

# Map! ☐☐☐☐

11

1. 00	1
2. 0000000000	1
3. 0000000000	1
4. value000	2
5. keyvalue000	3
6. key000	5
7. 00000000	6

1. ☐ ☐

```
map key value map map hash! map series map map! hash! object!
```

## 2. □□□□□□□□□□

```
#(<key> <value>...)
```

```
<key>      : 00000000, 0000000000000000: scalar!,all-word!, any-string!
```

```
<value> : any-type![]
```

### 3. □□□□□□□□

```
make map! <spec>
```

```
<spec> : key[]value[][][]integer[]
```

```
spec integer map! map
```

□□:

- `map` `spec`
- `reduce` `logic!`

11

```
#(a: 1 b: 2)
== #(
  a: 1
  b: 2
)

make map! [a 1 'b 2 "c" 3]
== #(
  a: 1
  b: 2
  "c" 3
)
```

key any-word! map key set-word!  
 map key value word key set-  
 word key word map keys-of  
 set-word word set-  
 word word

NOTE:

- hash! block! map! 空のmapを返す
- none key key
- map
- series! value map

map map copy

## 4. value

```
<map>/<key>
get '<map>/<key>

<map> : map! word
<key> : word key
```

select

```
select <map> <key>

<map> : map
<key> : key
```

map 的 get 方法返回一个 Option 类型的值，如果 key 存在，则返回 Some 类型的值，否则返回 None。/case 用于模式匹配。

```
get/case '<map>/<key>  
select/case <map> <key>
```

如果 key 不存在，则返回 None。

例如：

```
m: #(Ab: 2 aB: 5 ab: 10)  
m/ab  
== 2  
select m 'aB  
== 2  
get/case 'm/aB  
== 5  
select/case m 'ab  
== 10
```

## 5. key-value 对

Map 类型支持 key-value 对。

```
<map>/<key>: <value>  
set '<map>/<key> <value>  
  
<map> : map!word  
<key> : map!value!word!key  
<value> : value
```

Map 类型支持 key-value 对。

```
put <map> <key> <value>  
  
<map> : map  
<key> : map!value!key
```

Map 类型支持 key-value 对。

```
extend <map> <spec>  
  
<map> : map  
<spec> : 1
```

map 的更新操作分为三种：set/case、put/case 和 extend/case。其中，set/case 和 put/case 都需要指定要更新的 key，而 extend/case 只需要指定要更新的 map。

```
set/case '<map>/<key> <value>'  
put/case <map> <key> <value>  
extend/case <map> <spec>
```

extend 操作会遍历 map 中的每个 key，并对每个 key 应用指定的 spec。

NOTE:

- map 操作会遍历 map 中的每个 key，并对每个 key 应用指定的 spec。
- extend 操作会遍历 map 中的每个 key，并对每个 key 应用指定的 spec。

map

```

m: #(Ab: 2 aB: 5 ab: 10)
m/ab: 3
m
== #(
  Ab: 3
  aB: 5
  ab: 10
)

put m 'aB "hello"
m
== #(
  Ab: "hello"
  aB: 5
  ab: 10
)

set/case 'm/aB 0
m
== #(
  Ab: "hello"
  aB: 0
  ab: 10
)
set/case 'm/ab 192.168.0.1
== #(
  Ab: "hello"
  aB: 0
  ab: 192.168.0.1
)

m: #(%cities.red 10)
extend m [%cities.red 99 %countries.red 7 %states.red 27]
m
== #(
  %cities.red 99
  %countries.red 7
  %states.red 27
)

```

## 6. key

map key/value key none

```
m: #(a: 1 b: 2 "c" 3 d: 99)
m
== #(
  a: 1
  b: 2
  "c" 3
  d: 99
)
m/b: none
put m "c" none
extend m [d #[none]]
m
== #(
  a: 1
```

**NOTE**

□□□□□□□□□□□□□□□□ none! □ word!  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ spec□□block□□□□□□□□

```
clear 00000000000000000000key000000000000
```

```
clear #(a 1 b 2 c 3)
== #()
```

7. □□□□□□□□

- `find` `key` `map` `true` `none`

```
find #(a 123 b 456) 'b
== true
```

- length? map key/value

```
length? #(a 123 b 456)
== 2
```

- `keys-of` `map` `key` `block` `set-word` `word`

```
keys-of #(a: 123 b: 456)
== [a b]
```

- **values-of** `map value block`

```
values-of #(a: 123 b: 456)
== [123 456]
```

- **body-of** `map` `key/value` `block`

```
body-of #(a: 123 b: 456)
== [a: 123 b: 456]
```