可视化学习 Go 并发编程

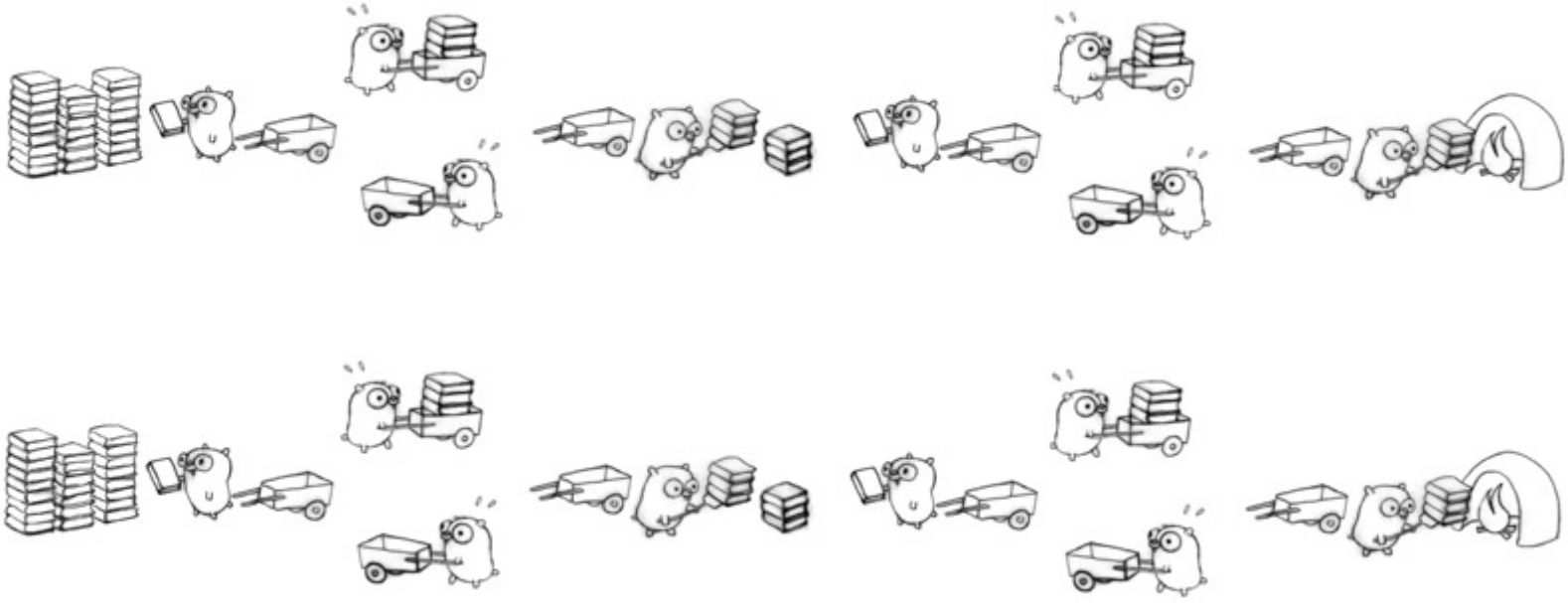
2017.8.5

黄庆兵 - 网易

bingohuang.com

并发

简单来说，并发是一种构造程序的方式



Concurrency is not Parallelism

Slide (http://talks.golang.org/2012/waza.slide)

1．并发很强大
2．并发帮助实现并行，使并行(扩展等)变得容易
3．并发不是并行，两者不同，但相关，并发重点是架构，并行重点是执行。

可视化 并发(Concurrency) & 并行(Parallelism)

- 并行(PARALLELISM)

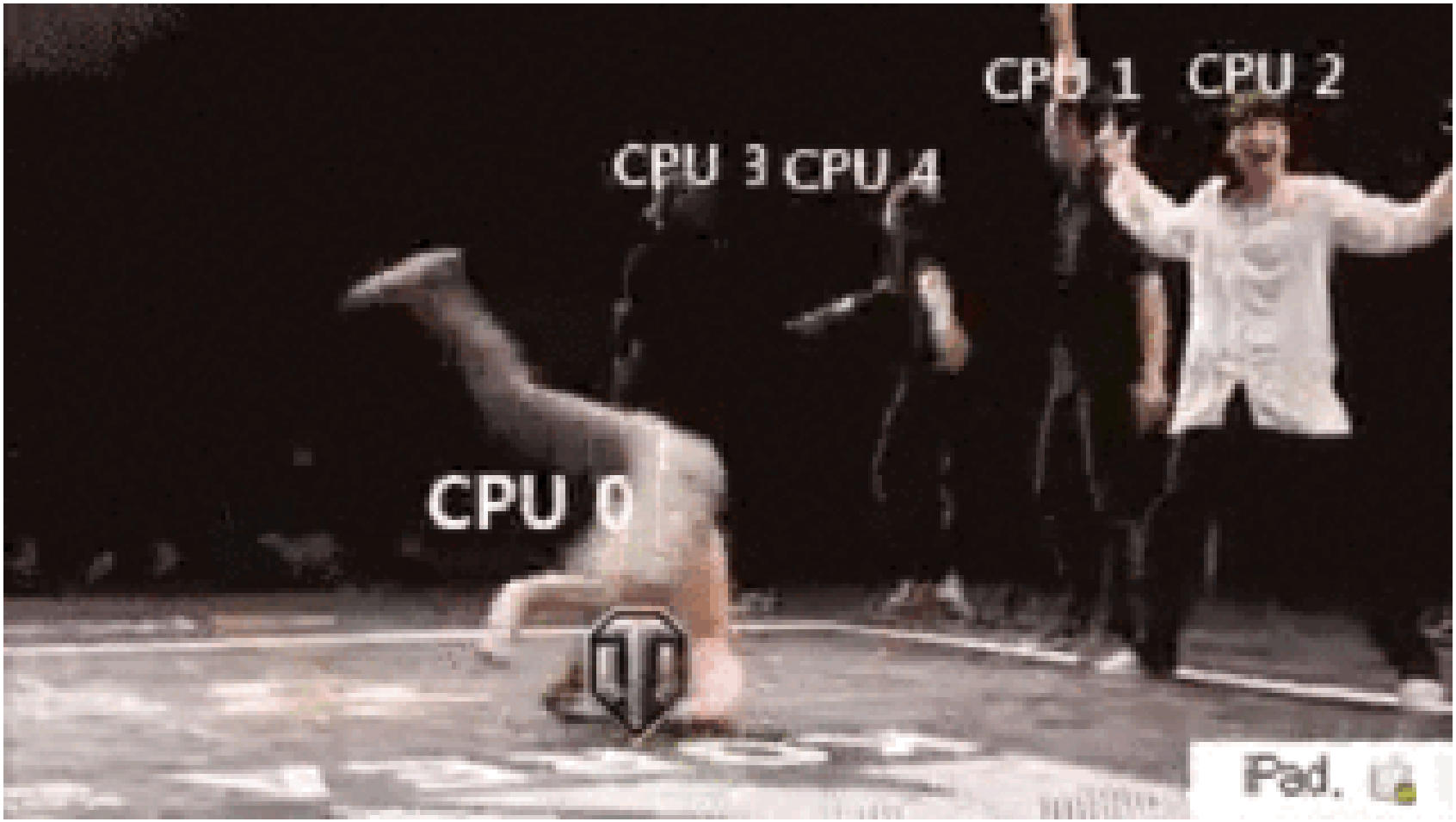
  这是并行 (/2017/go-concurrency-visualize/parallelism.html)

- 并发(CONCURRENCY)

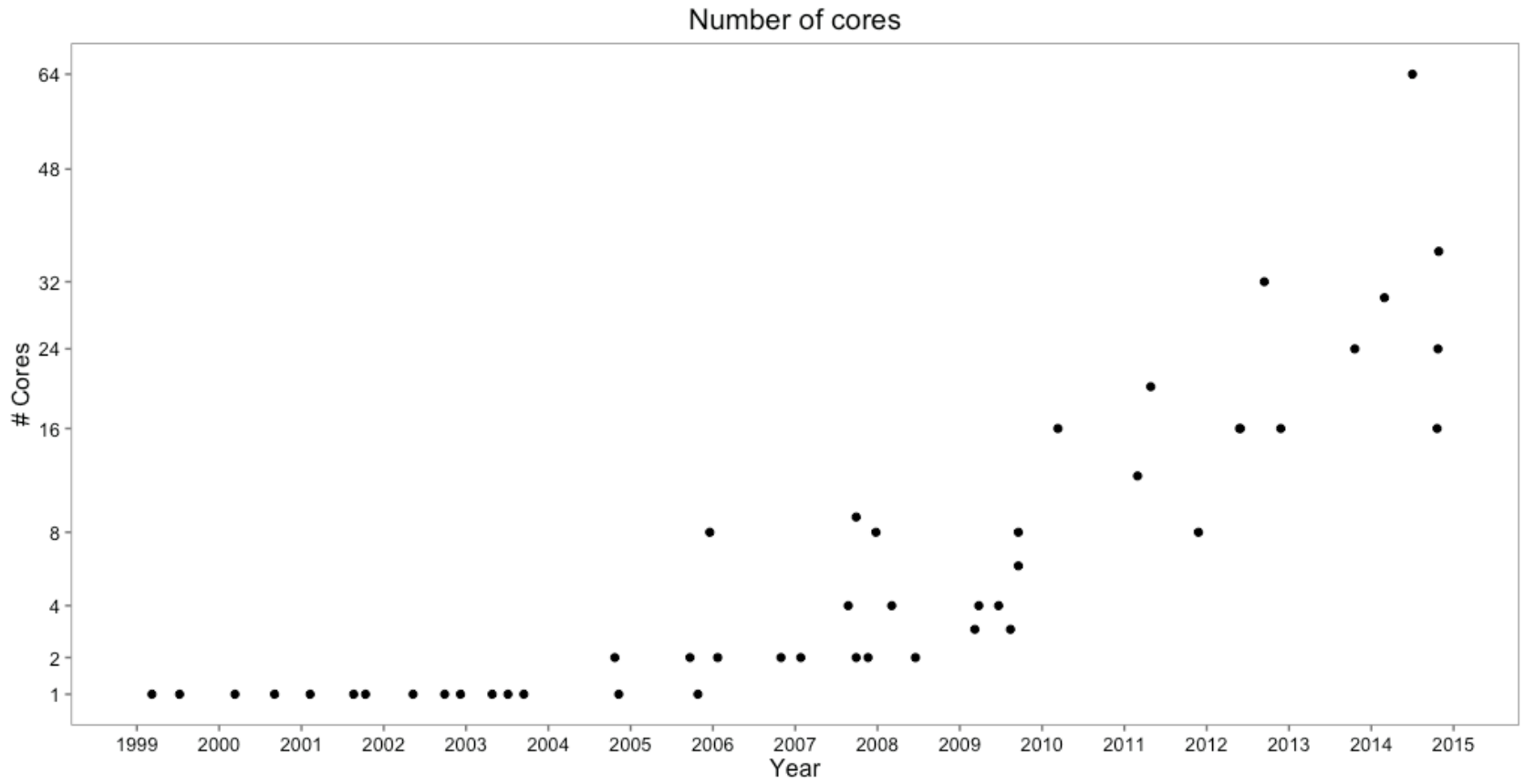
  这是并发 (/2017/go-concurrency-visualize/pingpong36.html)

为什么要关注并发？当今是多核的时代，并发的世界

当今不再是单核的时代



而是多核的趋势


Number of cores
Cores
64
48
32
24
16
8
4
2
1
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
Year

并发编程并不容易，但 Go 对并发有很好的支持

Go 语言中的并发

- goroutine - 并发执行
- channel - 同步和消息传输
- select - 多路并发控制

Goroutine

- 类似于 UNIX 中的 &
- 很像线程，但更轻量
- 一个 goroutine 就是一个独立运行的函数，和其他 goroutine 在同一地址空间
- 当一个 goroutine 阻塞时，所在的线程会阻塞，但其它 goroutine 不受影响

```
f("hello", "world") // f runs; we wait
```

```
go f("hello", "world") // f starts running
g() // does not wait for f to return
```

Channel

- 类似于 UNIX 中的 管道
- 允许在 goroutines 之间传递消息

```
timerChan := make(chan time.Time)
go func() {
    time.Sleep(deltaT)
    timerChan <- time.Now() // send time on timerChan
}()
// Do something else; when ready, receive.
// Receive will block until timerChan delivers.
// Value sent is other goroutine's completion time.
completedAt := <-timerChan
```

Select

- 类似于 switch
- 但它的判断条件是基于通信，而不是基于值的等量匹配

```
select {
case v := <-ch1:
    fmt.Println("channel 1 sends", v)
case v := <-ch2:
    fmt.Println("channel 2 sends", v)
default: // optional
    fmt.Println("neither channel was ready")
}
```

Go 让并发编程变的简单起来

但是问题来了

- 我们怎样去讲解 Go 的并发?
- 我们怎样思考 Go 的并发过程?
- 最终，我们怎样更好的实践 Go 并发编程?

祭出法宝 - GoTrace

一种将 Go 并发过程可视化的开源(https://github.com/divan/gotrace)工具

出自 divan(https://github.com/divan)大神，主要包含两个程序：

- gotrace(go)：分析 go tool trace 的执行结果
- gothree(js): 基于 ThreeJs 和 WebGL 生成 3D 图像

感谢 divan(https://github.com/divan)大神 提供了这款工具和不少 Go 并发模式的素材

说了这么多，耳听为虚，眼见为实

1. HELLO, WORLD!

```
package main

func main() {
  // 创建一个 int 类型的 channel
  ch := make(chan int)

  // 启动一个新的匿名 goroutine
  go func() {
      // 发送 42 给 channel
      ch <- 42
  }()
  // 从 channel 读取
  <-ch
}
```

Go并发可视化(/2017/go-concurrency-visualize/helloworld.html)

2. 计时器

```
func tick(d time.Duration) <-chan int {
    c := make(chan int)
    go func() {
        time.Sleep(d)
        c <- 1
    }()
    return c
}

func main() {
    trace.Start(os.Stderr)
    for i := 0; i < 24; i++ {
        c := tick(100 * time.Millisecond)
        <-c
    }
    trace.Stop()
}
```

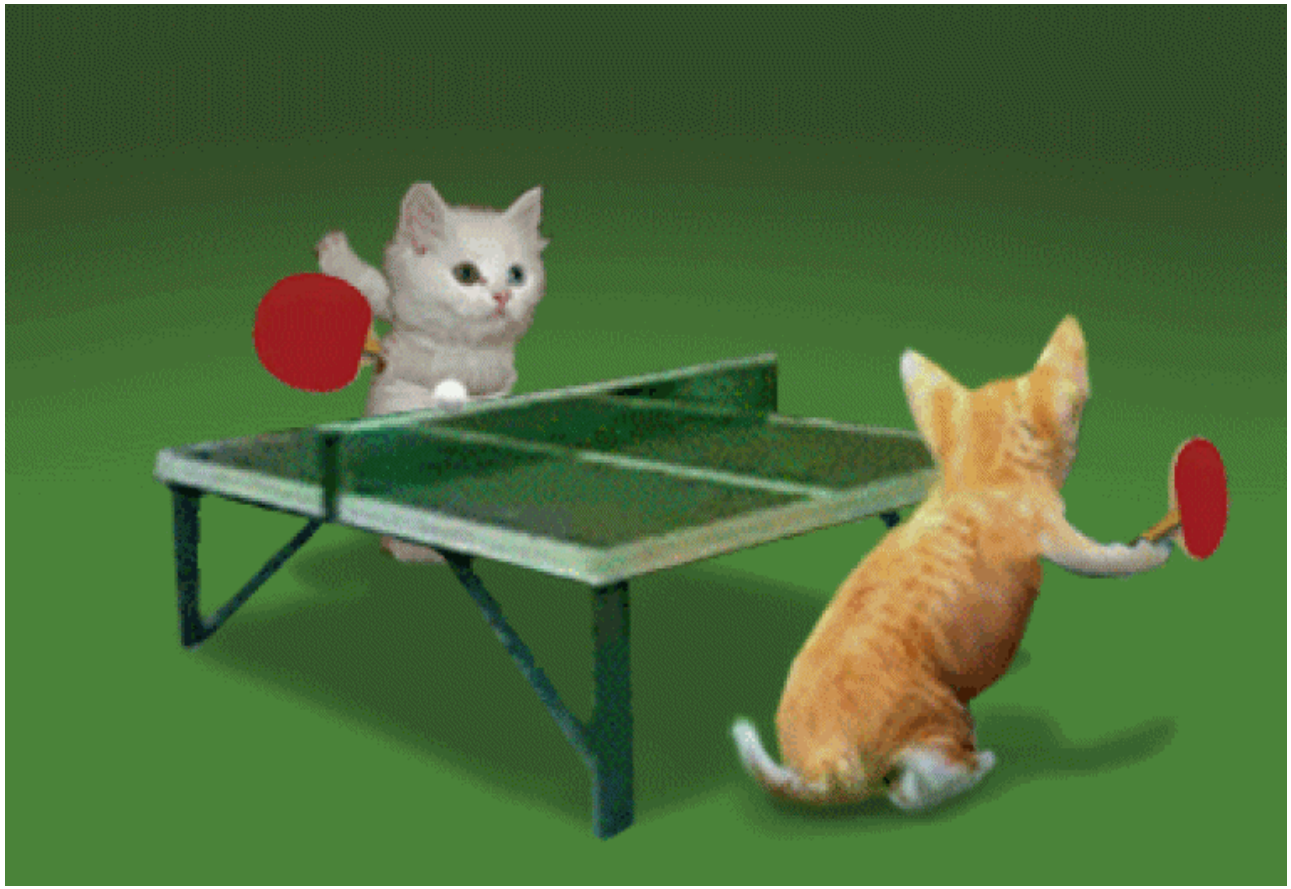
Go并发可视化(/2017/go-concurrency-visualize/timer.html)

3. 乒乓球 - 2 个玩家

```
func main() {
    var Ball int
    table := make(chan int)

    // 2个玩家
    go player(table)
    go player(table)

    table <- Ball
    time.Sleep(1 * time.Second)
    <-table
}

func player(table chan int) {
    for {
        ball := <-table
        ball++
        time.Sleep(100 * time.Millisecond)
        table <- ball
    }
}
```

- 参考 Sameer Ajmani (http://twitter.com/Sajma)(Google)的分享 “Advanced Go Concurrency Patterns” (https://talks.golang.org/2013/advconc.slide#43)

3. 乒乓球 - 2 个玩家



Go并发可视化(/2017/go-concurrency-visualize/pingpong2.html)

3. 乒乓球 - 3 个玩家

```
func main() {
    var Ball int
    table := make(chan int)

    // 3个玩家
    go player(table)
    go player(table)
    go player(table)

    table <- Ball
    time.Sleep(1 * time.Second)
    <-table
}
```

Go并发可视化(/2017/go-concurrency-visualize/pingpong3.html)

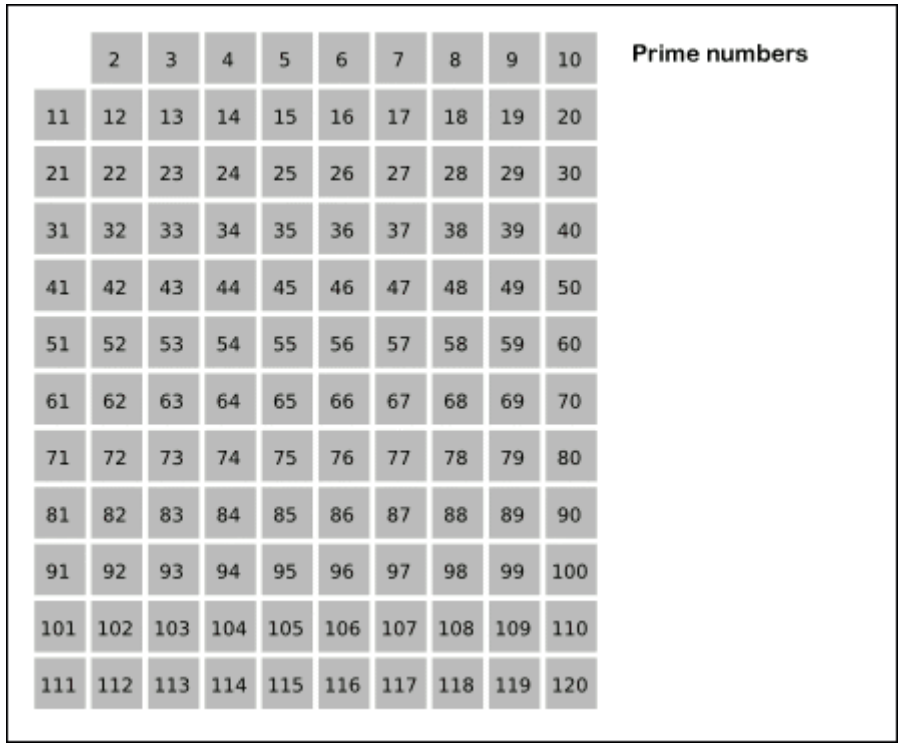
3. 乒乓球 - 36 个玩家

```
func main() {
    var Ball int
    table := make(chan int)

    // 36个玩家
    for i := 0; i < 36; i++ {
            go player(table)
    }

    table <- Ball
    time.Sleep(1 * time.Second)
    <-table
}
```

Go并发可视化(/2017/go-concurrency-visualize/pingpong36.html)

4. 素数筛-算法

感谢白明(http://tonybai.com/)老师的动图(http://tonybai.com/wp-content/uploads/Sieve_of_Eratosthenes_animation.gif)



	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

要找到小于或等于给定整数n的素数，可以使用Eratosthenes' sieve(埃拉托斯特尼)算法(https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes)。

> 算法核心思想：先用最小的素数2去筛，把2的倍数剔除掉；下一个未筛除的数就是素数(这里是3)。
> 再用这个素数3去筛，筛除掉3的倍数... 这样不断重复下去，直到筛完为止。

4. 素数筛-实现

```
func generate(ch chan<- int) {
    for i := 2; ; i++ {
        ch <- i // Send 'i' to channel 'ch'.
    }
}
func filter(src <-chan int, dst chan<- int, prime int) {
    for i := range src { // Loop over values received from 'src'.
        if i%prime != 0 {
            dst <- i // Send 'i' to channel 'dst'.
        }
    }
}
func main() {
    ch := make(chan int)
    go generate(ch)
    for i := 0; i < 10; i++ {
        prime := <-ch
        fmt.Println(prime)
        out := make(chan int)
        go filter(ch, out, prime)
        ch = out
    }
}
```

4. 素数筛-可视化

Go并发可视化(/2017/go-concurrency-visualize/primesieve.html)

一图胜千言

5. 其它-Goroutines 泄露

```
func leaker() {
    time.Sleep(1000000 * time.Minute)
}

func main() {
    trace.Start(os.Stderr)
    for i := 0; i < 1000; i++ {
        go leaker()
    }
    trace.Stop()
}
```

Go并发可视化(/2017/go-concurrency-visualize/leak.html)

GoTrace 用法简介

- go get -v -u github.com/divan/gotrace
- 默认使用 go1.6，推荐切换到 go18 分支，支持 go1.8.1

```
Usage: gotrace [trace.out] or [main.go]
       (if you pass .go file to gotrace, it will modify code on the fly,
       adding tracing, run it and collect the trace automagically)
```

- 直接运行 go 代码的效果并不好，推荐生成 trace，需加上如下代码：

```
trace.Start(os.Stderr)
trace.Stop()
```

- 结合 docker，可以用以下脚本：

```
docker run --rm -it -e GOOS=darwin -v $(pwd):/src \
  hub.c.163.com/bingohuang/gotrace:go1.8.1 \
  go build -o /src/binary /src/main.go
./binary 2> ./trace.out
gotrace ./trace.out
```

- 会自动打开浏览器，你可调整视角、缩放、旋转以及加粗线条来改变图像

场景

- 非常酷!
- 学习 Go 的并发模式
- 探究 Go 的并发过程

Thank you

2017.8.5

黄庆兵 - 网易

bingohuang.com

https://c.163yun.com (https://c.163yun.com)

http://talks.bingohuang.com (http://talks.bingohuang.com)