Maps explained: create, add, get, delete

yourbasic.org/golang

Go maps are implemented by hash tables and have efficient add, get and delete operations.

Create a new map

```
var m map[string]int               // nil map of string-int pairs

m1 := make(map[string]float64)      // Empty map of string-float64 pairs
m2 := make(map[string]float64, 100) // Preallocate room for 100 entries

m3 := map[string]float64{           // Map literal
    "e":  2.71828,
    "pi": 3.1416,
}
fmt.Println(len(m3))                // Size of map: 2
```

- A map (or dictionary) is an **unordered** collection of **key-value** pairs, where each key is **unique**.
- You create a new map with a **make** statement or a **map literal**.
- The default **zero value** of a map is `nil`. A nil map is equivalent to an empty map except that **elements can't be added**.
- The **`len`** function returns the **size** of a map, which is the number of key-value pairs.

> **Warning:** If you try to add an element to an uninitialized map you get the mysterious run-time error *Assignment to entry in nil map*.

Add, update, get and delete keys/values

Related

2 basic set implementations
To implement a set in Go you can use a key-value map with boolean or empty struct values.
yourbasic.org

Get slices of keys and values from a map
You can use a range statement to extract slices of keys and values from a map in Go.
yourbasic.org

Sort a map by key or value
To sort a map (which is an unordered collection) you must maintain a separate data structure, typically a slice of keys or values.
yourbasic.org

Hash tables explained [step-by-step example]
Hash tables are used to implement dictionary and map data structures. They offer a combination of efficient lookup, insert and delete operations.
yourbasic.org

Amortized time complexity
Amortized analysis considers both the cheap and expensive operations performed by an algorithm. It is used for algorithms that have expensive operations that happen only rarely.
yourbasic.org

Most Read

Do you make these Go coding mistakes?
Why Go? – Key advantages you may have overlooked
Go string handling overview [cheat sheet]
Type, value and equality of interfaces
Concurrent programming

See all 178 Go articles

```
m := make(map[string]float64)

m["pi"] = 3.14             // Add a new key-value pair
m["pi"] = 3.1416           // Update value
fmt.Println(m)             // Print map: "map[pi:3.1416]"

v := m["pi"]               // Get value: v == 3.1416
v = m["pie"]               // Not found: v == 0 (zero value)

_, found := m["pi"]        // found == true
_, found = m["pie"]        // found == false

if x, found := m["pi"]; found {
    fmt.Println(x)
}                           // Prints "3.1416"

delete(m, "pi")             // Delete a key-value pair
fmt.Println(m)              // Print map: "map[]"
```

- When you index a map you get two return values; the second one (which is optional) is a boolean that indicates if the key exists.
- If the key doesn't exist, the first value will be the default zero value.

For-each range loop

```
m := map[string]float64{
    "pi": 3.1416,
    "e":  2.71828,
}
fmt.Println(m) // "map[e:2.71828 pi:3.1416]"

for key, value := range m { // Order not specified
    fmt.Println(key, value)
}
```

- Iteration order is not specified and may vary from iteration to iteration.
- If an entry that has not yet been reached is removed during iteration, the corresponding iteration value will not be produced.
- If an entry is created during iteration, that entry may or may not be produced during the iteration.

> Starting with Go 1.12, the fmt package prints maps in key-sorted order to ease testing.

Performance and implementation

- Maps are backed by hash tables.
- Add, get and delete operations run in **constant** expected time. The time complexity for the add operation is amortized.
- The comparison operators == and != must be defined for the key type.

Go step by step

Core Go concepts: interfaces, structs, slices, maps, for loops, switch statements, packages.

Share this page: