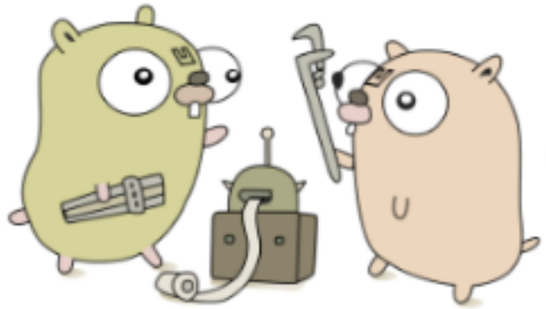
Go1.13简介

柴树杉(chai2010)

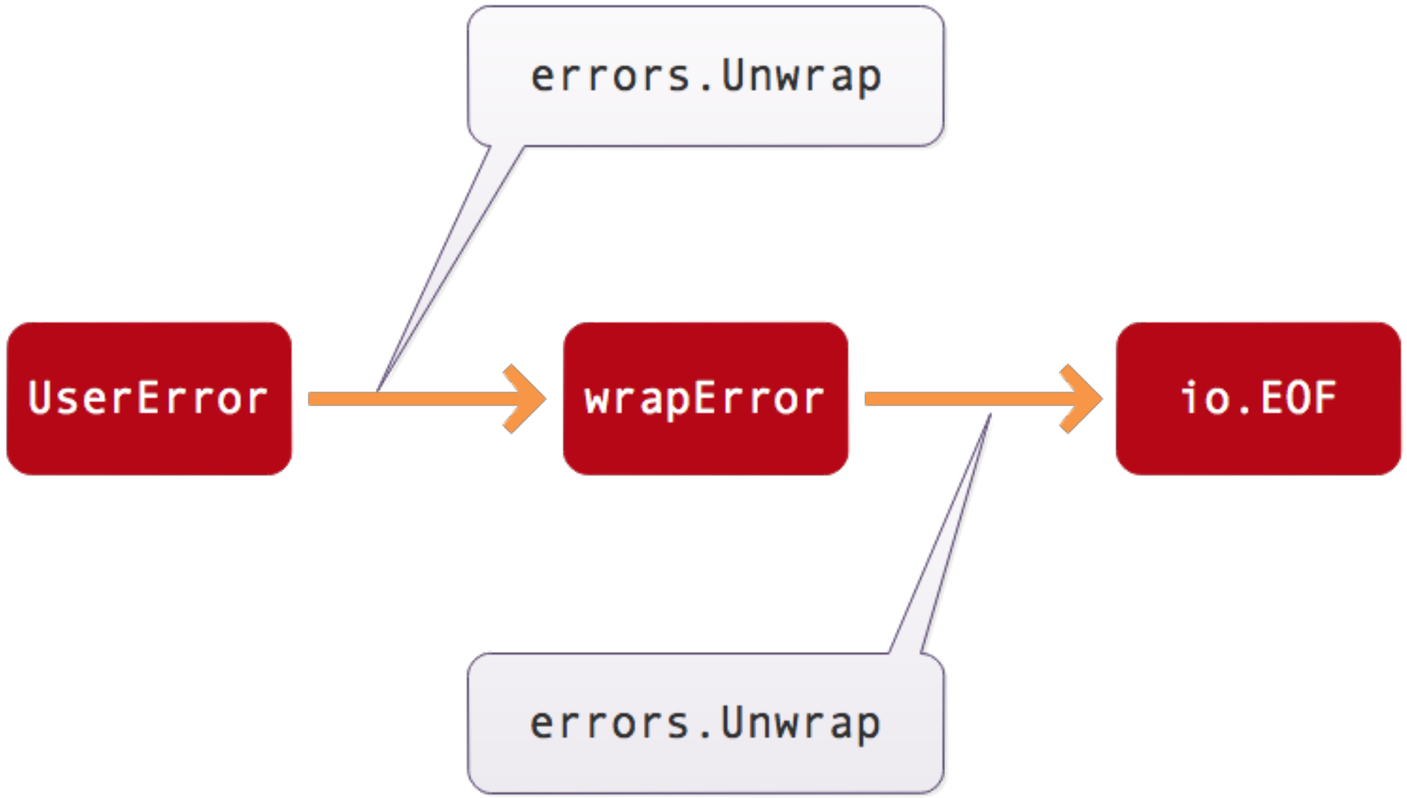
光谷码农



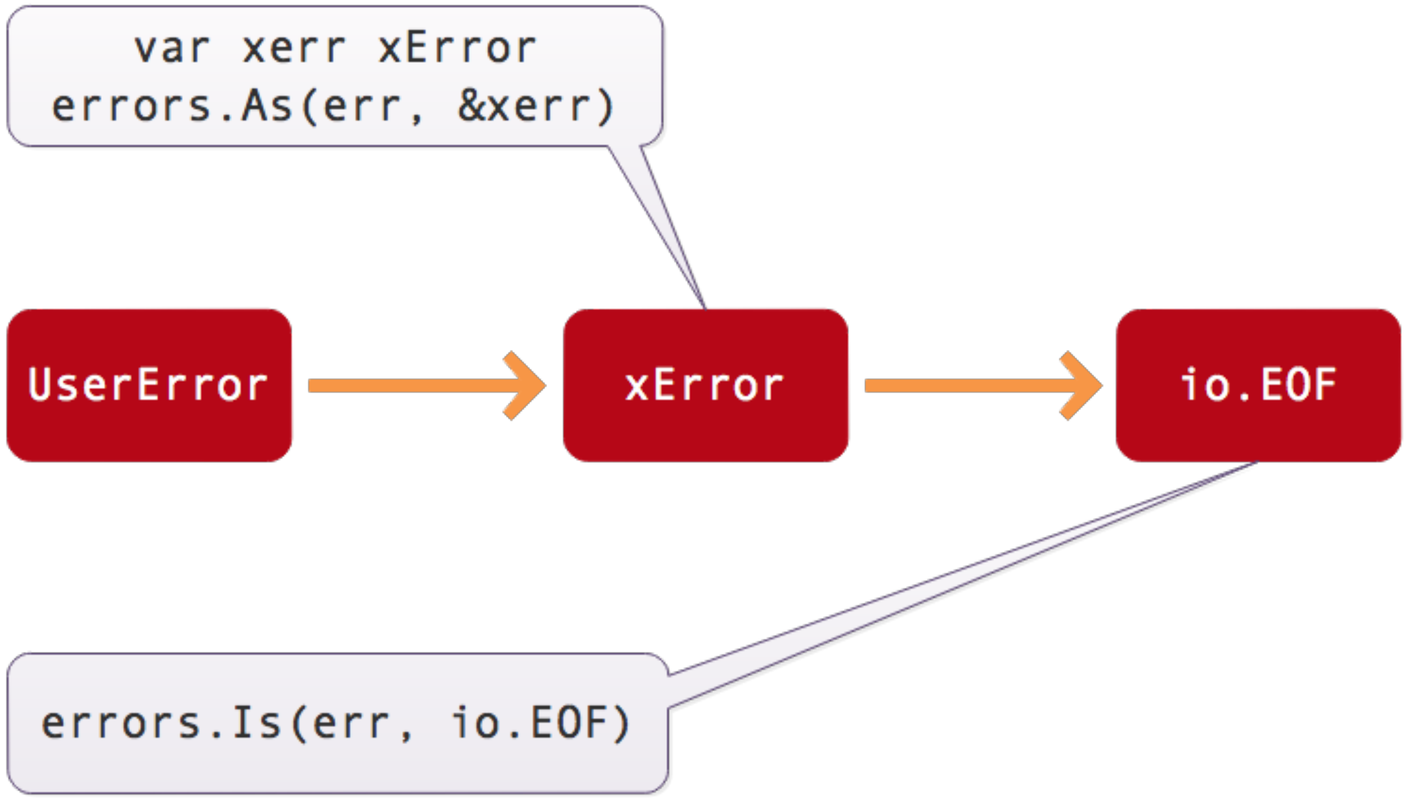
Go1.13两大变更

- 错误演变为错误链
- 数值面值型增强（二进制/八进制/十六进制浮点数/分隔符）

错误链



错误链查询: 类型/值


var xerr xError
errors.As(err, &xerr)
UserError
xError
io.EOF
errors.Is(err, io.EOF)

IEEE754浮点数

浮点数很重要

1. 浮点数设计者**William Kahan**因此获得图灵奖
2. Go之父Rob Pike说：正则和浮点数是每个码农必备技能

浮点数方程

if(x+1 == x): x = ?

Go浮点数基础

打印浮点数

打印浮点数，控制打印精度

```
fmt.Printf("%f\n", float32(0.3)) // 0.300000
fmt.Printf("%.10f\n", float32(0.3)) // 0.3000000119
fmt.Printf("%.20f\n", float32(0.3)) // 0.30000001192092895508
```

浮点数面值

```
fmt.Println(float32(0.25)) // 普通写法
fmt.Println(float32(25E-2)) // 十进制科学记数法
fmt.Println(float32(0x1p-2)) // 十六进制浮点数

fmt.Println(float32(0x_2p-3)) // 下划线分隔
```

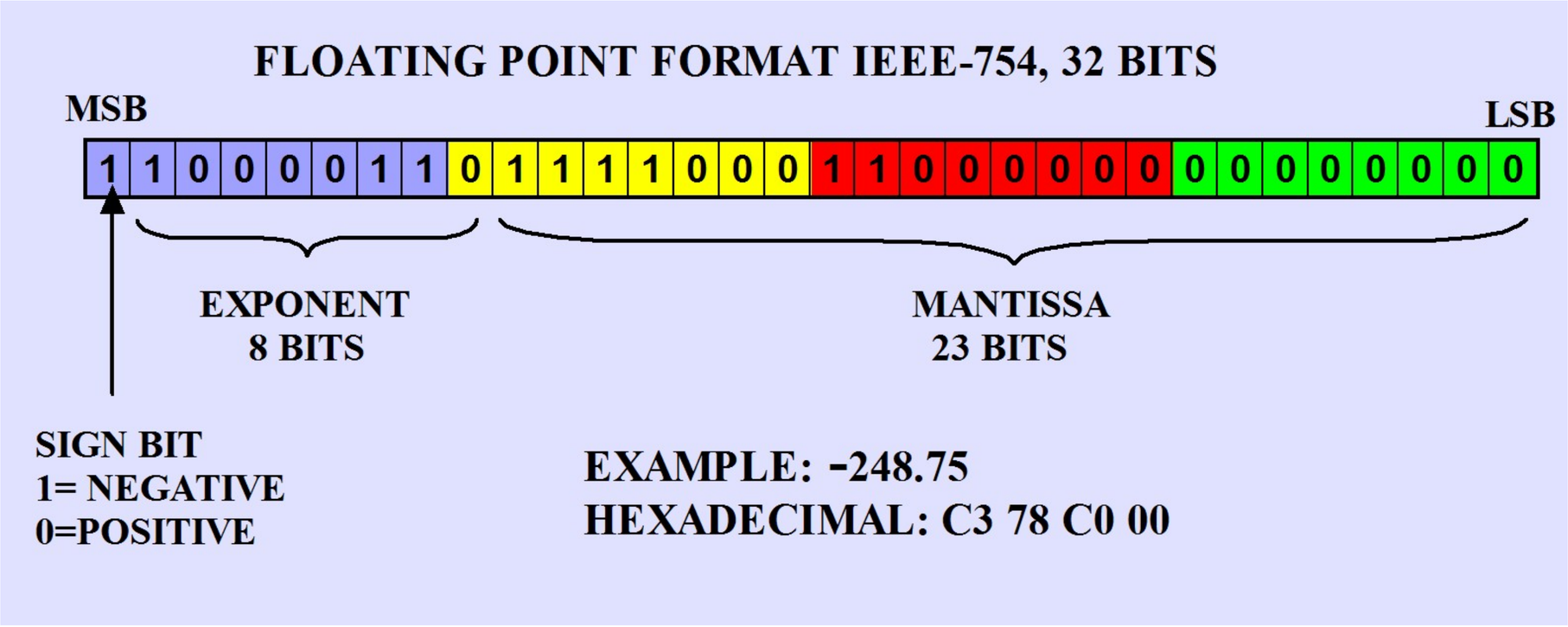
浮点数位模式

```
fmt.Printf("%b\n", math.Float32bits(1))

fmt.Printf("%b\n", (0 << 31) + (127 << 23) + 0)
fmt.Println(math.Float32frombits((0 << 31) + (127 << 23) + 0))

// 11111110000000000000000000000000
// 1 x 10^0
```

浮点数内存布局


FLOATING POINT FORMAT IEEE-754, 32 BITS
MSB
LSB
1 1 0 0 0 0 1 1 0 1 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EXPONENT
8 BITS
MANTISSA
23 BITS
SIGN BIT
1= NEGATIVE
0=POSITIVE
EXAMPLE: -248.75
HEXADECIMAL: C3 78 C0 00

正负零

```
math.Float32frombits((0 << 31) + (  0 << 23) + 0)       // +0

math.Float32frombits((1 << 31) + (  0 << 23) + 0)       // −0
```

- 第31bit为符号位

- 正负零是相等的

无穷和非数

```
math.Float32frombits((0 << 31) + (255 << 23) + 0)      // +Inf

math.Float32frombits((1 << 31) + (255 << 23) + 0)      // -Inf

math.Float32frombits((0 << 31) + (255 << 23) + 1)      // NaN

math.Float32frombits((1 << 31) + (255 << 23) + 2)      // NaN
```

- 第23~31bit为指数位
- 指数为255，有效数字为0，表示无穷
- 指数为255，有效数字不为0，表示非数NaN
- NaN不能比较，和自身也不相等

绝对值最小的数

```
math.Float32frombits((0 << 31) + (  0 << 23) + (1<<0))  // 1e-45

math.Float32frombits((0 << 31) + (  0 << 23) + (1<<1))  // 3e-45

math.Float32frombits((0 << 31) + (  0 << 23) + (1<<2))  // 6e-45
```

- 指数为0，有效数最低为为1
- const math.SmallestNonzeroFloat32 =

  1.401298464324817070923729583289916131280e-45
- // 1 / 2**(127 - 1 + 23)

规范化数

```
math.Float32frombits((0 << 31) + (  0 << 23) + (1<<22)) // 5.877472e-39
math.Float32frombits((0 << 31) + (  1 << 23) + 0)          // 1.1754944e-38
```

- 有效数字整数部分只保留一个bit位（必然是1）
- 因此：如果指数不为0，省略个位数1
- 可以提高一个精度

绝对值最大的数

```
math.Float32frombits((0 << 31) + (254 << 23) + 1<<23-1)

// 3.4028235e+38
```

- 指数最大为254，有效数最大全是1

浮点数FAQ

为何float32有6个精度?

1. float32有效位有23bit，大概可以表示8万多个数
2. 因此精度保留到10万，也就是6个十进制小数位

为何没有0.3?

1. float32只有2^32种状态
2. 但是范围却是：1.4*10^-45 到 3.4*10^38
3. 根据抽屉原理，必然有很多数无法唯一表达
4. 二进制的浮点数全是有二的指数值组合成，0.3无法组合
5. 比如十进制不能表达1/3，但是三进制种可以表示为0.

满足结合律吗?

```
var a = float32(1<<24)
fmt.Println((a+1+1) == (a+(1+1))) // false
```

float32只有32bit有效数，当整数大于1<<23时，从各位开始丢精度

四舍五入?

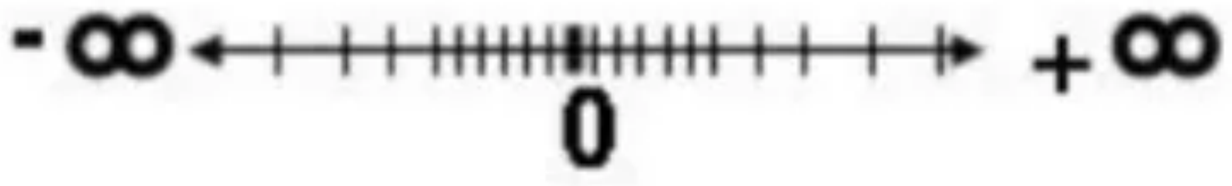
1. 5处于中间位置，因此五入是不公平的
2. IEEE754根据五舍或五入的结果，选择合适的方式
3. 关键是结果要长得漂亮!

指数为何不用补码?

1. 移码就是x-127是真实的指数，因此127是0
2. 原因是负指数对应结果小，因此要保证指数编码后也比较小
3. 好处：可以当作[]int32来排序[]float32

在数轴是怎么分布的?

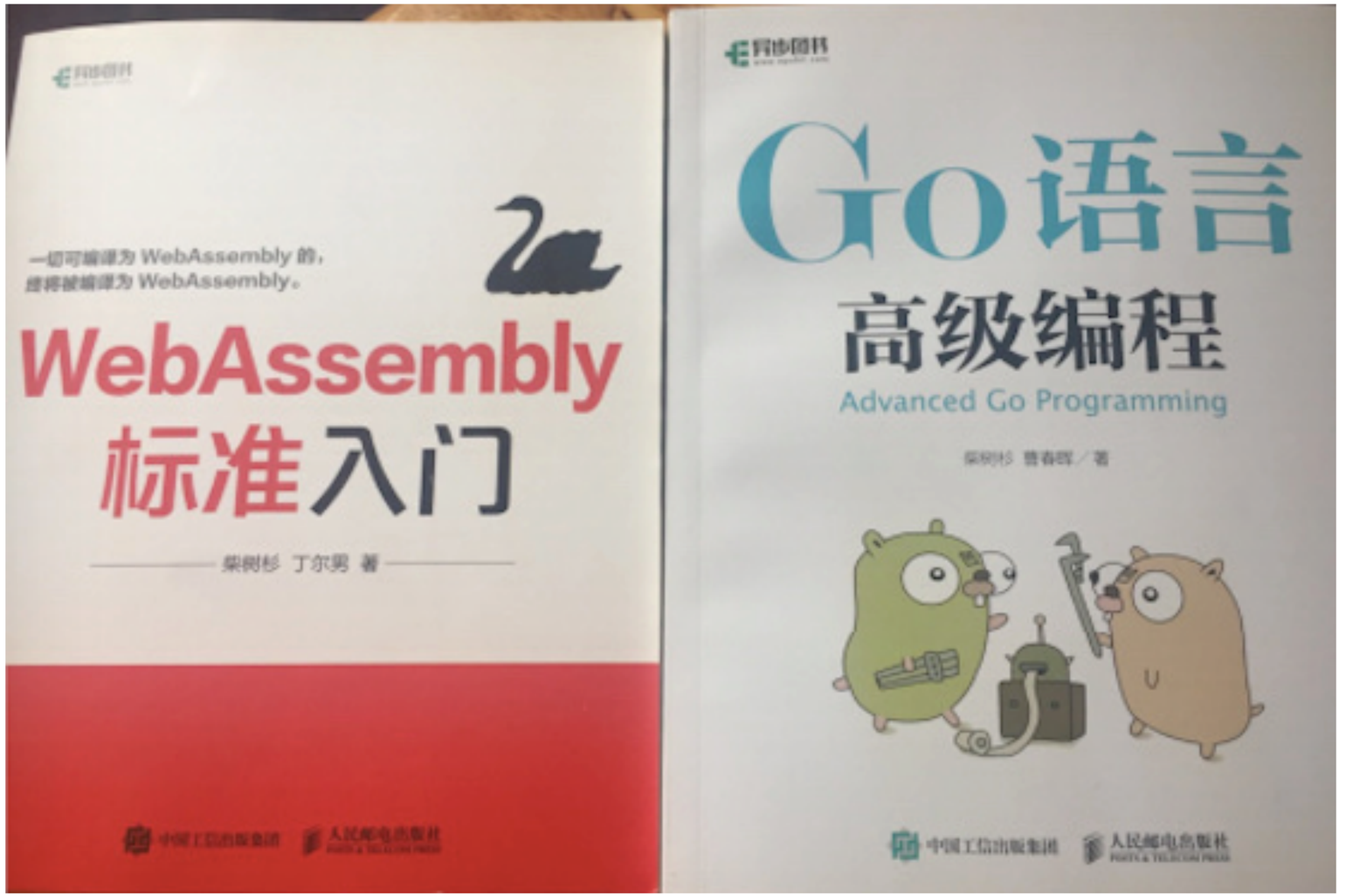
1. 浮点数分布不均匀
2. 越靠近原点越密（相邻小数比整数靠的更近）
3. 同一指数阶码分布均匀
4. 想象一个10厘米长度的毫米直尺，通过不停指数放大或缩小2倍
5. 当放大到毫米宽度大于1米时，直尺就无法精确表示米了
6. 直尺的1000个毫米单位就和浮点数有效位类似的作用



浮点数的四维空间

```
var m = map[float32]string{
    math.Float32frombits((0 << 31) + (255 << 23) + 1): "NaN1",
    math.Float32frombits((0 << 31) + (255 << 23) + 2): "NaN2",
    1<<24 + 1: "1<<24 + 1",
    0.3:       "0.3",
}
```

广告时间


一切可编译为 WebAssembly 的，
终将被编译为 WebAssembly。
WebAssembly
标准入门
柴树杉 丁尔男 著
Go语言
高级编程
Advanced Go Programming
柴树杉 曹春晖 著