Golang Operator Overloading

Asked 5 years, 6 months ago Active 2 years, 10 months ago Viewed 17k times

8

I understand that golang does not provide operator overloading, as it believe that it is increasing the complexity.

So I want to implement that for structures directly.

```
package main

import "fmt"

type A struct {
    value1 int
    value2 int
}

func (a A) AddValue(v A) A {
    a.value1 += v.value1
    a.value2 += v.value2
    return a
}


func main() {
    x, z := A{1, 2}, A{1, 2}
    y := A{3, 4}

    x = x.AddValue(y)

    z.value1 += y.value1
    z.value2 += y.value2

    fmt.Println(x)
    fmt.Println(z)
}
```

https://play.golang.org/p/1U8omyF8-V

From the above code, the ***AddValue*** works as I want to. However, my only concern is that it is a pass by value and hence I have to return the newly added value everytime.

Is there any other better method, in order to avoid returning the summed up variable.

go methods struct operator-overloading

Share Edit Follow Flag

edited Jun 13 '18 at 15:58
icza
284k ● 42 ● 648 ● 622

asked Oct 9 '15 at 14:11
Sundar
1,499 ● 1 ● 10 ● 19

3 Yes, use pointer receiver. – icza Oct 9 '15 at 14:13

1 Answer

Active Oldest Votes

21

Yes, use pointer receiver:

```
func (a *A) AddValue(v A) {
    a.value1 += v.value1
    a.value2 += v.value2
}
```

By using a pointer receiver, the address of a value of type `A` will be passed, and therefore if you modify the pointed object, you don't have to return it, you will modify the "original" object and not a copy.

You could also simply name it `Add()`. And you could also make its argument a pointer (for consistency):

```
func (a *A) Add(v *A) {
    a.value1 += v.value1
    a.value2 += v.value2
}
```

And so using it:

```
x, y := &A{1, 2}, &A{3, 4}

x.Add(y)

fmt.Println(x)  // Prints &{4 6}
```

Notes

Note that even though you now have a pointer receiver, you can still call your `Add()` method on non-pointer values if they are addressable, so for example the following also works:

```
a, b := A{1, 2}, A{3, 4}
a.Add(&b)
fmt.Println(a)
```

`a.Add()` is a shorthand for `(&a).Add()`. Try these on the Go Playground.

Share Edit Follow Flag

edited Feb 10 '17 at 13:53

answered Oct 9 '15 at 14:13

icza

284k ● 42 ● 648 ● 622

Great. This works as expected. Question: When I make the argument also as pointer, then will it help in performance, by passing it as reference instead of value. – Sundar Oct 9 '15 at 14:37

2 @Sundar In case of small structs it doesn't matter and you shouldn't care. If your structs are big, then yes, it (may) help. For a discussion on the topic, see [Pointers vs. values in parameters and return values](). – icza Oct 9 '15 at 14:52