Table of Contents

 Welcome 1.1

Go

Go 2.1

- Go Documents 2.1.1
  - errors 2.1.1.1
- Go Gotchas 2.1.2
  - Assignment to entry in nil map 2.1.2.1
  - Invalid memory address or nil pointer dereference 2.1.2.2
  - Array won't change 2.1.2.3
  - How does characters add up? 2.1.2.4
  - What happened to ABBA? 2.1.2.5
  - Where is my copy? 2.1.2.6
  - Why doesn’t append work every time? 2.1.2.7
- Go Blog 2.1.3

Weekly

2020 3.1

- Weekly 01 3.1.1
- Weekly 02 3.1.2

Introduction

A bunch of programming documents.

Go

A bunch of Go learning stuffs.

Go Doc

Go Standard library Translation

errors

本文是 Go 标准库中 errors 包文档的翻译， 原文地址为：
https://golang.org/pkg/errors/

概述

errors 包实现了用于处理错误的函数。

示例：

```
package main

import (
    "fmt"
    "time"
)

// MyError 是一个包含了时间和消息的错误实现
type MyError struct {
    When time.Time
    What string
}

func (e MyError) Error() string {
    return fmt.Sprintf("%v: %v", e.When, e.What)
}

func oops() error {
    return MyError{
        time.Date(1989, 3, 15, 22, 30, 0, 0, time.UTC),
        "the file system has gone away",
    }
}

func main() {
    if err := oops(); err != nil {
        fmt.Println(err)
    }
}
```

示例执行结果：

```
1989-03-15 22:30:00 +0000 UTC: the file system has gone away
```

New 函数

```
func New(text string) error
```

根据给定的文本返回一个错误。

示例：

```
package main

import (
    "errors"
    "fmt"
)

func main() {
    err := errors.New("emit macho dwarf: elf header corrupted")
    if err != nil {
        fmt.Print(err)
    }
}
```

示例执行结果：

```
emit macho dwarf: elf header corrupted
```

fmt 包的 Errorf 函数可以让用户使用该包的格式化功能来创建描述错误的消息。

示例：

```
package main

import (
    "fmt"
)

func main() {
    const name, id = "bimmler", 17
    err := fmt.Errorf("user %q (id %d) not found", name, id)
    if err != nil {
        fmt.Print(err)
    }
}
```

示例执行结果：

```
user "bimmler" (id 17) not found
```

Go Gotchas

This collection of Go gotchas and pitfalls will help you find and fix similar problems in your own code.

Assignment to entry in nil map

Why does this program panic?

```
var m map[string]float64
m["pi"] = 3.1416
```

```
# Output
panic: assignment to entry in nil map
```

Answer

You have to initialize the map using the make function (or a map literal) before you can add any elements:

```
m := make(map[string]float64)
m["pi"] = 3.1416
```

Invalid memory address or nil pointer dereference

Why does this program panic?

```
package main

import (
    "math"
    "fmt"
)

type Point struct {
    X, Y float64
}

func (p *Point) Abs() float64 {
    return math.Sqrt(p.X*p.X + p.Y*p.Y)
}

func main() {
    var p *Point
    fmt.Println(p.Abs())
}
```

```
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x0 pc=0x499043]

goroutine 1 [running]:
main.(*Point).Abs(...)
    /tmp/sandbox466157223/prog.go:13
main.main()
    /tmp/sandbox466157223/prog.go:18 +0x23
```

Answer

The uninitialized pointer p in the main function is nil, and you can't follow the nil pointer.

> If x is nil, an attempt to evaluate *x will cause a run-time panic.
>
> — The Go Programming Language Specification: Address operators

You need to create a Point

```
func main() {
    var p *Point = new(Point)
    fmt.Println(p.Abs())
}
```

Since methods with pointer receivers take either a value or a pointer, you could also skip the pointer altogether:

```
func main() {
    var p Point // has zero value Point{X:0, Y:0}
    fmt.Println(p.Abs())
}
```

Array won’t change

Why does the array value stick?

```
package main

import "fmt"

func Foo(a [2]int) {
    a[0] = 6
}

func main() {
    a := [2]int{1, 2}
    Foo(a) // Try to change a[0].
    fmt.Println(a) // Output: [1 2]
}
```

Answer

> - Arrays in Go are **values**
> - When you pass an array to a function, **the array is copied.**

If you want to Foo to update the elements of a function, use a **Slice** instead.

```
package main

import "fmt"

func Foo(a []int) {
    if len(a) > 0 {
        a[0] = 6
    }
}

func main() {
    a := []int{1, 2}
    Foo(a) // Change a[0].
    fmt.Println(a)  // Output: [6 2]
}
```

A slice does not store any data, it just describes a section of an underlying array.

When you change an element of a slice, you modify the corresponding element of its underlying array, and other slices that share the same underlying array will see the change.

How does characters add up?

Why doesn’t these print statements give the same result?

```
fmt.Println("H" + "i")
fmt.Println('H' + 'i')

// Output:
// Hi
// 177
```

Answer

The rune literals 'H' and 'i' are integer values identifying Unicode code points: 'H' is 72 and 'i' is 105.

You can turn a code point into a string with a conversion.

```
fmt.Println(string(72) + string('i')) // "Hi"
```

You can also use the **fmt.Sprintf** function.

```
s := fmt.Sprintf("%c%c, world!", 72, 'i')
fmt.Println(s)// "Hi, world!"
```

What happened to ABBA?

What's up with strings.TrimRight?

```
fmt.Println(strings.TrimRight("ABBA", "BA")) // Output: ""
```

Answer

The Trim, TrimLeft and TrimRight functions strip all Unicode code points contained in a **cutset**. In this case, all trailing A:s and B:s are stripped from the string, leaving the empty string.

To strip a trailing string, use **strings.TrimSuffix**.

```
fmt.Println(strings.TrimSuffix("ABBA", "BA")) // Output: "AB"
```

Where is my copy?

Why does the copy disappear?

```
var src, dst []int
src = []int{1, 2, 3}
copy(dst, src) // Copy elements to dst from src.
fmt.Println("dst:", dst)

// Output:
// dst: []
```

Answer

The number of elements copied by the copy function is the **minimum of len(dst) and len(src)**. To make a full copy, you must allocate a big enough destination slice.

```
var src, dst []int
src = []int{1, 2, 3}

dst = make([]int, len(src)) // Update Here

copy(dst, src) // Copy elements to dst from src.
fmt.Println("dst:", dst)

// Output:
// dst: [1 2 3]
```

The return value of the copy function is the number of elements copied. See Copy function for more about the built-in copy function in Go.

Using append

You could also use the append function to make a copy by appending to a nil slice.

```
var src, dst []int
src = []int{1, 2, 3}
dst = append(dst, src...)
fmt.Println("dst:", dst)

// Output:
// dst: [1 2 3]
```

Note that the capacity of the slice allocated by append may be a bit larger than len(src).

Why doesn’t append work every time?

What’s up with the append function?

```
a := []byte("ba")

a1 := append(a, 'd')
a2 := append(a, 'g')

fmt.Println(string(a1)) // bag
fmt.Println(string(a2)) // bag
```

Answer

If there is room for more elements, append reuses the underlying array. Let's take a look:

```
a := []byte("ba")
fmt.Println(len(a), cap(a)) // 2 32
```

This means that the slices **a, a1 and a2** will refer to the **same underlying array** in our example.

To avoid this, we need to use two separate byte arrays.

```
const prefix = "ba"

a1 := append([]byte(prefix), 'd')
a2 := append([]byte(prefix), 'g')

fmt.Println(string(a1)) // bad
fmt.Println(string(a2)) // bag
```

Go Blog

Some Go Learning notes.

- A Quick Introduction to Elasticsearch for Node Developers

每周阅读

记录每一周的阅读记录及链接

Week 01(20201024-20201030)

Book

- 《三体III-死神永生》
- 《Javascript 设计模式与开发实践》

Blog

- A Quick Introduction to Elasticsearch for Node Developers

Blockchain

- An Introduction to Binance Smart Chain (BSC)
- Adding Binance Smart Chain and JNTR to your MetaMask

Week 01(20201031-20201106)

Book

- 《三体III-死神永生》
- 《Javascript 设计模式与开发实践》

Blog

- How to be a Better Software Engineer: Book Recommendations
- Best Practices Every Node Developer Should Follow
- Redis + NodeJS 实现一个能处理海量数据的异步任务队列系统
- GitHub Actions 入门教程
- Creating Fast APIs In Go Using Fiber

Tutorial

- basic bash guide

Library

- oclif 命令行工具框架
- Fiber - Go web framework
- Go cobra

Blockchain

- The Best Way To Learn Blockchain Programming
- How to Build Blockchain App - Ethereum Todo List 2019
- Getting Up to Speed on Ethereum