key: 5 , value: 50
}
```

```
func ExampleBTree_IteratorAscFrom_NoExistKey() {
        m := make(map[interface{}]interface{})
        for i := 1; i <= 5; i++ {
                m[i] = i * 10
        }
        tree := gtree.NewBTreeFrom(3, gutil.ComparatorInt, m)

        tree.IteratorAscFrom(0, true, func(key, value interface{})
bool {
                fmt.Println("key:", key, ", value:", value)
                return true
        })

        // Output:
}
```

```
func ExampleBTree_IteratorAscFrom_NoExistKeyAndMatchFalse() {
        m := make(map[interface{}]interface{})
        for i := 1; i <= 5; i++ {
                m[i] = i * 10
        }
        tree := gtree.NewBTreeFrom(3, gutil.ComparatorInt, m)

        tree.IteratorAscFrom(0, false, func(key, value interface{})
bool {
                fmt.Println("key:", key, ", value:", value)
                return true
        })

        // Output:
        // key: 1 , value: 10
        // key: 2 , value: 20
        // key: 3 , value: 30
        // key: 4 , value: 40
        // key: 5 , value: 50
}
```

IteratorDesc

- IteratorDescftreeftruefalse
-

```
IteratorDesc(f func(key, value interface{}) bool)
```

-

```
func ExampleBTree_IteratorDesc() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 10; i++ {
                tree.Set(i, 10-i)
        }

        tree.IteratorDesc(func(key, value interface{}) bool {
                fmt.Println("key:", key, ", value:", value)
                return true
        })

        // Output:
        // key: 9 , value: 1
        // key: 8 , value: 2
        // key: 7 , value: 3
        // key: 6 , value: 4
        // key: 5 , value: 5
        // key: 4 , value: 6
        // key: 3 , value: 7
        // key: 2 , value: 8
        // key: 1 , value: 9
        // key: 0 , value: 10
}
```

IteratorDescFrom

- IteratorDescFromftree key keymatchtruekeyftruefalse
-

```
IteratorDescFrom(key interface{}, match bool, f func(key, value
interface{}) bool)
```

-

```
func ExampleBTree_IteratorDescFrom() {
        m := make(map[interface{}]interface{})
        for i := 1; i <= 5; i++ {
                m[i] = i * 10
        }
        tree := gtree.NewBTreeFrom(3, gutil.ComparatorInt, m)

        tree.IteratorDescFrom(5, true, func(key, value interface{})
bool {
                fmt.Println("key:", key, ", value:", value)
                return true
        })

        // Output:
        // key: 5 , value: 50
        // key: 4 , value: 40
        // key: 3 , value: 30
        // key: 2 , value: 20
        // key: 1 , value: 10
}
```

MarshalJson

- MarshalJSONjson.Marshal
-

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

-

```
func ExampleBTree_MarshalJSON() {
        tree := gtree.NewBTree(3, gutil.ComparatorString)
        for i := 0; i < 6; i++ {
                tree.Set("key"+gconv.String(i), "val"+gconv.String
(i))
        }

        bytes, err := json.Marshal(tree)
        if err == nil {
                fmt.Println(gconv.String(bytes))
        }

        // Output:
        // {"key0":"val0","key1":"val1","key2":"val2","key3":"val3","
key4":"val4","key5":"val5"}
}
```