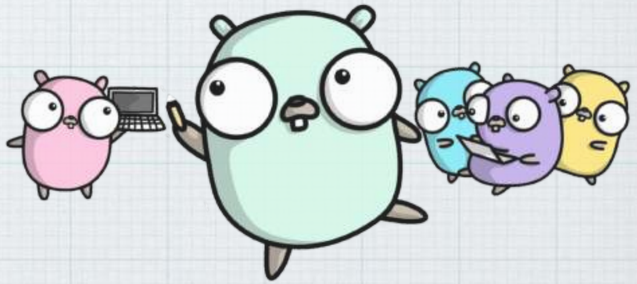
Golang 高级技巧

v1

xiaorui.cc

github.com/rfyiamcool



menu

* 基础配置
* 常见的坑
* 并发编程
* 效率库包
* Q&A

编码规范

字段对其，注释对其

```
type User struct{
    Username    string // 用户名
    Email       string // 邮箱
    URI         string // 后缀
    API         string // 地址
    IsOpen      bool   // 开放
    CreateTime  bool   // 创建时间
}
```

```
var (
  isExist      bool
  hasConflict  bool
  canManage    bool
  allowGitHook bool
)
```

Bool使用"判断"语义的前缀

```
type Scheme string

const (
    HTTP  Scheme = "http"
    HTTPS Scheme = "https"
)

const (
    ModeAdd     = iota + 1
    ModeDel
    ModeUpdate
    ModeUpsert
)
```

自定义类型常量，iota从1开始

编码规范

```
import (
  "context"
  "crypto/tls"
  "fmt"
  "net/http"
  "time"

  "github.com/gin-gonic/gin"

  "git.xiaorui.cc/ocean/jellyfish/pkg/log"
  "git.xiaorui.cc/ocean/jellyfish/internal/api"
  "git.xiaorui.cc/ocean/jellyfish/internal/config"
  "git.xiaorui.cc/ocean/jellyfish/internal/middleware"
)
```

按照来源分段import

```
type Option interface {
  // ...
}

func WithCache(c bool) Option {
  // ...
}

func WithLogger(log *zap.Logger) Option {
  // ...
}

// Open creates a connection.
func Open(opts ...Option) (*Connection, error) {
  // ...
}
```

动态参数

编码规范

默认对象

```
const (
  _defaultPort = 8080

  defaultUser = "user"
)

var (
  ErrNotFound = errors.New(“not found”)
)
```

* 注意函数内做好语义拆分

* 不要try-catch那样使用panic

* 代码要减少 if for 嵌套

接口合理性检

```
type Handler struct {
  // ...
}

// 用于触发编译期的接口的合理性检查机制
// 如果Handler没有实现http.Handler,会在编译期报错
var _ http.Handler = (*Handler)(nil)

func (h *Handler) ServeHTTP(
  w http.ResponseWriter,
  r *http.Request,
) {
  // ...
}
```

编码规范

```
var (
    defaultGitlabClient *Client
    once = sync.Once{}
)


// NewGitlabClient create gitlab client
func NewGitlabClient(disfName string) (*Client, error) {
  once.Do(func() {
     defaultGitlabClient, err = newClient(disfName)
  })

  return defaultGitlabClient, err
}
```

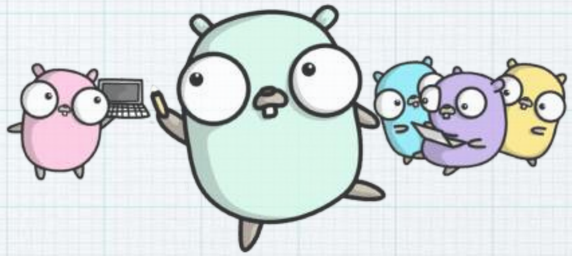
* 安全的单例
* 函数注释

```
type T struct { a int }

// value receiver
func (tv  T) Mv(a int) { tv.a = 123 }

// pointer receiver
func (tp *T) Mp(f int) { tp.a = 123 }
```

* 值接收器
* 指针接收器

Go工程目录

- cmd
- docs
- pkg
- internal
- external
- schema
- model
- api
  - handler
  - router
  - controller
  - service
  - dao
  - ...



这个目录吧

仁者见者，智者见智

😅

开心就好

test

* Testify
  * assert
  * mock
* GoMock
* Monkey

```go
import (
    "testing"
    "github.com/stretchr/testify/assert"
)

func TestSomething(t *testing.T) {

    // assert equality
    assert.Equal(t, 123, 123, "they should be equal")

    // assert inequality
    assert.NotEqual(t, 123, 456, "they should not be equal")

    // assert for nil (good for errors)
    assert.Nil(t, object)

    // assert for not nil (good when you expect something)
    if assert.NotNil(t, object) {

    // now we know that object isn't nil, we are safe to make
    // further assertions without causing any errors
    assert.Equal(t, "Something", object.Value)
    }
}
```

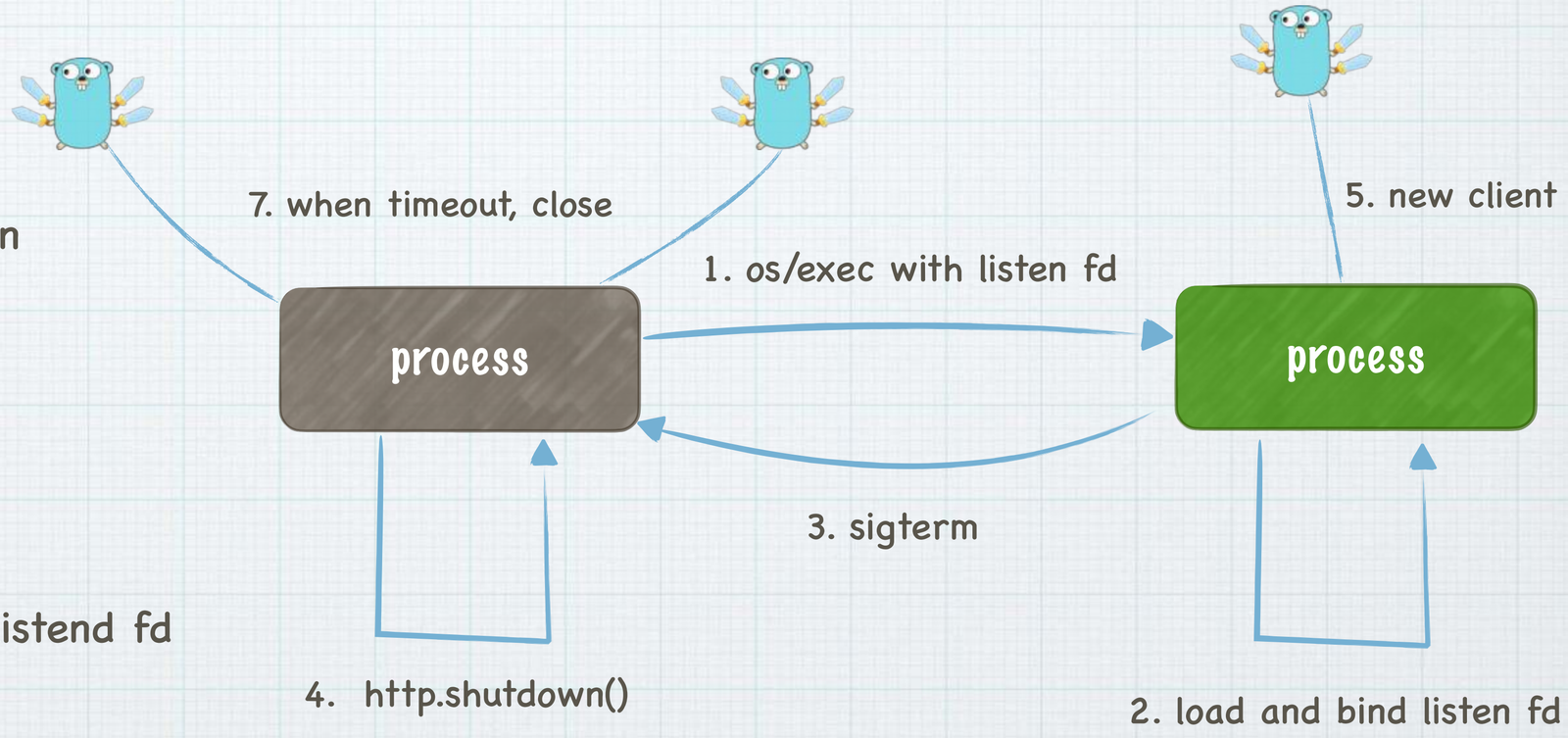
优雅退出

- don't catch sigkill (-9)
- process
  - bind signal
  - listen signal
  - shutdown
    - http
    - grpc
    - other tcp frame
  - exit

```go
// bind signal
signal.Notify(sigch, syscall.SIGHUP, syscall.SIGINT, syscall.SIGTERM, syscall.SIGQUIT)

// listen signal
for sig := range sigch {
    log.Infof("sig recv: %s", sig.String())

    switch sig {
    case syscall.SIGQUIT, syscall.SIGTERM, syscall.SIGINT:
        cancel()
        close(sigch)

    default:
        continue
    }
}

srv.SetKeepAlivesEnabled(false)
if err := srv.Shutdown(ctx); err != nil {
    log.Errorf(err.Error())
}

// gc
serverCall()
libsExitCall()

log.Info("service exited")
os.Exit(int(atomic.LoadInt32(&state)))
```

优雅重启

- so_reuseport
  - 直接开新进程也可进行listen
  - 优雅关闭老进程
- os/exec新进程
  - 启动新二进制文件，并传递listend fd
  - 新进程载入listen fd，开始接受请求
  - 给老进程一个新信号，使其优雅退出



makefile

- 统一执行入口
  - auto code format
  - unit-test
  - go lint
  - build

```
.PHONY: build

BASE_PATH     =  "git.xiaorui.cc/ocean/shark"
NOW           =  $(shell date -u '+%Y%m%d%I%M%S')
GO_LIST       =  $(shell go list ${BASE_PATH}/... | grep -v vendor)
GO_FILES      =  $(shell find . -name '*.go' | grep /pkg/ | grep -v _test.go | grep -v vendor)
zone          ?= "local"

all: fmt build

fmt:
    @gofmt -l -w $(GO_FILES)

build:
    @./scripts/build.sh

init:
    @go mod tidy
    @go mod vendor

test:
    @go test -cover ${GO_LIST}

lint:
    @hash revive 2>&- || go get -u github.com/mgechev/revive
    @revive -config .revive_lint_config -formatter friendly -exclude ./vendor/... cmd master minion pkg
cache

race:
    @go test -cover -race ${GO_LIST}

build-master:
    @./scripts/build.sh master

run-master: build-master
    @./dist/master_release --config=configs/dev.toml

dev-master:
    @go build -mod=vendor -o master_release cmd/master/main.go
    @./master_release --config=configs/dev.toml

gen-proto:
    @protoc -I proto --
gogofaster_out=plugins=grpc,Mgoogle/protobuf/any.proto=github.com/gogo/protobuf/types,:proto
proto/*.proto
```

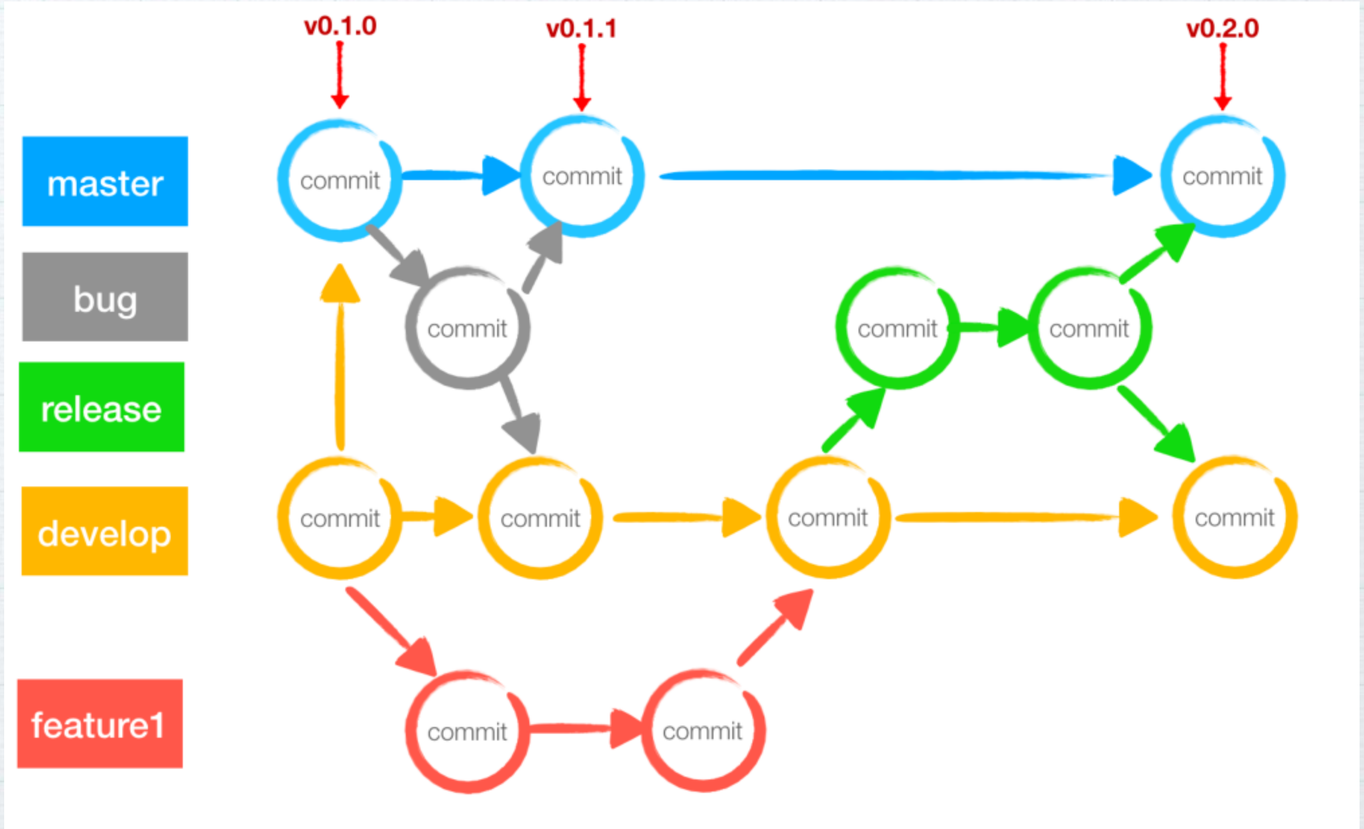
version

```
# set var
version_path=git.hualala.com/ocean/shark/pkg/version
dist=./dist/${project}
git_commit=$(git rev-parse --short HEAD || echo unsupported)
branch=$(git symbolic-ref --short -q HEAD)
go_version=$(go version)
build_date=$(date "+%Y-%m-%d %H:%M:%S")
release_tag=$(git describe --tags $(git rev-list --tags --max-count=1) 2>/dev/null || echo "")
current_version="v0.0.0"

# GOOS=linux GOARCH=amd64 -x
CGO_ENABLED=1 GOOS=linux GOARCH=amd64 go build -mod=vendor -o $dist -ldflags "-X $version_path
.Version=$current_version
                              -X '$version_path.BuildDate=$build_date'
                              -X '$version_path.Branch=$branch'
                              -X '$version_path.GoVersion=$go_version'
                              -X '$version_path.GitCommit=$git_commit'
                              -X $version_path.BuildUser='$(whoami)@$(hostname)" ${main_file}
```

```
// 生成版本信息。在 git 构建时填充。
var (
    Version   string
    GitCommit string
    Branch    string
    BuildUser string
    BuildDate string
    GoVersion string
)

// versionInfoTmpl 模版包含版本相关的使用规范。
var versionInfoTmpl = `
{{.program}}, version {{.version}} (branch: {{.branch}}, gitCommit: {{.gitCommit}})
  build user:       {{.buildUser}}
  build date:       {{.buildDate}}
  go version:       {{.goVersion}}
`

// Print 返回版本的相关信息.
func Print(program string) string {
    m := map[string]string{
        "program":   program,
        "version":   Version,
        "gitCommit": GitCommit,
        "branch":    Branch,
        "buildUser": BuildUser,
        "buildDate": BuildDate,
        "goVersion": GoVersion,
    }
    t := template.Must(template.New("version").Parse(versionInfoTmpl))

    var buf bytes.Buffer
    if err := t.ExecuteTemplate(&buf, "version", m); err != nil {
        panic(err)
    }
    return strings.TrimSpace(buf.String())
}
```

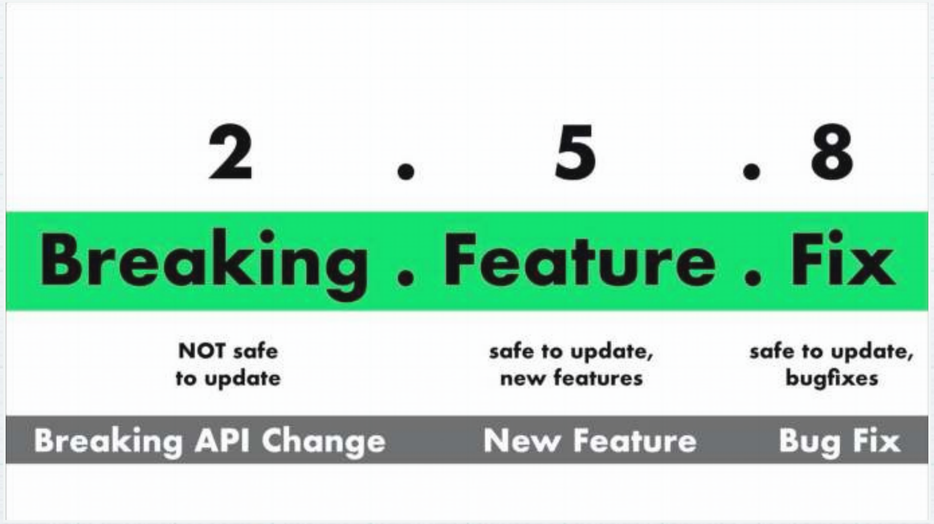
gitflow

* master
    * git merge hotfix
    * git merge release/v1.0.0-rc.0
* hotfix
    * git checkout -b hotfix/fix-timeout master
* release
    * git checkout -b release/v1.0.0-rc.0 develop
* develop
    * ...
* feature
    * git checkout -b feature/add-timeout develop



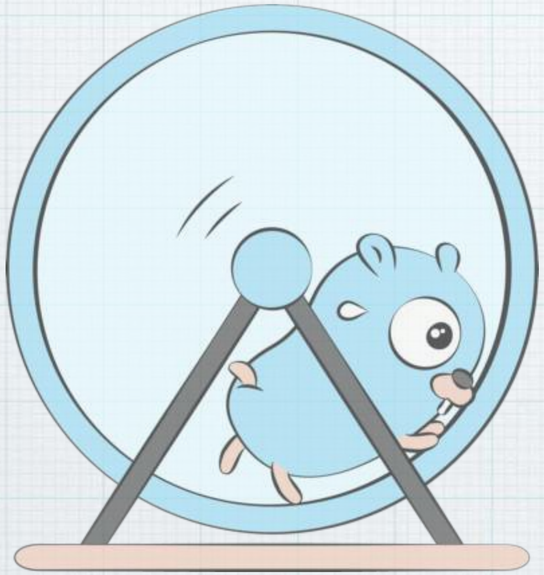
semver

* major 主版本
  * 不兼容的API的调整
  * not safe to update
* minor 次版本
  * new feature
  * safe to update
* patch 修订版
  * bugfix
  * safe to update
* pre-release label
  * alpha 内部版本
  * beta 公测版本
  * rc 正式版本的候选版本



* v1.0.2-alpha.1
* v1.0.2-alpha.2
* v1.0.2-beta.1
* v1.0.2-rc.1
* v1.0.2

常见的坑



* for range go
* net/http
* defer
* go in docker

for range go

- for range 遍历并发
  - 局部对象
  - 值传递给函数

```go
package main

import (
    "fmt"
    "sync"
)

func main() {
    wg := sync.WaitGroup{}
    for _, gid := range []string{"1", "2", "3"} { // gid只会实例化一次，后面迭代就是把值复制到gid的地址上。
        // gid := gid

        // go func(gid string) {
        // defer wg.Done()
        // fmt.Println("string: ", gid)
        // fmt.Printf("%p \n", &gid)
        // }(gid)

        wg.Add(1)
        go func() {
            defer wg.Done()
            fmt.Println("string: ", gid)
            fmt.Printf("%p \n", &gid)
        }()
    }
    wg.Wait()
}
```

net/http

- 修改默认的连接数
- 未读body则连接无法复用
- 未关闭io对象则协程泄露

```go
package main

import (
    "net/http"
    "time"
    "fmt"
)

func main() {
    http.DefaultTransport.(*http.Transport).MaxIdleConnsPerHost = 100 // std default = 2
    http.DefaultTransport.(*http.Transport).MaxIdleConns = 500        // std default = 100

    count := 100
    for i := 0; i < count; i++ {
        resp, err := http.Get("http://www.xiaorui.cc")
        if err != nil {
            panic(err)
        }
        time.Sleep(1 * time.Second)

        // io.Copy(ioutil.Discard, resp.Body) 连接无法复用，主动关闭

        // resp.Body.Close() loop协程泄露
    }
}
```

defer

- retrun with defer
  - 设置返回值
  - call defer list
  - ret
- 匿名返回值
- 有名返回值
  - 提前定义

```
func main() {
    println(DeferFunc1(1))
    println(DeferFunc2(1))
    println(DeferFunc3(1))
}
```

```
// 4
func DeferFunc1(i int) (t int) {
    t = i
    defer func() {
        t += 3
    }()
    return t
}

// 1
func DeferFunc2(i int) int {
    t := i
    defer func() {
        t += 3
    }()
    return t
}

// 3
func DeferFunc3(i int) (t int) {
    defer func() {
        t += i
    }()
    return 2
}
```

COW

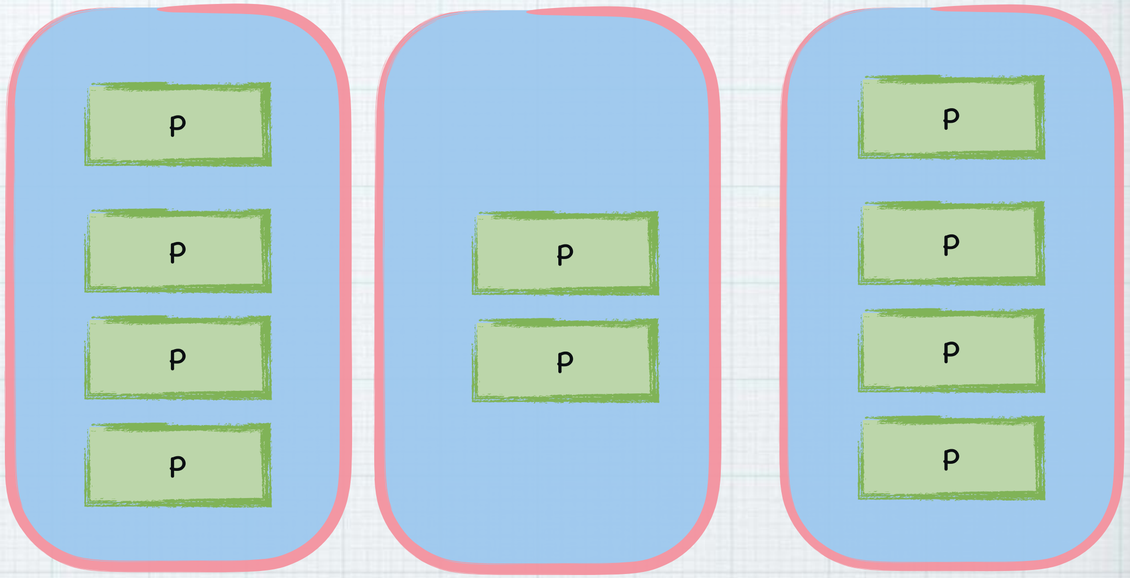
```
CMPL    runtime.writeBarrier(SB), $0
JNE     911
MOVQ    "".&m+64(SP), AX
MOVUPS  (AX), X0
MOVUPS  X0, "".dict(SB)
MOVUPS  16(AX), X0
MOVUPS  X0, "".dict+16(SB)
MOVUPS  32(AX), X0
MOVUPS  X0, "".dict+32(SB)
MOVUPS  48(AX), X0
MOVUPS  X0, “".dict+48(SB)
. . .
…
```

```
fatal error: sync: unlock of unlocked mutex
goroutine 866557 [running]:
runtime.throw(0xc0236c, 0x1e)
    /opt/go1.10.3.linux-amd64/src/runtime/panic.go:616 +0x81 fp=0xc44a9d65c0 sp=0xc44a9d65a0
pc=0x42b191
sync.throw(0xc0236c, 0x1e)
    /opt/go1.10.3.linux-amd64/src/runtime/panic.go:605 +0x35 fp=0xc44a9d65e0 sp=0xc44a9d65c0
pc=0x42b105
sync.(*Mutex).Unlock(0xc42522c050)
    /opt/go1.10.3.linux-amd64/src/sync/mutex.go:184 +0xc2 fp=0xc44a9d6608 sp=0xc44a9d65e0 pc=0x46c652
sync.(*Map).Store(0xc42522c050, 0xabd0c0, 0xc4374e15a0, 0xbabe00, 0xc4305738c0)
    /opt/go1.10.3.linux-amd64/src/sync/map.go:162
```

```
var (
    dict    = sync.Map{}
    dictPtr = sync.Map{}
)

func update() {
   var m = sync.Map{}
   for k, v := range fetchData() {
       m.Store(k, v)
   }
   dict = m
}

func update() {
   var m = sync.Map{}
   for k, v := range fetchData() {
       m.Store(k, v)
   }
   dict = &m
}
```

go in docker

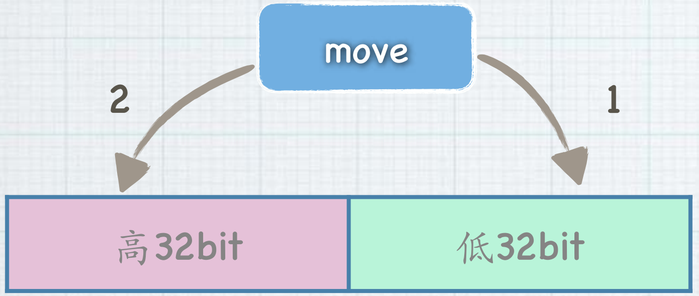
- golang默认P的数量为取cpu core
- P数的增多有何问题？
  - 上下文切换开销
- 根据docker的cpu-quota来动态配置P



http://github.com/uber-go/automaxprocs

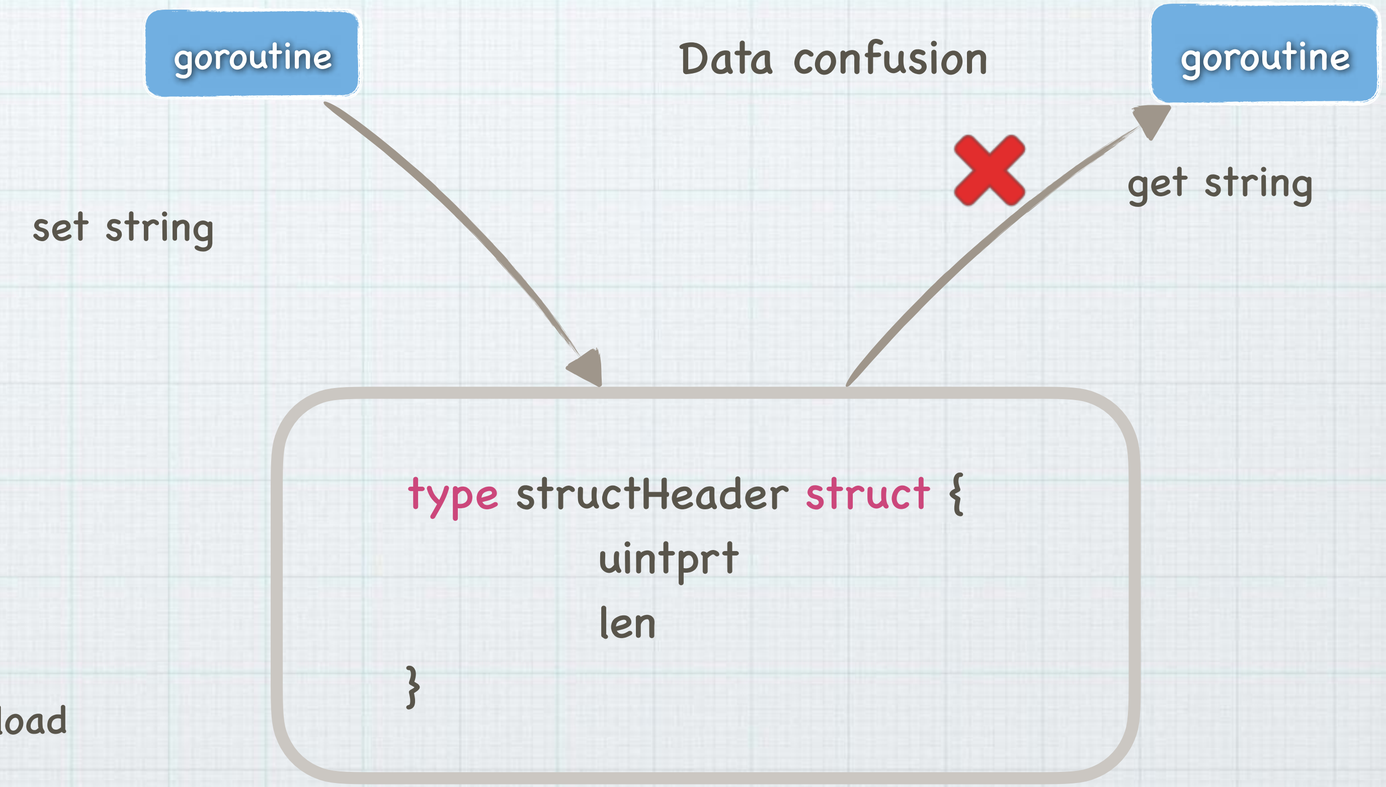
```
root@a4f33fdd0240:/# cat /proc/cpuinfo| grep "processor"| wc -l
64
```

data race

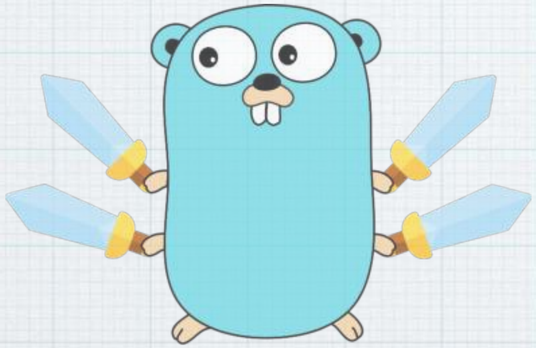


* data race
  * int64 in 32bit system
  * string

* solution
  * copy and set
  * mutex
  * atomic store & load



并发编程



* 同步原语
* 并发模型
* atomic && cpu
* sync包
* 经验

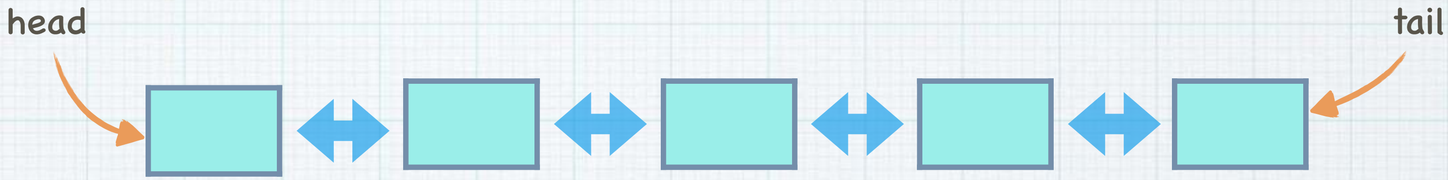
同步原语

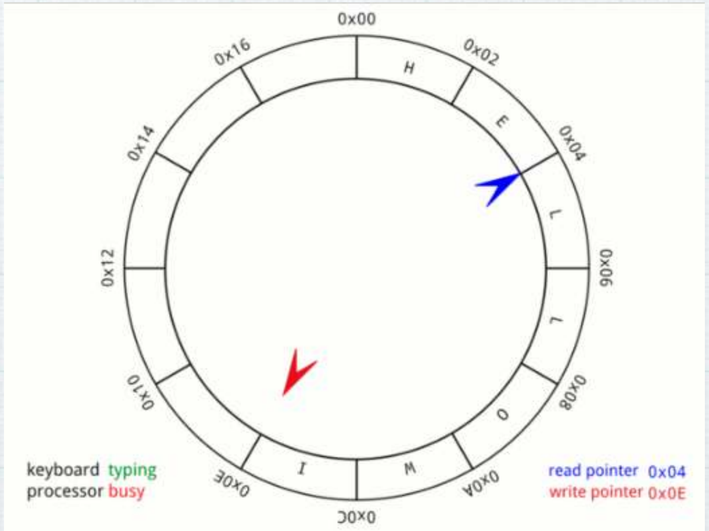
* Sync
  * mutex
  * rwmutex
  * once
  * cond
  * atomic

* Extend
  * ReentrantLock
  * Semaphore
  * SpinLock
  * Fslock
  * SingleFlight
  * NetLocker

通信

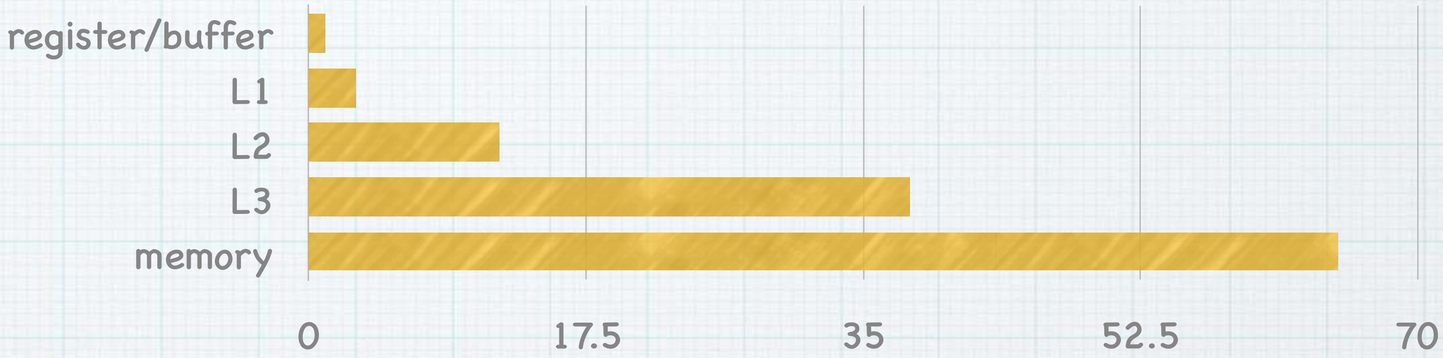
- channel
  - 缓冲
  - 无缓冲
- deque
  - 两端都可处理
  - 随意扩展
- ringbuffer
  - 有界实现
  - 无界实现
- disruptor
  - lock free

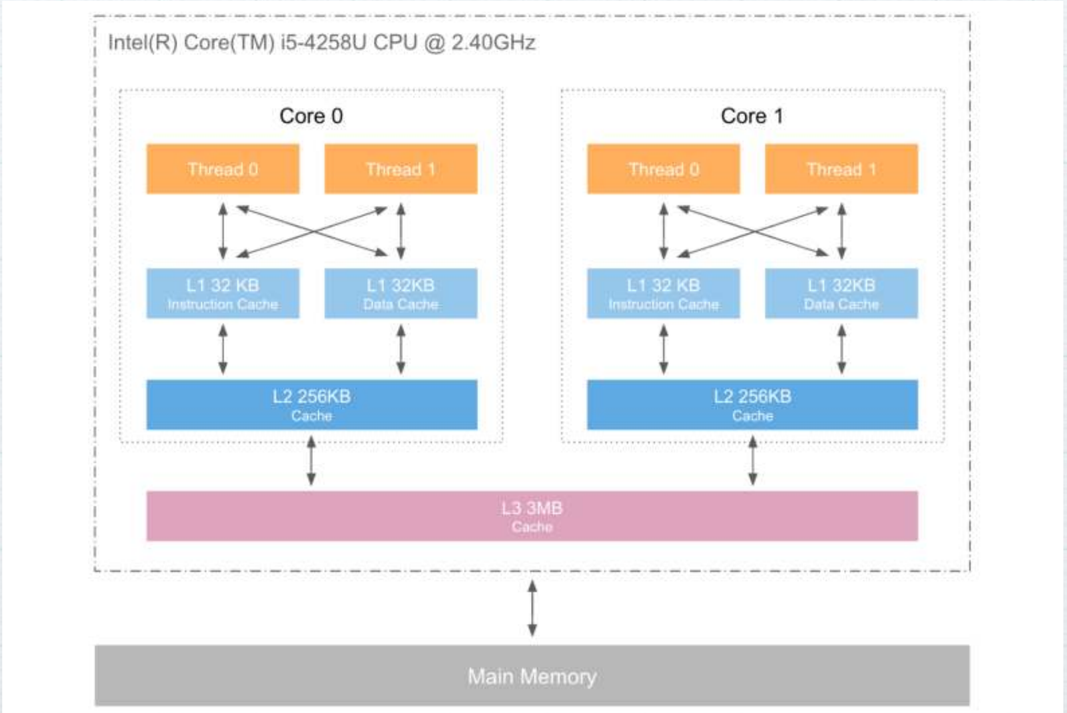




cpu cache

- read cache
  - cache hit
    - 从近到源读取
    - buf -> l1 -> l2 -> l3 -> mem
- cache miss
  - 再从远到近逐步载入
  - main -> l3 -> l2 -> l1
- wirte
  - write through
  - write back (主流)
- MESI 一致性协议
- cacheline = 64 bytes





atomic

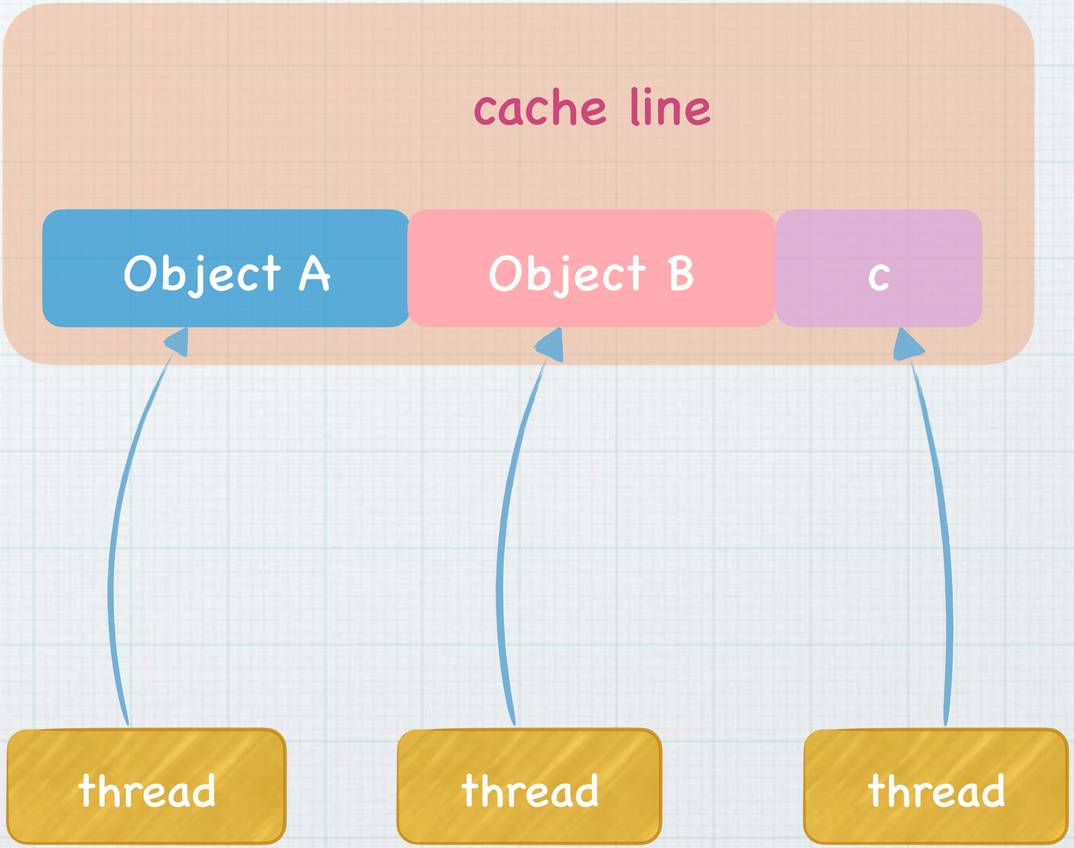
* add
* cas
* store
* load

指令 { CMPXCHG }

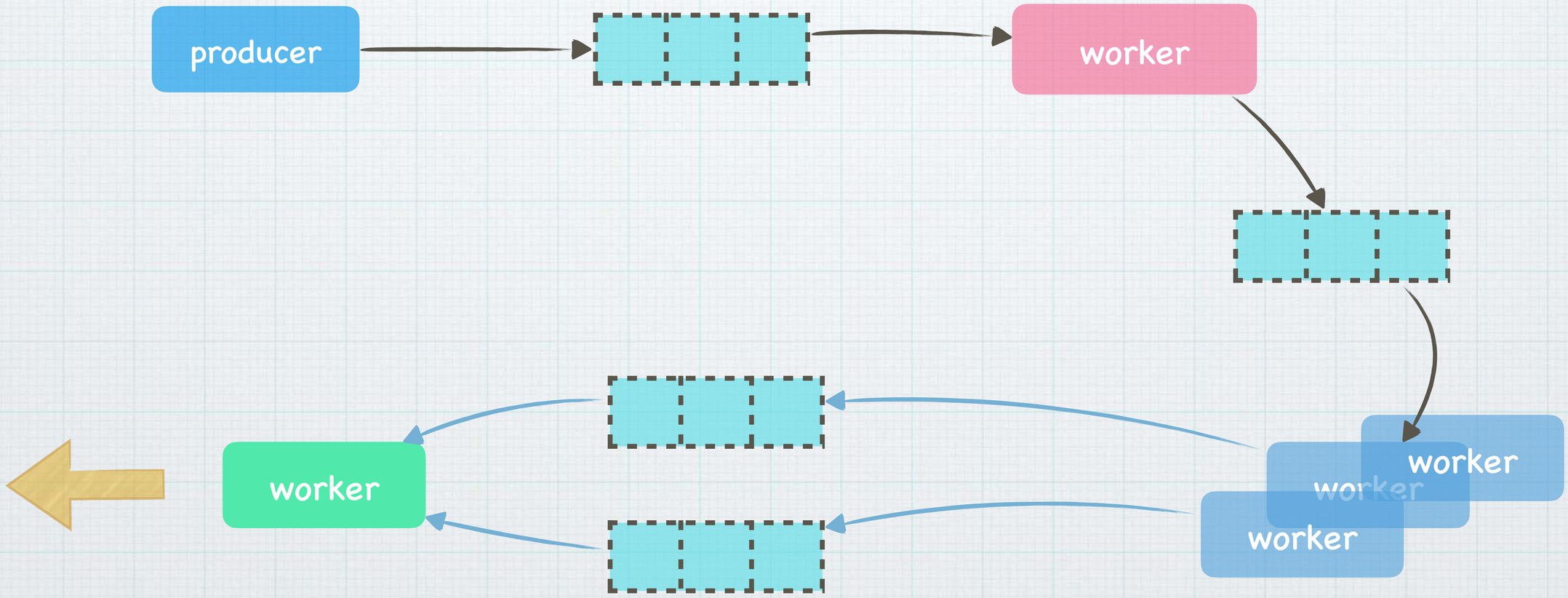
* 场景
  * 并发 a = a + 1
  * 轻量级，减少futex锁竞争
  * spinlock
    * 适合快锁逻辑
    * cas 四次尝试 + cpu pause
    * runtime.gosched or osyield
    * sleep
  * 可见性，规避脏读
  * ...

false sharing

```
type NoPadding struct {
    a uint64
    b uint64
}

type WithPadding struct {
    a uint64
    _ [56]byte
    b uint64
    _ [56]byte
}
```

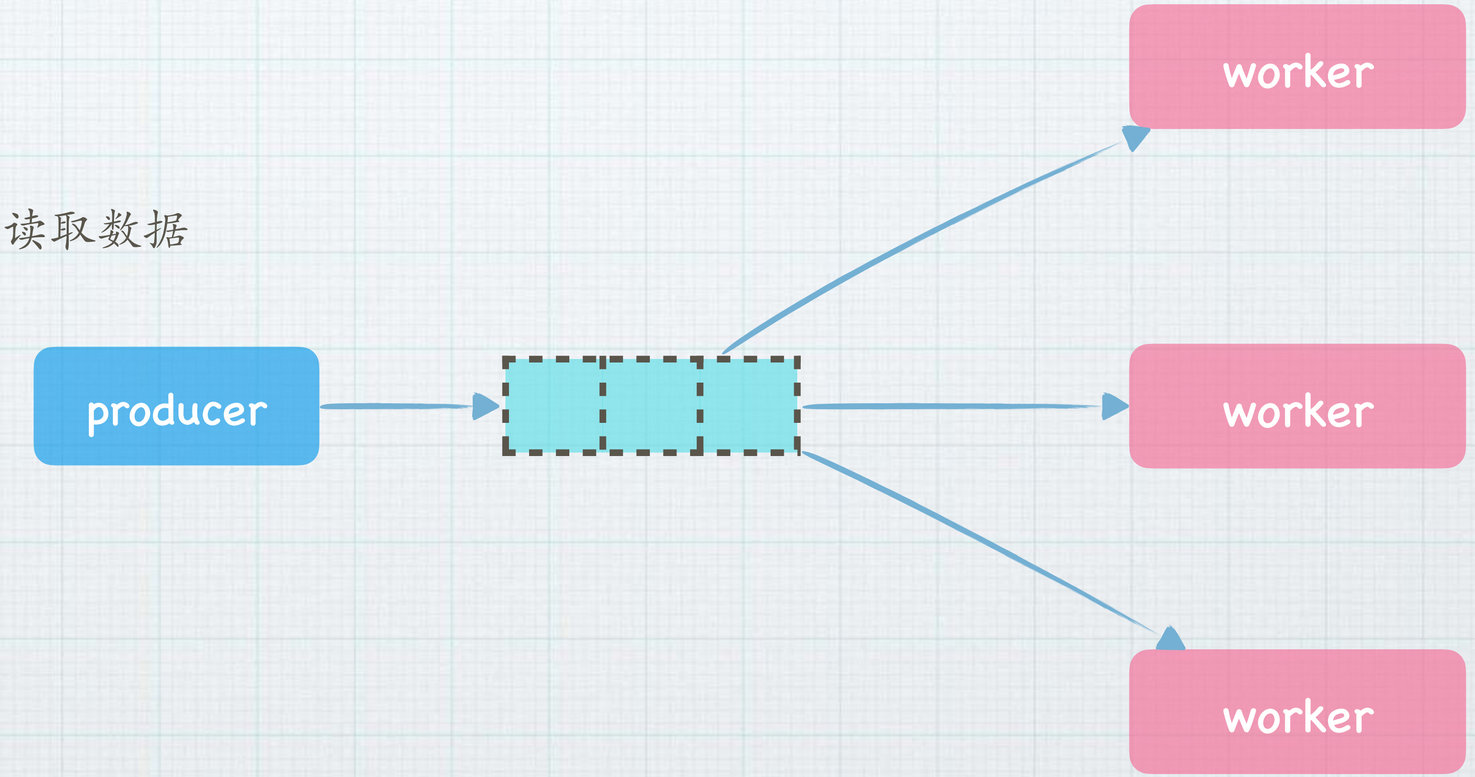


pipeline

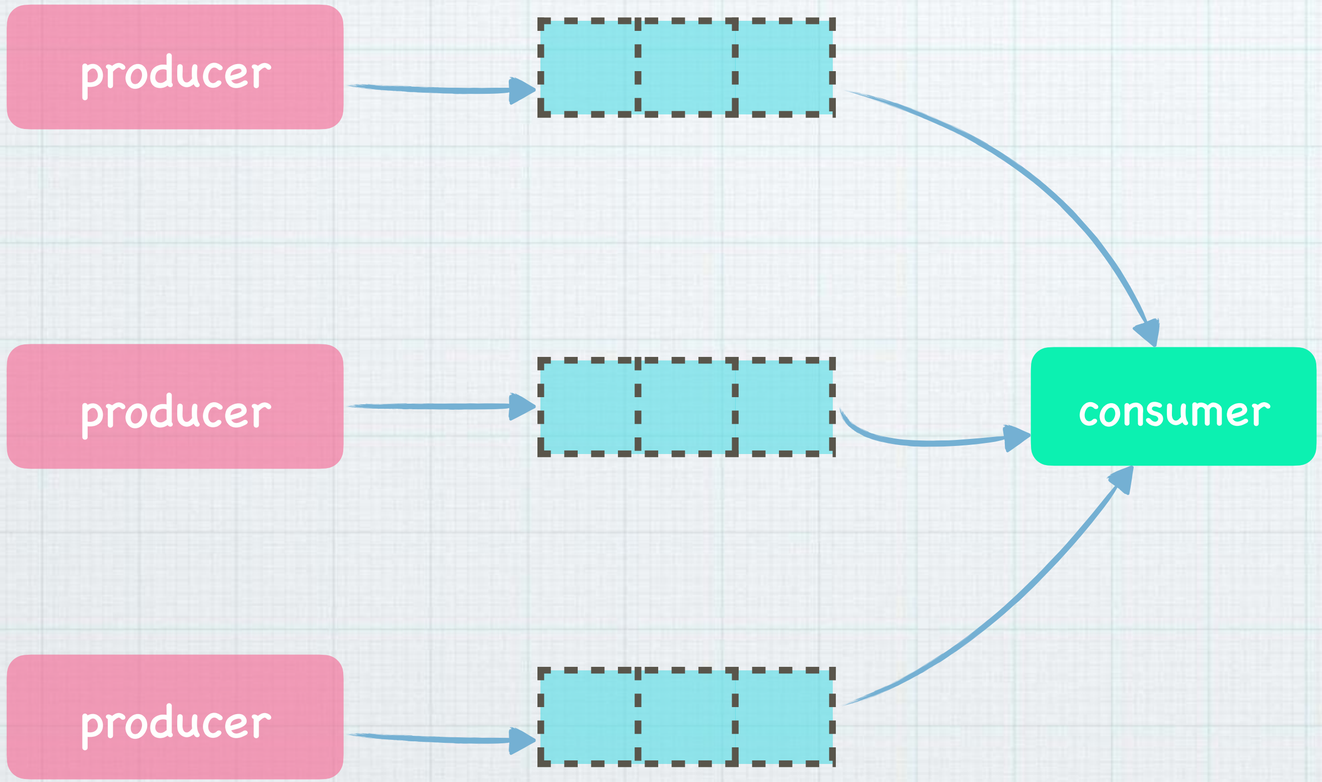


fan-out

* FAN-OUT模式
  * 多个goroutine从同一个通道读取数据
  * 扇出模式
  * 分发任务



fan-in



* FAN-IN模式
  * 1个goroutine从多个通道读取数据
  * 扇入

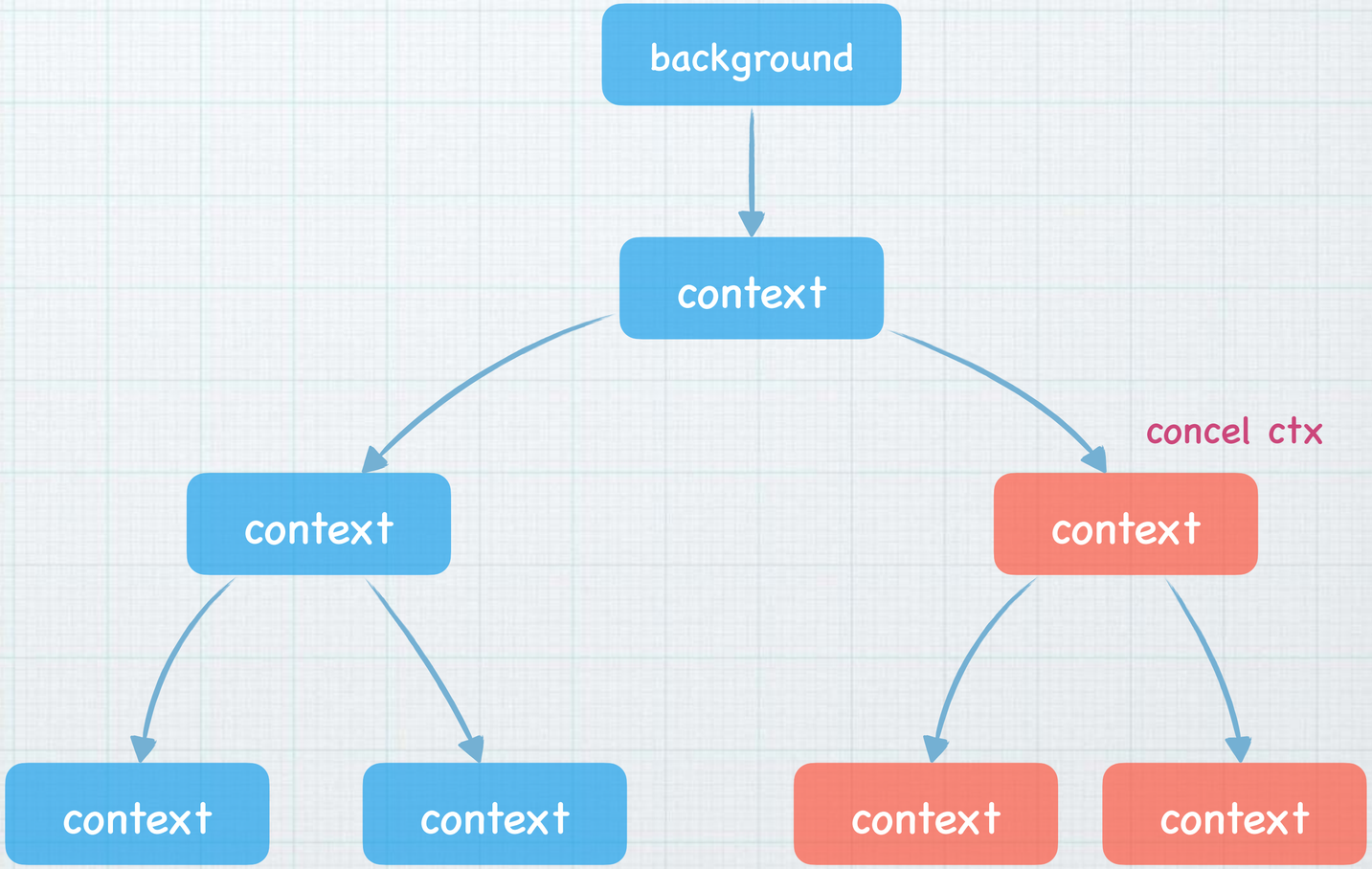
data race

```go
package main

import (
    "fmt"
    "time"
)

func main() {
    a := 1
    go func() {
        a = 2
    }()
    fmt.Println("a is ", a)

    time.Sleep(1 * time.Second)
}
```

```
a is  1
==================
WARNING: DATA RACE
Write at 0x00c00001c0b8 by goroutine 7:
  main.main.func1()
      /Users/ruifengyun/go/src/github.com/rfyiamcool/test/data_race.go:11 +0x38

Previous read at 0x00c00001c0b8 by main goroutine:
  main.main()
      /Users/ruifengyun/go/src/github.com/rfyiamcool/test/data_race.go:13 +0x88

Goroutine 7 (running) created at:
  main.main()
      /Users/ruifengyun/go/src/github.com/rfyiamcool/test/data_race.go:10 +0x7a
==================
Found 1 data race(s)
exit status 66
```

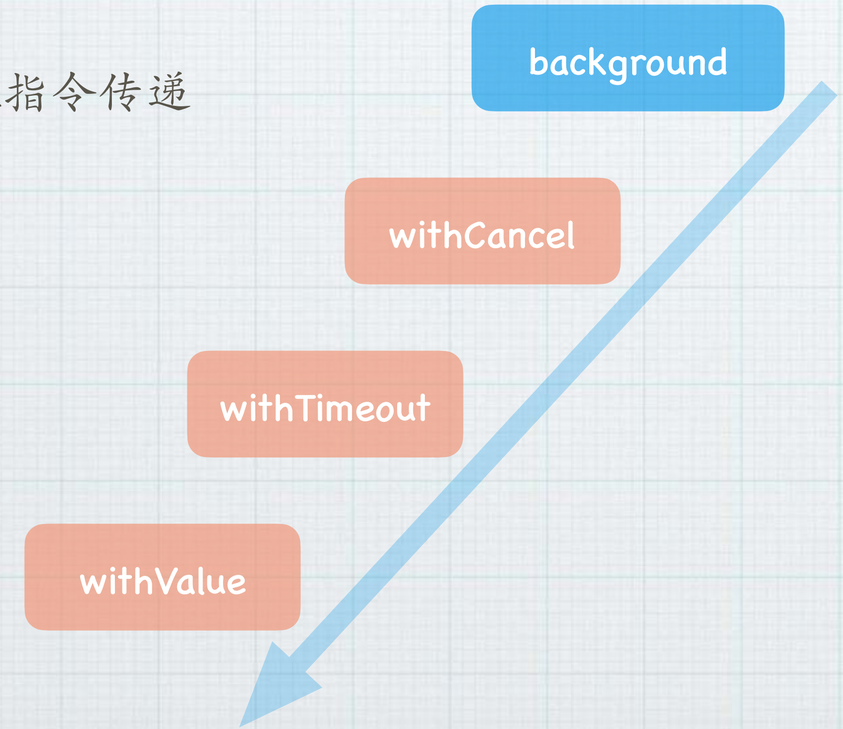
context

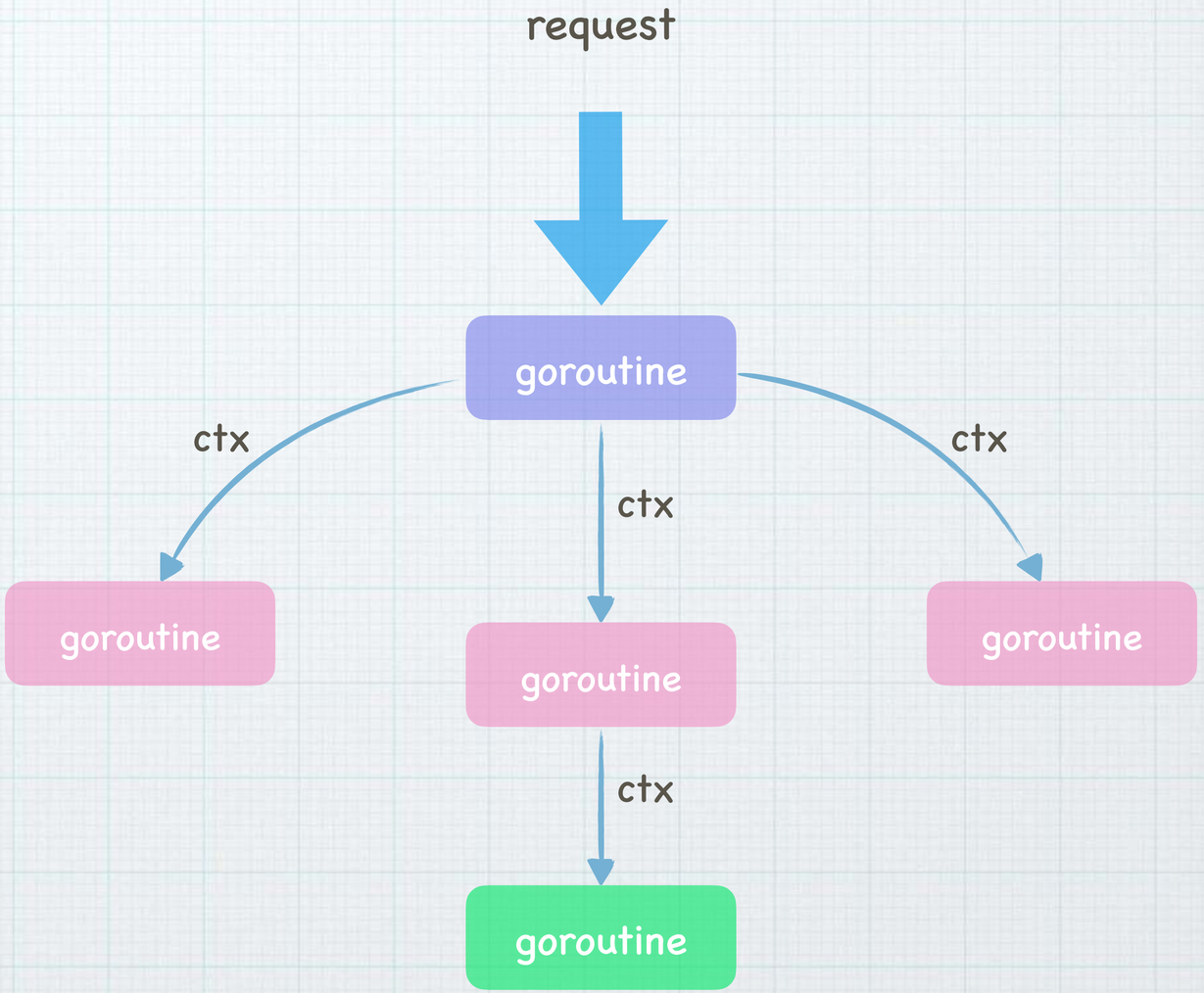
* context.Background()
  * context.WithCancel()
  * context.WithTimeout()
  * context.WithDeadline()
  * context.WithValue()



context

* ctx 并发安全
* 函数之间数据及信号传递
* 协程指令传递





defer

* 1.14 之前
  * defer 单次调用 ≈ 40ns
* 1.14 之后
  * ssa编译期间可优化defer

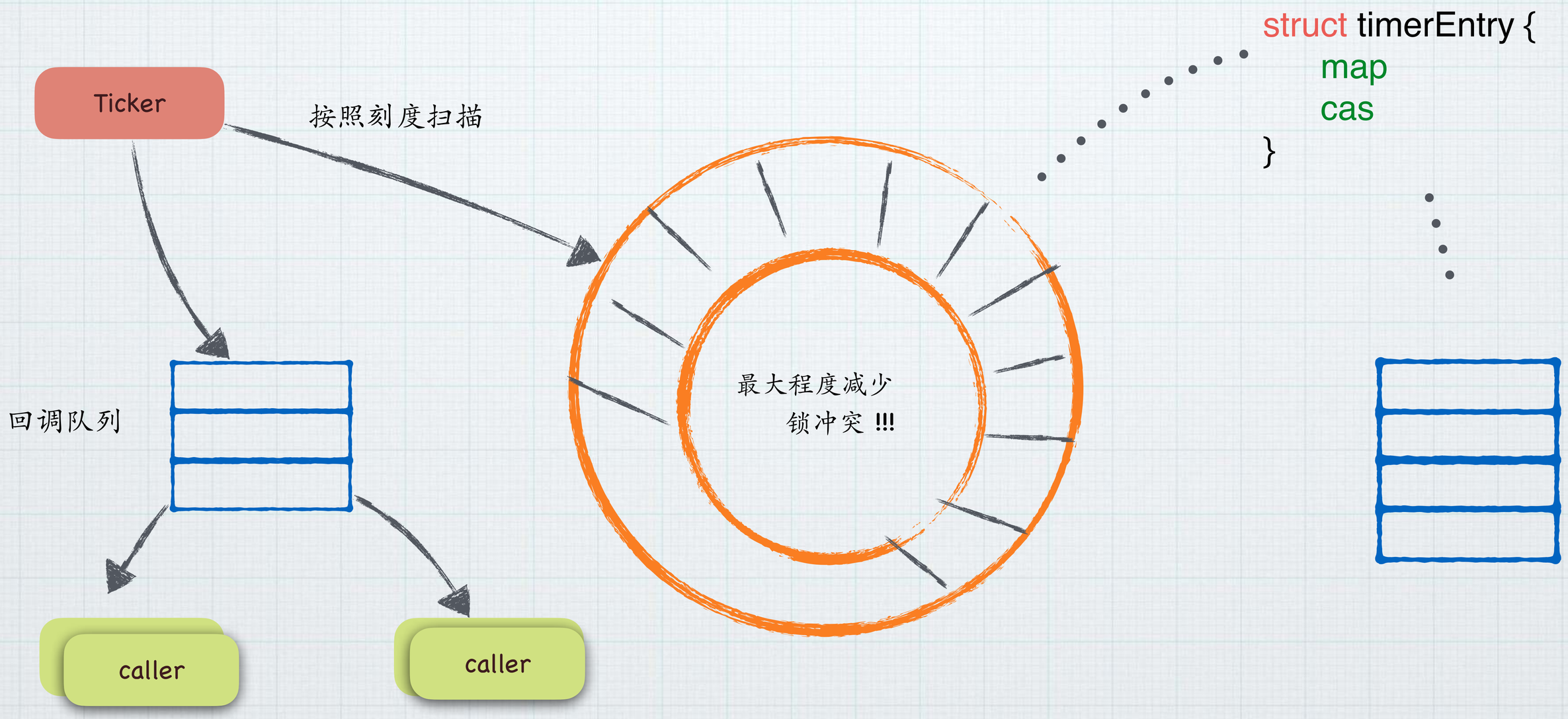
```
var (
    lock sync.Mutex
    incr int
)

func add1() {
    lock.Lock()
    defer lock.Unlock()

    incr ++
}

func add2() {
    lock.Lock()
    incr ++
    lock.Unlock()
}
```

timer

- 升级go v1.10.3以上
  - runtime改进为p个timer定时器
- 升级到go v1.14.x
  - 定时四叉堆收敛到p里面
  - 在findrunnable()时可触发
- 实现自定义时间轮
  - 锁分散到每个槽位
  - 使用map存储定时任务
  - 损失精度来减少锁竞争

- 标准库timer
  - 严重的锁竞争
  - 业务上允许低时间精度

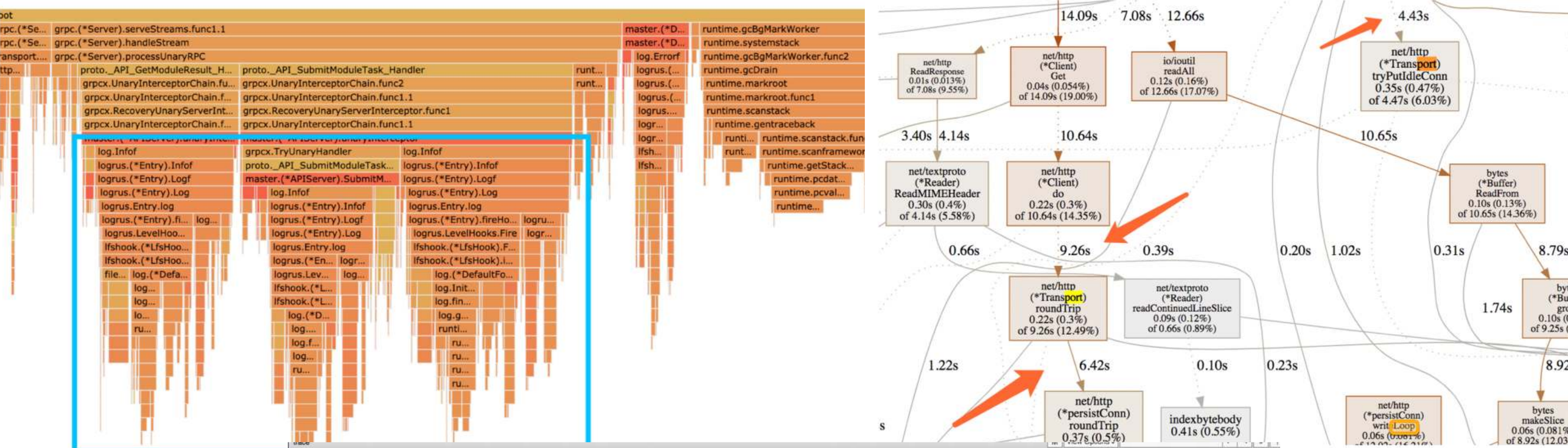
timerWheel


Ticker
按照刻度扫描
回调队列
caller
caller
最大程度减少
锁冲突 !!!
struct timerEntry {
map
cas
}

优化工具

- pprof cpu on
- pprof cpu off
  - use fgprof
- pprof heap
- pprof trace

- 为什么cpu开销大？
- 为什么这么多协程？
- 协程都在干嘛？
- 排查死锁？
- 内存泄露点？
- 内存优化点？gc 时延优化？
- cpu-off
  - 哪个方法慢？
  - 时延高的函数


grpc.(*Server).serveStreams.func1.1
grpc.(*Server).handleStream
grpc.(*Server).processUnaryRPC
proto._API_GetModuleResult_H...
proto._API_SubmitModuleTask_Handler
grpcx.UnaryInterceptorChain.fu...
grpcx.UnaryInterceptorChain.func2
grpcx.UnaryInterceptorChain.f...
grpcx.UnaryInterceptorChain.func1.1
grpcx.RecoveryUnaryServerInt...
grpcx.RecoveryUnaryServerInterceptor.func1
grpcx.UnaryInterceptorChain.f...
grpcx.UnaryInterceptorChain.func1.1
log.Infof
logrus.(*Entry).Infof
logrus.(*Entry).Logf
logrus.(*Entry).Log
logrus.Entry.log
grpcx.TryUnaryHandler
proto._API_SubmitModuleTask...
master.(*APIServer).SubmitM...
logrus.(*Entry).fireHo...
logrus.LevelHooks.Fire
lfshook.(*LfsHook).F...
lfshook.(*LfsHook).i...
log.(*DefaultFo...
log.Init...
log.fin...
log.g...
runti...
master.(*D...
log.Errorf
logrus.(...
logrus.(...
logrus.(...
lfsh...
lfsh...
runtime.gcBgMarkWorker
runtime.systemstack
runtime.gcBgMarkWorker.func2
runtime.gcDrain
runtime.markroot
runtime.markroot.func1
runtime.scanstack
runtime.gentraceback
runtime.scanstack.fun
runtime.scanframewor
runtime.getStack...
runtime.pcdat...
runtime.pcval...
runtime...
14.09s
7.08s
12.66s
net/http
ReadResponse
0.01s (0.013%)
of 7.08s (9.55%)
net/http
(*Client)
Get
0.04s (0.054%)
of 14.09s (19.00%)
io/ioutil
readAll
0.12s (0.16%)
of 12.66s (17.07%)
4.43s
net/http
(*Transport)
tryPutIdleConn
0.35s (0.47%)
of 4.47s (6.03%)
3.40s
4.14s
10.64s
10.65s
net/textproto
(*Reader)
ReadMIMEHeader
0.30s (0.4%)
of 4.14s (5.58%)
net/http
(*Client)
do
0.22s (0.3%)
of 10.64s (14.35%)
bytes
(*Buffer)
ReadFrom
0.10s (0.13%)
of 10.65s (14.36%)
0.66s
9.26s
0.39s
0.20s
1.02s
0.31s
8.79s
net/http
(*Transport)
roundTrip
0.22s (0.3%)
of 9.26s (12.49%)
net/textproto
(*Reader)
readContinuedLineSlice
0.09s (0.12%)
of 0.66s (0.89%)
1.74s
1.22s
6.42s
0.10s
0.23s
net/http
(*persistConn)
roundTrip
0.37s (0.5%)
indexbytebody
0.41s (0.55%)
net/http
(*persistConn)
writ Loop
0.06s (0.081%)
bytes
makeSlice
0.06s (0.081%
of 8.92s (12.03
s
STATS (pid 1)
Heap:
Goroutines:
Threads:
PROCS (pid 0)
GC
Network
Timers
Syscalls
Proc 0
Proc 1
Proc 2
Proc 3
Proc 4
Proc 5
Proc 6
Proc 7
Proc 8
Proc 9
Proc 10
Proc 11

经验

* 一定要想好兜底 !!!
* fast fail
* 做好指标监控，关注时延和QPS，注意上线前后的cpu及mem
* 尽量在所有请求及逻辑中加入可靠的超时逻辑

其他优化

- sync.pool
  - 对象缓冲
  - 减少gc时延
- sync.map
  - 适合读多写少的场景
  - 适合缓存命中较多的场景
- 分段锁，减少锁竞争
- 拆分channel，提高管道的吞吐
- 预先分配合理的大小 (map slice)
- 善用读写锁替换独占锁
- 使用连接池
- 使用可多路复用的协议
- ...

效率库包

- ratelimiter
- circuit breaker
- waitgroup
- cache
- gpool

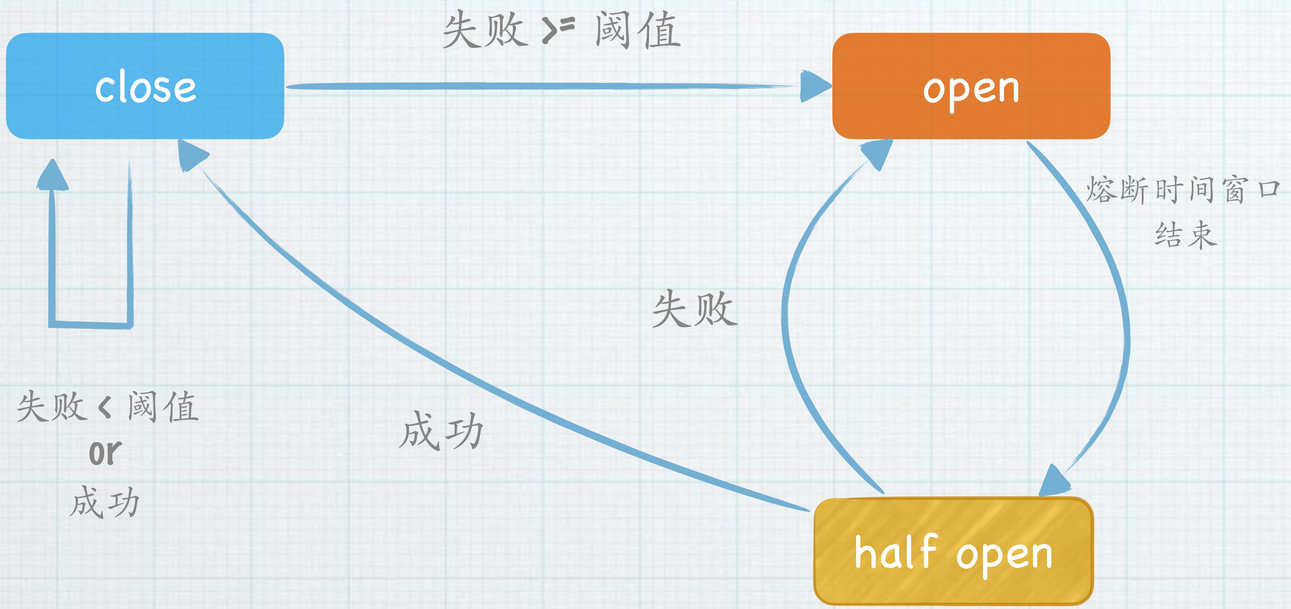
- backoff
- gjson
- gops
- cast
- retry
- ... ...

ratelimiter 限流限频

- 令牌桶
  - 可突增 (burst)
  - golang.org/x/time/rate
- 漏桶
  - 输出恒定
  - https://github.com/uber-go/ratelimit

```go
import (
    "fmt"
    "time"

    "go.uber.org/ratelimit"
    "golang.org/x/time/rate"
)

func token() {
    l := rate.NewLimiter(rate.Every(time.Duration(10)*time.Millisecond), 100)
    start := time.Now()
    for i := 0; i < 50; i++ {
        c := l.Allow()
        fmt.Println(c)
        time.Sleep(200 * time.Millisecond)
    }
}

func leaky() {
    rl := ratelimit.New(100) // per second

    prev := time.Now()
    for i := 0; i < 10; i++ {
        now := rl.Take()
        fmt.Println(i, now.Sub(prev))
        prev = now
    }
}
```

circuit breaker 熔断器




```go
import (
    "io/ioutil"
    "net/http"
    "time"

    "github.com/sony/gobreaker"
)

var cb *gobreaker.CircuitBreaker

func init() {
    var st = gobreaker.Settings{
        Timeout: time.Duration(60 * time.Second),
        ReadyToTrip: func(counts gobreaker.Counts) bool {
            failureRatio := float64(counts.TotalFailures) / float64(counts.Requests)
            return counts.Requests >= 3 && failureRatio >= 0.6
        },
    }

    cb = gobreaker.NewCircuitBreaker(st)
}

func Get(url string) ([]byte, error) {
    body, err := cb.Execute(func() (interface{}, error) {
        resp, err := http.Get(url)
        if err != nil {
            return nil, err
        }

        defer resp.Body.Close()
        body, err := ioutil.ReadAll(resp.Body)
        if err != nil {
            return nil, err
        }

        return body, nil
    })
    if err != nil {
        return nil, err
    }

    return body.([]byte), nil
}
```

* https://github.com/sony/gobreaker
* https://github.com/rfyiamcool/easybreaker

waitgroup

* 扩展标准库 waitgroup
  * merge errors
  * easy api
  * add timeout
  * add context
  * concurrency go func

```go
func Test_Simple(t *testing.T) {
    counter := 0
    lock := sync.Mutex{}
    fn := func() error {
        lock.Lock()
        defer lock.Unlock()

        counter++
        return nil
    }

    wg, ctx := New()
    wg.Async(fn)
    wg.AsyncMany(fn, 5)

    <-ctx.Done()
    wg.Wait()

    ...
}

func Test_Ctx(t *testing.T) {
    ctx, cancel := context.WithCancel(context.Background())
    defer cancel()

    wg, cctx := NewWithContext(ctx)
    wg.Async(errfn)
    wg.AsyncMany(fn, 5)

    select {
    case <-cctx.Done():
    case <-ctx.Done():
    }

    ...
}

func Test_WaitTimeout(t *testing.T) {
    ctx, cancel := context.WithCancel(context.Background())
    defer cancel()

    wg, cctx := NewWithContext(ctx)
    // _ = cctx

    wg.Async(
        func() error {
            time.Sleep(5 * time.Second)
            return io.ErrClosedPipe
        },
    )

    wg.WaitTimeout(1 * time.Second)
    <-cctx.Done()
}
```

https://github.com/rfyiamcool/waitgroup

cache

- interface
  - https://github.com/karlseguin/ccache
  - https://github.com/VictoriaMetrics/fastcache
  - https://github.com/dgraph-io/ristretto
- ringbuffer
  - https://github.com/coocood/freecache
  - https://github.com/allegro/bigcache

```go
import (
    "github.com/allegro/bigcache"
)

config := bigcache.Config {
        // number of shards (must be a power of 2)
        Shards: 1024,

        // 标记过期时间
        LifeWindow: 10 * time.Minute,

        // 删除无效key的检查时间间隔
        CleanWindow: 5 * time.Minute,

        // entry的值大小
        MaxEntrySize: 500,

        // 内存大小限制
        HardMaxCacheSize: 8192,

        // 删除回调
        OnRemove: nil,
    }

cache, initErr := bigcache.NewBigCache(config)
if initErr != nil {
    log.Fatal(initErr)
}

cache.Set("my-unique-key", []byte("value"))

if entry, err := cache.Get("my-unique-key"); err == nil {
    fmt.Println(string(entry))
}
```

gpool 协程池

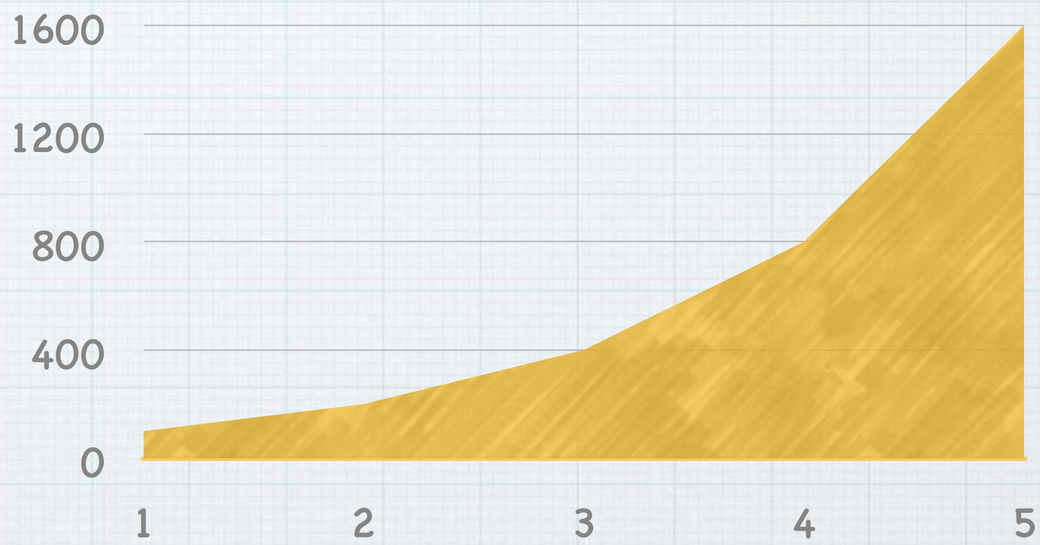
* 对上游调用流量整形
* more stack
* 规避由协程暴增后，gc对gfree的影响

```go
package main

import (
    "fmt"
    "sync"
    "time"

    "github.com/rfyiamcool/gpool"
)

var (
    wg = sync.WaitGroup{}
)

func main() {
    gp, err := gpool.NewGPool(&gpool.Options{
        MaxWorker:   5,                // 最大的协程数
        MinWorker:   2,                // 最小的协程数
        JobBuffer:   1,                // 缓冲队列的大小
        IdleTimeout: 120 * time.Second, // 协程的空闲超时退出时间
    })

    if err != nil {
        panic(err.Error())
    }

    for index := 0; index < 1000; index++ {
        wg.Add(1)
        idx := index
        gp.ProcessAsync(func() {
            fmt.Println(idx, time.Now())
            time.Sleep(1 * time.Second)
            wg.Done()
        })
    }

    wg.Done()
}
```

https://github.com/rfyiamcool/gpool

backoff 退避

```go
func Test1(t *testing.T) {
    b := NewBackOff(
        WithMinDelay(100*time.Millisecond),
        WithMaxDelay(10*time.Second),
        WithFactor(2),
    )

    equals(t, b.Duration(), 100*time.Millisecond)
    equals(t, b.Duration(), 200*time.Millisecond)
    equals(t, b.Duration(), 400*time.Millisecond)
    for index := 0; index < 100; index++ {
        b.Duration()
    }

    // is max
    equals(t, b.Duration(), 10*time.Second)
    b.Reset()
    equals(t, b.Duration(), 100*time.Millisecond)
}

func TestJitter(t *testing.T) {
    b := NewBackOff(
        WithMinDelay(100*time.Millisecond),
        WithMaxDelay(10*time.Second),
        WithFactor(2),
        WithJitterFlag(true),
    )

    equals(t, b.Duration(), 100*time.Millisecond)
    between(t, b.Duration(), 100*time.Millisecond, 200*time.Millisecond)
    between(t, b.Duration(), 100*time.Millisecond, 400*time.Millisecond)
    b.Reset()
    equals(t, b.Duration(), 100*time.Millisecond)
}
```



https://github.com/rfyiamcool/backoff

retry 重试

* each delay time
* backoff
* timeout
* times
* context
* recovery

```go
package main

import (
    "errors"
    "log"

    "github.com/rfyiamcool/go-retry"
)

func main() {
    r := retry.New()
    var running = false
    err := r.Ensure(func() error {
        if !running {
            log.Println("to retry")
            running = true
            return retry.Retriable(errors.New("diy"))
        }

        log.Println("ok")
        return nil
    })
    if err != nil {
        log.Fatal(err)
    }
}
```

https://github.com/rfyiamcool/go-retry

gjson 模拟查找

```go
package main

import "github.com/tidwall/gjson"

const json = `{"name":{"first":"Janet","last":"Prichard"},"age":47}`

func main() {
    value := gjson.Get(json, "name.last")
    println(value.String())
}
```

```
{
    "name": {"first": "Tom", "last": "Anderson"},
    "age":37,
    "children": ["Sara","Alex","Jack"],
    "fav.movie": "Deer Hunter",
    "friends": [
      {"first": "Dale", "last": "Murphy", "age": 44, "nets": ["ig", "fb", "tw"]},
      {"first": "Roger", "last": "Craig", "age": 68, "nets": ["fb", "tw"]},
      {"first": "Jane", "last": "Murphy", "age": 47, "nets": ["ig", "tw"]}
    ]
}

```
"name.last"          >> "Anderson"
"age"                >> 37
"children"           >> ["Sara","Alex","Jack"]
"children.#"         >> 3
"children.1"         >> "Alex"
"child*.2"           >> "Jack"
"c?ildren.0"         >> "Sara"
"fav\.movie"         >> "Deer Hunter"
"friends.#.first"    >> ["Dale","Roger","Jane"]
"friends.1.last"     >> "Craig"

"friends.#(last=="Murphy").first"    >> "Dale"
"friends.#(last=="Murphy")#.first"   >> ["Dale","Jane"]
"friends.#(age>45)#.last"            >> ["Craig","Murphy"]
"friends.#(first%"D*").last"         >> "Murphy"
"friends.#(first!%"D*").last"        >> "Craig"
"friends.#(nets.#(=="fb"))#.first"   >> ["Dale","Roger"]
```

无需预先定义结构
就匹配查找！

https://github.com/tidwall/gjson


```

gops 调试利器

* base
    * gops <pid>
    * gops stack
    * gops memstats
    * gops stats
    * ...
* pprof
    * gops pprof-cpu
    * gops pprof-heap
    * gops trace

```
$ gops stack (<pid>|<addr>)
gops stack 85709
goroutine 8 [running]:
runtime/pprof.writeGoroutineStacks(0x13c7bc0, 0xc42000e008, 0xc420ec8520, 0xc420ec8520)
    /Users/jbd/go/src/runtime/pprof/pprof.go:603 +0x79
runtime/pprof.writeGoroutine(0x13c7bc0, 0xc42000e008, 0x2, 0xc428f1c048, 0xc420ec8608)
    /Users/jbd/go/src/runtime/pprof/pprof.go:592 +0x44
runtime/pprof.(*Profile).WriteTo(0x13eeda0, 0x13c7bc0, 0xc42000e008, 0x2, 0xc42000e008, 0x0)
    /Users/jbd/go/src/runtime/pprof/pprof.go:302 +0x3b5
github.com/google/gops/agent.handle(0x13cd560, 0xc42000e008, 0xc420186000, 0x1, 0x1, 0x0, 0x0)
    /Users/jbd/src/github.com/google/gops/agent/agent.go:150 +0x1b3
github.com/google/gops/agent.listen()
    /Users/jbd/src/github.com/google/gops/agent/agent.go:113 +0x2b2
created by github.com/google/gops/agent.Listen
    /Users/jbd/src/github.com/google/gops/agent/agent.go:94 +0x480
# ...
```

```
package main

import (
    "log"
    "time"

    "github.com/google/gops/agent"
)

func main() {
    if err := agent.Listen(agent.Options{}); err != nil {
        log.Fatal(err)
    }
    time.Sleep(time.Hour)
}
```

https://github.com/google/gops

cast 万能转换

* 丑陋的 `strconv`
* 类型转换

```
func StringToDate(s string) (time.Time, error)
func ToBool(i interface{}) bool
func ToBoolE(i interface{}) (bool, error)
func ToBoolSlice(i interface{}) []bool
func ToBoolSliceE(i interface{}) ([]bool, error)
func ToDuration(i interface{}) time.Duration
func ToFloat32(i interface{}) float32
func ToFloat64(i interface{}) float64
func ToInt(i interface{}) int
func ToInt16(i interface{}) int16
func ToIntE(i interface{}) (int, error)
func ToIntSlice(i interface{}) []int
func ToSlice(i interface{}) []interface{}
func ToSliceE(i interface{}) ([]interface{}, error)
func ToStringE(i interface{}) (string, error)
func ToStringMap(i interface{}) map[string]interface{}
func ToStringMapBool(i interface{}) map[string]bool
func ToStringMapInt(i interface{}) map[string]int
func ToStringMapInt64(i interface{}) map[string]int64
func ToStringMapStringSlice(i interface{}) map[string][]string
func ToStringMapStringSliceE(i interface{}) (map[string][]string, error)
func ToStringSlice(i interface{}) []string
func ToStringSliceE(i interface{}) ([]string, error)
func ToTime(i interface{}) time.Time
func ToTimeE(i interface{}) (tim time.Time, err error)
func ToUint(i interface{}) uint
...
```

https://github.com/spf13/cast

other

* https://github.com/jinzhu/copier
* https://github.com/davecgh/go-spew
* https://github.com/abice/go-enum
* https://github.com/mitchellh/mapstructure
* https://github.com/rfyiamcool/go-stats
* ...

“Q&A!”

-峰云就她了