链滴

Go 边看边练 -《Go 学习笔记》系列（九）

作者：88250

原文链接：https://ld246.com/article/1438596722873

来源网站：链滴

上一篇： https://hacpai.com/article/1438311936449
<hr>
<h3 id="ToC">ToC</h3>

Go 边看边练 -《Go 学习笔记》系列（一）- 变量、常量
Go 边看边练 -《Go 学习笔记》系列（二）- 类型、字符串
Go 边看边练 -《Go 学习笔记》系列（三）- 指针
Go 边看边练 -《Go 学习笔记》系列（四）- 控制流 1
Go 边看边练 -《Go 学习笔记》系列（五）- 控制流 2
Go 边看边练 -《Go 学习笔记》系列（六）- 函数
Go 边看边练 -《Go 学习笔记》系列（七）- 错误处理
Go 边看边练 -《Go 学习笔记》系列（八）- 数组、切片
Go 边看边练 -《Go 学习笔记》系列（九）- Map、结构体
Go 边看边练 -《Go 学习笔记》系列（十）- 方法
Go 边看边练 -《Go 学习笔记》系列（十一）表达式
Go 边看边练 -《Go 学习笔记》系列（十二）接口
Go 边看边练 -《Go 学习笔记》系列（十三）Goroutine
Go 边看边练 -《Go 学习笔记》系列（十四）Channel

<hr>
<h3 id="4-3-Map">4.3 Map</h3>
引用类型，哈希表。键必须是支持相等运算符 (==、!=) 类型，比如 <code>number</code> <code>string</code>、<code>pointer</code>、<code>array</code>、<code>struct</code>，以及对应的 <code>interface</code>。值可以是任意类型，没有限制。

```
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">m := map[int]struct {
</span></span><span class="highlight-line"><span class="highlight-cl">    name string
</span></span><span class="highlight-line"><span class="highlight-cl">    age int
</span></span><span class="highlight-line"><span class="highlight-cl">}{
</span></span><span class="highlight-line"><span class="highlight-cl">    1: {"user1", 10},
/ 可省略元素类型。
</span></span><span class="highlight-line"><span class="highlight-cl">    2: {"user2", 20},
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">println(m[1].name

</span></span></code></pre>
<p>预先给 <code>make</code> 函数一个合理元素数量参数，有助于提升性能。因为事先申请
大块内存，可避免后续操作时频繁扩张。</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">m := make(map[string]int, 1000)
</span></span></code></pre>
<p>常见操作：</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">m := map[string]int{
</span></span><span class="highlight-line"><span class="highlight-cl">    "a": 1,
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">if v, ok := m["a"];
k { // 判断 key 是否存在。
</span></span><span class="highlight-line"><span class="highlight-cl">    println(v)
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">println(m["c"]) //
于不存在的 key，直接返回 \0，不会出错。
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">m["b"] = 2 // 新
或修改。
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">delete(m, "c") //
除。如果 key 不存在，不会出错。
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">println(len(m)) //
获取键值对数量。cap 无效。
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">for k, v := range m
{ // 迭代，可仅返回 key。随机顺序返回，每次都不相同。
</span></span><span class="highlight-line"><span class="highlight-cl">    println(k, v)
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
<p>不能保证迭代返回次序，通常是随机结果，具体和版本实现有关。</p>
<p>从 <code>map</code> 中取回的是一个 <code>value</code> 临时复制品，对其成员的修
是没有任何意义的。</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">type user struct{ name string }
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">m := map[int]user{
// 当 map 因扩张而重新哈希时，各键值项存储位置都会发生改变。 因此，map
```

```
    1: {"user1"}, // 设计成 not addressable。 类似 m[1].name 这种期望透过原 value
} // 指针修改成员行为自然会被禁止。

m[1].name = "To" // Error: cannot assign to m[1].name
```

正确做法是完整替换 `value` 或使用指针。

```
u := m[1]
u.name = "Tom"
m[1] = u // 替换 value。

m2 := map[int]*usr{
    1: &user{"user1"},
}
m2[1].name = "Jak" // 返回的是指针复制品。透过指针修改原对象是允许的。
```

可以在迭代时安全删除键值。但如果期间有新增操作，那么就不知道会有什么意外了。

<iframe src="https://wide.b3log.org/playground/953c1eef3162b8a3aec5b78d0405e667.go?embed=true" width="99%" height="600" sandbox="allow-scripts allow-same-origin"></iframe>

4.4 Struct

值类型，赋值和传参会复制全部内容。可用 "_" 定义补位字段，支持指向自身类型的指针成员。

```
type Node struct {
    _ int
    id int
    data *byte
    next *Node
}

func main() {
    n1 := Node{
        id: 1,
        data: nil,
    }

    n2 := Node{
        id: 2,
        data: nil,
        next: &1,
    }
}
```

顺序初始化必须包含全部字段，否则会出错。

<pre><code class="highlight-chroma"><span class="highlight
cl">type User struct {
    name string
    age int
}

u1 := User{"Tom",
20}
u2 := User{"Tom"}
// Error: too few values in struct initializer
</code></pre>
支持匿名结构，可用作结构成员或定义变量。
<pre><code class="highlight-chroma"><span class="highlight
cl">type File struct {
    name string
    size int
    attr struct {
        perm int
        owner int
    }
}

f := File{
    name: "test.txt",
    size: 1025,
    // attr: {0755, 1},
// Error: missing type in composite literal
}

f.attr.owner = 1
f.attr.perm = 0755

var attr = struct {
    perm int
    owner int
}{2, 0755}

f.attr = attr
</code></pre>
支持 "=="、"!=" 相等操作符，可用作 <code>map</code> 键类型。
<pre><code class="highlight-chroma"><span class="highlight
cl">type User struct {
    id int
    name string
}

m := map[User]int

    User{1, "Tom"}:
00,
}
</code></pre>
可定义字段标签，用反射读取。标签是类型的组成部分。
<pre><code class="highlight-chroma"><span class="highlight

cl">var u1 struct { name string "username" }
var u2 struct { na
e string }

u2 = u1 // Error: c
nnot use u1 (type struct { name string "username" }) as
            // type str
ct { name string } in assignment
</code></pre>
空结构 "节省" 内存，比如用来实现 <code>set</code> 数据结构，或者实现没有 "状态" 只有法的 "静态类"。
<pre><code class="highlight-chroma"><span class="highlight
cl">var null struct{}

set := make(map[s
ring]struct{})
set["a"] = null
</code></pre>
<h4 id="4-4-1-匿名字段">4.4.1 匿名字段</h4>
匿名字段不过是一种语法糖，从根本上说，就是一个与成员类型同名 (不含包名) 的字段。被匿嵌入的可以是任何类型，当然也包括指针。
<pre><code class="highlight-chroma"><span class="highlight
cl">type User struct {
    name string
}

type Manager str
ct {
    User
    title string
}

m := Manager{
    User: User{"To
"}, // 匿名字段的显式字段名，和类型名相同。
    title: "Administr
tor",
}
</code></pre>
可以像普通字段那样访问匿名字段成员，编译器从外向内逐级查找所有层次的匿名字段，直到发目标或出错。
<pre><code class="highlight-chroma"><span class="highlight
cl">type Resource struct {
    id int
}

type User struct {
    Resource
    name string
}

type Manager str
ct {
    User

```
    title string
}

var m Manager
m.id = 1
m.name = "Jack"
m.title = "Administrator"
```

外层同名字段会遮蔽嵌入字段成员，相同层次的同名字段也会让编译器无所适从。解决方法是使显式字段名。

```
type Resource struct {
    id int
    name string
}

type Classify struct {
    id int
}

type User struct {
    Resource // Resource.id 与 Classify.id 处于同一层次。
    Classify
    name string // 遮蔽 Resource.name。
}

u := User{
    Resource{1, "people"},
    Classify{100},
    "Jack",
}

println(u.name) // User.name: Jack
println(u.Resource.name) // people

// println(u.id) // Error: ambiguous selector u.id
println(u.Classify.i) // 100
```

不能同时嵌入某一类型和其指针类型，因为它们名字相同。

```
type Resource struct {
    id int
}

type User struct {
```

    *Resource
    // Resource // E
ror: duplicate field Resource
    name string
}

u := User{
    &Resource
1},
    "Administrator",
}

println(u.id)
println(u.Resource.
d)
</code></pre>
<h4 id="4-4-2-面向对象">4.4.2 面向对象</h4>
面向对象三大特征里，Go 仅支持封装，尽管匿名字段的内存布局和行为类似继承。没有 <code
class</code> 关键字，没有继承、多态等等。
<pre><code class="highlight-chroma"><span class="highlight
cl">type User struct {
    id int
    name string
}

type Manager str
ct {
    User
    title string
}

m := Manager{Us
r{1, "Tom"}, "Administrator"}

// var u User = m
/ Error: cannot use m (type Manager) as type User in assignment

/ 没有继承，自然也不会有多态。

var u User = m.Us
r // 同类型拷贝。
</code></pre>
内存布局和 C struct 相同，没有任何附加的 object 信息。

可用 <code>unsafe</code> 包相关函数输出内存地址信息。
<pre><code class="highlight-chroma"><span class="highlight
cl">m : 0x2102271b0, size: 40, align: 8
m.id : 0x2102271
0, offset: 0
m.name : 0x2102
71b8, offset: 8
m.title: 0x2102271
8, offset: 24
</code></pre>

下一篇： https://hacpai.com/article/1438699210452
<hr>

本系列是基于雨痕的《Go 学习笔记》（第四）整理汇编而成，非常感谢雨痕的辛勤付出与分享！
转载请注明：文章转载自：黑客与画家的社区 [https://acpai.com]
如果你觉得本章节做得不错，请在下面打赏一下吧~

<hr>
<blockquote>
社区小贴士

关注标签 [golang] 可以方便查看 Go 相关帖子
关注作者后如有新帖将会收到通知

</blockquote>