Go Booby Traps 6

Michel Casabianca
casa@sweetohm.net

Go programming language is easy to learn, but there are some tricky traps. This article series is trying to show these booby traps so that you avoid them.

We would like to reverse a slice. It sounds like a job for Generics, so we write following code:

```
package main

import "fmt"

func reverse[S []any](s S) {
        for i, j := 0, len(s)-1; i < j; i, j = i+1, j-1 {
                s[i], s[j] = s[j], s[i]
        }
}

func main() {
        slice := []int{1, 2, 3}
        reverse(slice)
        fmt.Printf("%#v\n", slice)
}
```

On the Playground

But it doesn't compile with following error message: `[]int does not implement []any`! Why doesn't it compile? Could you fix it?

Explanation

First, we must say that this issue is not related to Generics and is more fundamental. We could summarize this issue with this question: why can't we send type `[]int` to a function that takes `[]interface{}`?

We will demonstrate that with a proof by reduction to the absurd. Let's say that we can send `[]int` to function `F([]interface{})`, then following example should compile:

```
package main

func F(v []interface{}) {
        v[0] = "Oops!"
}

func main() {
        v := []int{1, 2, 3}
```

```
        F(v)
        println(v[0])
}
```

On the Playground

This example doesn't compile with following error message: `cannot use v (variable of type []int) as type []interface{} in argument to F` and we understand why: this would assign a string into a integer slice!

To go back to our first example, we must find a way to describe parameter type for our `reverse()` function without using `[]any`.

We can fix the example with following code:

```
package main

import "fmt"

func reverse[S []E, E any](s S) {
        for i, j := 0, len(s)-1; i < j; i, j = i+1, j-1 {
                s[i], s[j] = s[j], s[i]
        }
}

func main() {
        slice := []int{1, 2, 3}
        reverse(slice)
        fmt.Printf("%#v\n", slice)
}
```

Sur le Playground

We say to compiler: the parameter of the function is a slice of type `E` and type `E` might be anything. Compiler is happy and there you are!

For a more detailed explanation, see this article.

Enjoy!