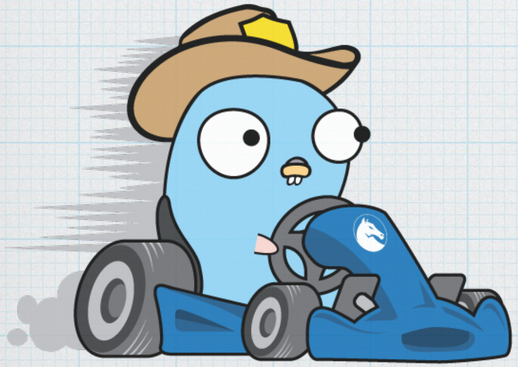
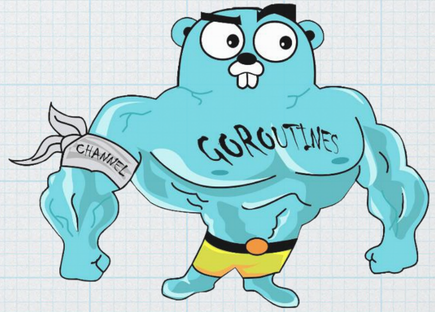
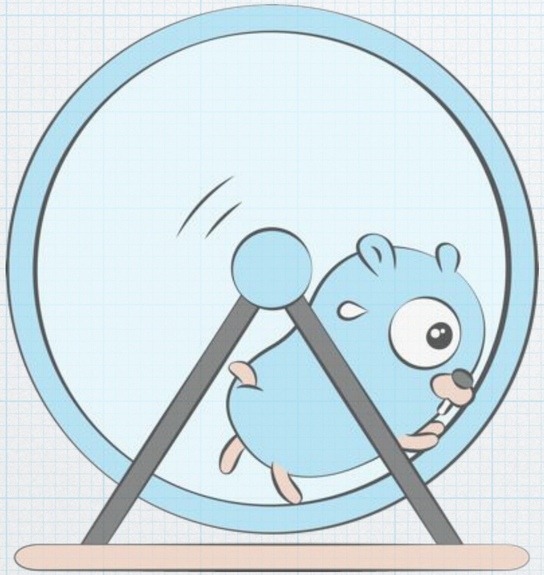
Golang 项目实战

v1

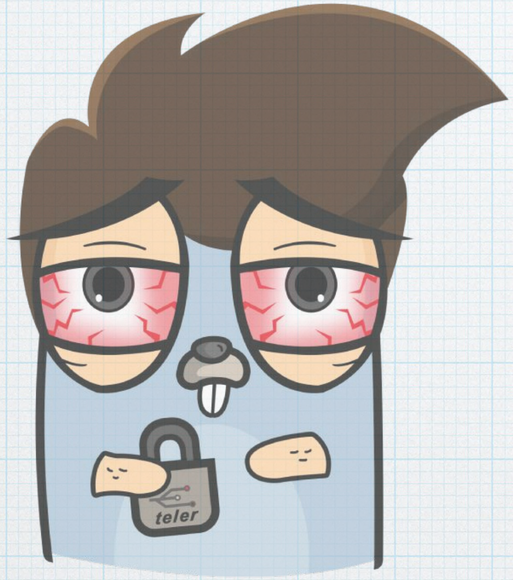


峰云 @容器云





* 代码规范
* git 规范
* 设计模式
* 并发编程
* 效率库包
* Q&A



项目规范

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 (
  isDone       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

* 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

编码规范

```
// bad
if err != nil {
  // error code
} else {
  // normal code
}
```

```
// good
if err != nil {
  // error handling
  return err
}
// normal code
```

及时 return

```
for closed := false; !closed; {
    select {
    case fields := <-queue:
    default:
      closed = true
    }
}
```

使用临时 bool 来退出

for select/switch

循环

```
// bad
type Client struct {
  version int
  http.Client
}
```

```
// good
type Client struct {
  http.Client

  version int
}
```

编码规范

```go
package main

func main() {
   num := 10

   if num == 1 {
   } else if num == 2 {
   } else if num == 5 {
   } else if num == 7 {
   } else if num == 8 {
   } else if num == 10 {
   }

   switch num {
   case 2:
   case 5:
   case 7:
   case 8:
   case 10:
   }

   var (
       mapping := make(map[int]func(), 10)
   )

   register := func(n int, fn func()) {
      mapping[n] = fn
   }

   get := func(n int) func() {
      return mapping[n]
   }

   register(num, func() {})
   get(num)()
}
```

* 简单条件使用 **switch**
* 复杂条件使用 **if else**
* 太深嵌套的 **if else** 拆分函数

```java
@Override
public void run() {
    if (state == ChannelState.RECEIVED) {
        try {
            handler.received(channel, message);
        } catch (Exception e) {
            logger.warn( msg: "ChannelEventRunnable handle " + state + " operation er
                    + ", message is " + message, e);
        }
    } else {
        switch (state) {
        case CONNECTED:
            try {
                handler.connected(channel);
            } catch (Exception e) {
                logger.warn( msg: "ChannelEventRunnable handle " + state + " operatio
            }
            break;
        case DISCONNECTED:
            try {
                handler.disconnected(channel);
            } catch (Exception e) {
                logger.warn( msg: "ChannelEventRunnable handle " + state + " operatio
            }
            break;
        case SENT:
            try {
                handler.sent(channel, message);
            } catch (Exception e) {
```

Dubbo: cpu 的分支预测 😅

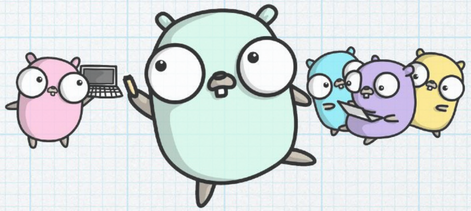
Go工程目录

- cmd
- api
- configs
- docs
- internal
- pkg
- scrtips
- tools
- third_party
- README.md

- internal
  - conf
  - server
  - router
  - controller
  - service
  - dao

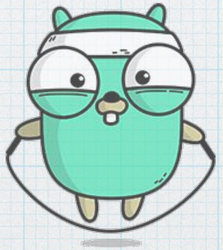
适合自己就好，不要过度纠结 !!!

ddd ?



这个目录吧？

仁者见仁，智者见智



适合自己就好！

一眼能看明白就好！

test

* Testify
  * assert
  * mock
* GoMock
* Monkey
* Gotests

```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)
    }
}
```

```go
func TestAbs_1(t *testing.T) {
    tests := []struct {
        x    float64
        want float64
    }{
        {-0.3, 0.3},
        {-2, 2},
        {-3.1, 3.1},
        {5, 5},
    }

    for _, tt := range tests {
        got := Abs(tt.x)
        assert.Equal(t, got, tt.want)
    }
}
```

```go
func TestXxx(t *testing.T) {
    type args struct {
        // TODO: Add function input parameter definition.
    }

    type want struct {
        // TODO: Add function return parameter definition.
    }
    tests := []struct {
        name string
        args args
        want want
    }{
        // TODO: Add test cases.
    }
    for _, tt := range tests {
        t.Run(tt.name, func(t *testing.T) {
            if got := Xxx(tt.args); got != tt.want {
                t.Errorf("Xxx() = %v, want %v", got, tt.want)
            }
        })
    }
}
```

使用 github.com/cweill/gotests 生成测试模板

```go
//Sprintf
func BenchmarkSprintf(b *testing.B) {
    num := 10
    b.ResetTimer()
    for i := 0; i < b.N; i++ {
        fmt.Sprintf("%d", num)
    }
}

//Format
func BenchmarkFormat(b *testing.B) {
    num := int64(10)
    b.ResetTimer()
    for i := 0; i < b.N; i++ {
        strconv.FormatInt(num, 10)
    }
}

//Itoa
func BenchmarkItoa(b *testing.B) {
    num := 10
    b.ResetTimer()
    for i := 0; i < b.N; i++ {
        strconv.Itoa(num)
    }
}
```

```
%  go test -bench=. -benchmem
goos: darwin
goarch: amd64
pkg: program/benchmark
cpu: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
BenchmarkSprintf-12     16364854                63.70 ns/op            2 B/op          1 allocs/op
BenchmarkFormat-12      493325650                2.380 ns/op           0 B/op          0 allocs/op
BenchmarkItoa-12        481683436                2.503 ns/op           0 B/op          0 allocs/op
PASS
ok      program/benchmark    4.007s
```

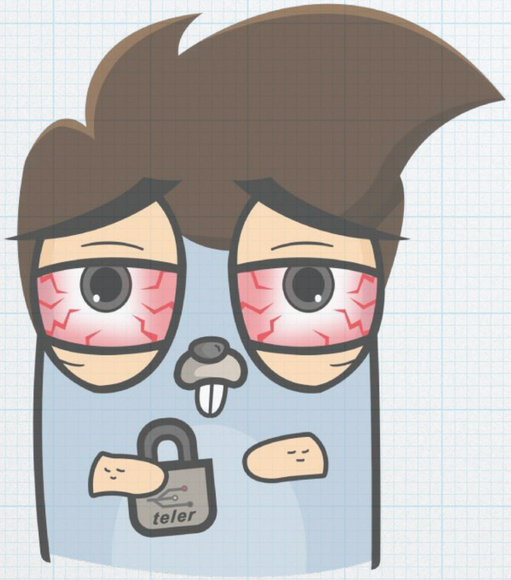
* mem
    * go test -bench=. -benchmem -cpuprofile=cpu.out
    * go tool pprof cpu.out
* cpu
    * go test -bench=. -benchmem -memprofile=mem.out
    * go tool pprof mem.out

makefile

- 统一执行入口
  - make test
  - make fmt
  - make lint
  - make build
  - ...

项目操作命令收敛在 Makefile 里.

```
.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
```



接口规范

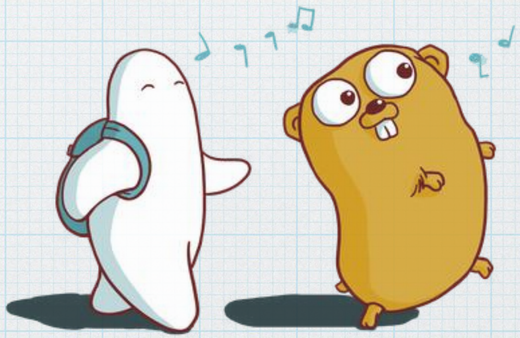
restful

* get
  * 查询数据
* post
  * 新增数据（非幂等）
* put
  * 新增数据（幂等）
  * 修改数据
* delete
  * 删除数据
* patch
  * 修改部分字段的数据

建议按照这个标准走！

建议按照这个标准走！

建议按照这个标准走！



restful

- 新增用户
  - post /users
- 修改操作
  - put /users/{id}
  - put /users/{id} {name: xx, age: xx}
  - put /users/{id}/name
- 删除用户
  - delete /users/{id}
  - delete /users?ids=1,2,3,4,5
  - delete /users&sex=boy
- 查询用户
  - get /users/{id}
  - get /users?ids=1,2,3,4,5
  - get /users?location=bj&job=coder&page=100&limit=100
  - get /users?age_gt=10&age_lt=30job=tester,coder

restful

* GET /authors/12/categories/2
* GET /authors/12?categories=2

避免使用多级目录

* 接口需要加入版本声明 /api/v1/ . . .
* uri 结尾不要有 /
* 资源名要使用名词
* uri 路径要用小写

* GET /author/12
* GET /authors/12

避免使用单数

根据需求使用 post 实现复杂修改和查询操作.

batch method

```
# 批量创建
POST /api/resources

Body: {
        "data": [ { "name": "Mr.Bean" }
}

# 批量更新
PUT /api/resources

Body: {
        "data": { "1": { "name": "Mr.B
}

# 批量删除
DELETE /api/resources
Body: {
        "data": [1, 2, 3]
}
```

```
# 批量创建
POST /api/resources

Body: {
        "method": "create",
        "data": [ { "name": "Mr.Bean" }, { "name": "Chaplin" }, { "name": "Jim Carrey" } ]
}

# 批量更新
POST /api/resources

Body: {
        "method": "update",
        "data": { "1": { "name": "Mr.Bean" }, "2": { "name": "Chaplin" } }
}

# 批量删除
POST /api/resources
Body: {
        "method": "delete",
        "data": [1, 2, 3]
}
```

http code

* http 2xx
  * 200 成功
  * 201 创建成功
* http 3xx
  * 301 永久重定向
  * 302 临时重定向
* http 4xx
  * 400 请求错误
  * 401 未授权
  * 403 无权限
  * 404 未找到
  * 405 method方法错误
  * 429 Too Many Requests
* http 5xx
  * 500 内部服务器错误
  * 502 无效网关
  * 503 服务不可用
  * 504 转发超时

http code

```
{
    "code": 0,
    "message": "    ",
    "data": {
        ...
        ...
        ...
    }
}
```

* 访问用户信息（未登陆）
  * http status code = 401
  * code = 401
  * msg = 未登陆
* 访问用户信息（空用户）
  * http status code = 200? 400 ? 404
  * code = 100101
  * msg = 无此用户
* 修改用户信息（邮箱唯一）
  * http status code = 403
  * code = 100102
  * msg = 邮箱不能修改
* 修改用户信息（数据库操作发生异常）
  * http status code = 500
  * code = 500
  * msg = 服务内部异常

http code

* 业务code 100101
  * 10 服务
  * 01 模块
  * 01 具体错误标号

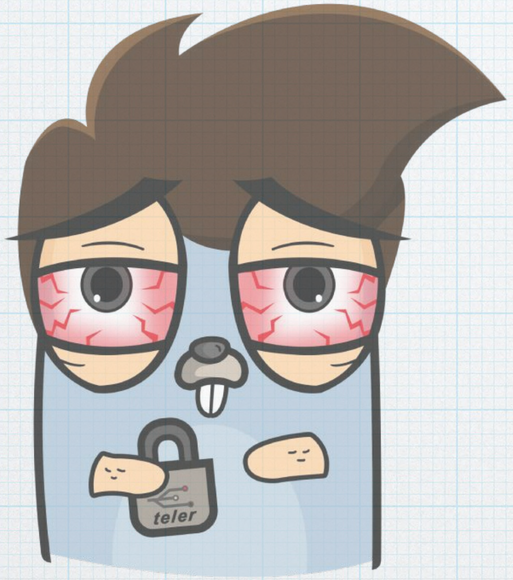
* 社区的争论：(对于业务上的异常的返回)
  * 一梭子返回 http 200, 定义业务errcode ;
    * facebook
    * bilibili
    * ...
  * 根据错误类型返回不同的 http code, 同样含有errcode ;
    * twitter
    * bing
    * ...

grpc

- grpc
  - protobuf
    - 生成静态代码规避运行时反射
    - 避免json这种不定长频繁的边界判断
    - tag 顺序且压缩
    - varint, zigzag 编码
  - http 2.0
    - 多路复用，避免了 h1.0 的队头阻塞
    - 避免 h1.0 连接池下的连接开销
    - 二进制头部压缩
- mode
  - unary
  - client streaming
  - server streaming
  - bidi streaming

- 简单易用
- 性能好
- idl强约束

```proto
option go_package = "api/proto;proto";

// User 服务接口列表
service UserService {
    // AddUser 增加一个用户
    rpc AddUser(User) returns (User) {
    }

    // GetUser 查询用户
    rpc GetUser(GetUserReq) returns (GetUserResp) {
    }

    // Put 修改用户
    rpc UpdateUser(User) returns (UpdateUserResp) {
    }

    // DeleteUser 删除一个用户
    rpc DeleteUser(DeleteUserReq) returns (common.Response) {
    }

    // ListUsers 按条件查询匹配用户
    rpc ListUsers(ListUsersReq) returns (ListUsersResp) {
    }
}
```

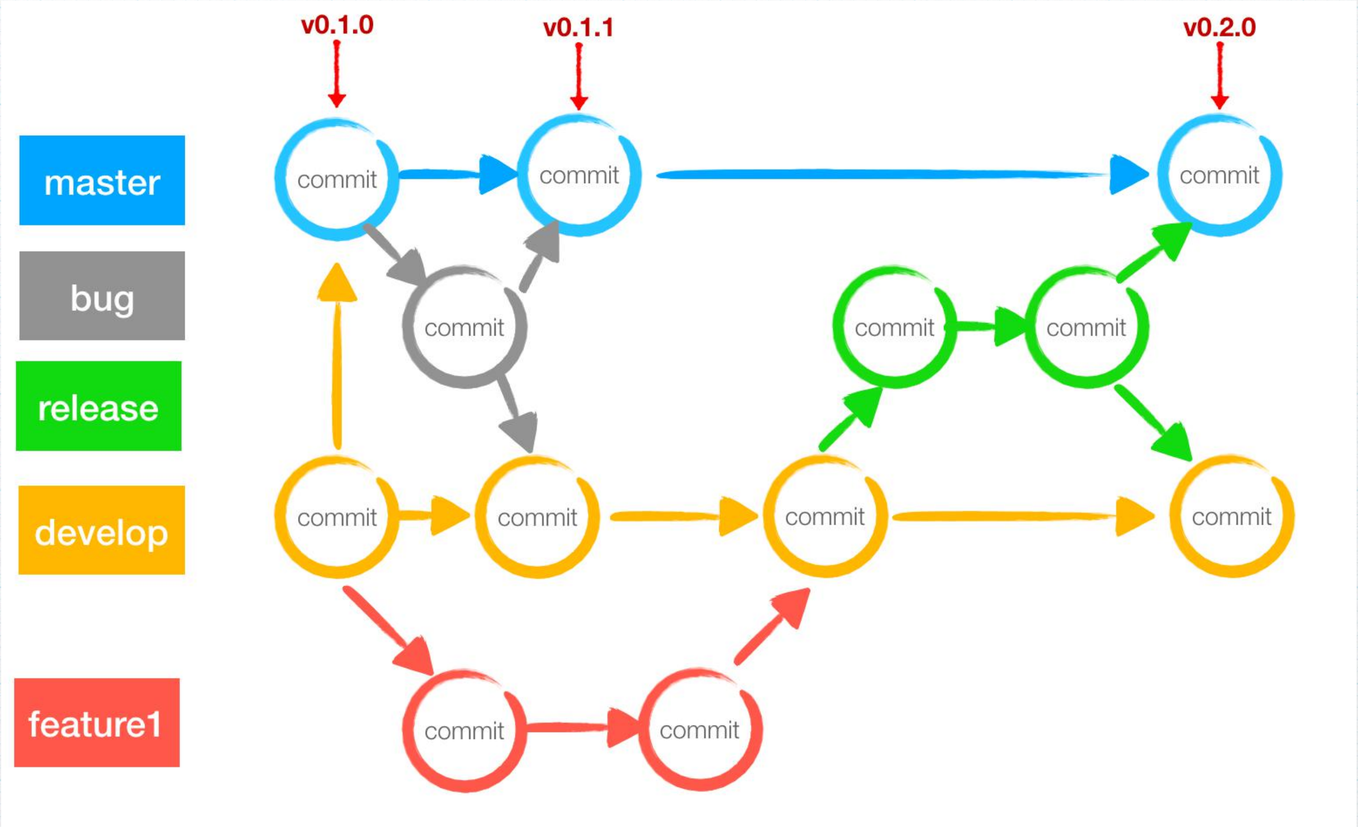


git 规范

git

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



simple gitflow

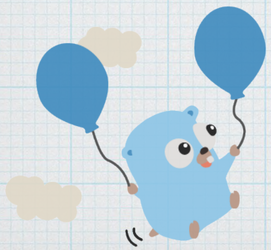
Master

hot fix

develop

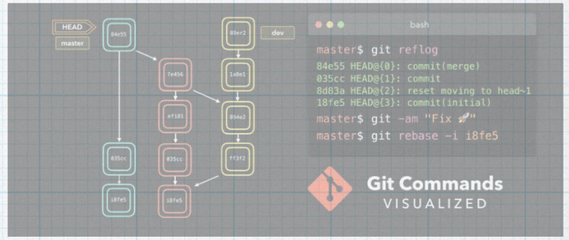
feature

feature



git 规范

- 分支管理
  - 代码提交在应该提交的分支上
  - 随时可以切换到线上稳定版本代码
  - 多个版本的开发工作同时进行
- 记录的可读性
  - commit内容按照格式执行.
  - 正确设置user.name和user.email信息.
- 版本号(tag)
  - 版本号(tag)命名规则
  - 主版本号.次版本号.修订号，如2.1.13
  - 对master标记tag意味着该tag能发布到生产环境
  - 版本号仅标记于master分支，用于标识某个可发布/回滚的版本代码
  - 仅项目管理员有权限对master进行合并和标记版本号



git commit规范

- type
  - feat: 新功能
  - fix: 修复 bug
  - docs: 文档变动
  - style: 单纯的格式调整
  - refactor: 修复和添加新功能之外的代码改动
  - perf: 提升性能的改动
  - test: 添加或修正测试代码
- <type>(<scope>): <subject>
  - feat(detail): 详情页修改样式
  - fix(login): 登录页面错误处理
  - test(list): 列表页添加测试代码
- scope 修改范围
  - 这次修改涉及到的部分简单概括
  - login、train-order
- subject 修改的描述
  - 具体的修改描述信息
  - provide issue id
- <type>(<scope>): <subject>
  - feat(detail): 详情页修改样式
  - fix(login): 登录页面错误处理
  - test(list): 列表页添加测试代码

reset vs revert

- 强烈建议
  - 在公共分支做 revert .
    - revert 会删除某提交的变更，然后产生新的提交，并不会真正删除history.
  - 在自己分支做 reset .
    - soft: 回滚到某提交，但变更保留在工作区内，供开发者选择
    - hard: 彻彻底底的回滚到某提交，清空暂存区和工作区.

不要在公共分支做 reset 操作 !!!

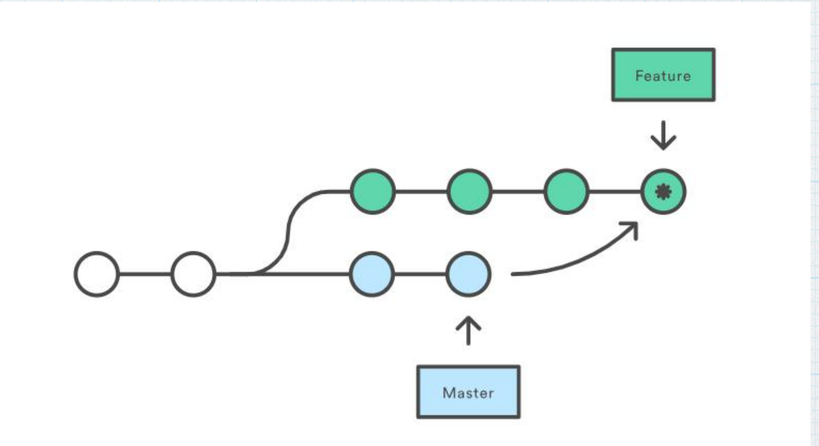
rebase vs merge

* 建议
  * 下游分支更新上游分支内容的时候使用 **rebase**
  * 上游分支合并下游分支内容的时候使用 **merge**
  * 更新当前分支的内容时一定要使用 **--rebase** 参数

不建议在公共分支上使用rebase

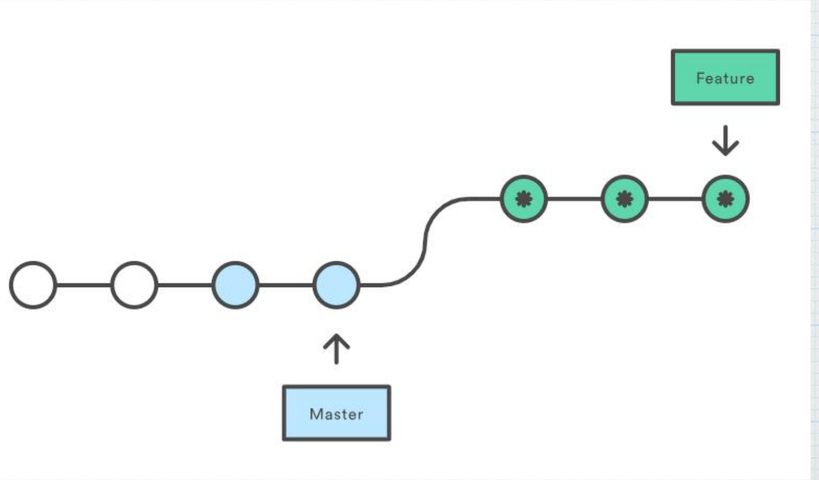
merge

注重历史记录



rebase

注重线性提交



合并多个commit记录

- merge squash
  - 在上游分支操作
  - git merge -squash <branch>
- rebase squash
  - 在下游分支操作
  - git rebase -i HEAD^4

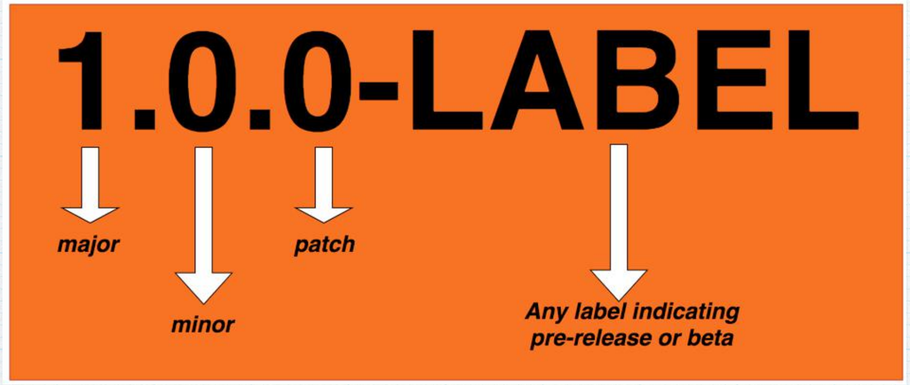

```
s 9a54fd4 update1
s 0d4a808 下班提交
s 159db45 吃饭提交
pick 92f277a 改完了

# Rebase 326fc9f..0d4a808 onto d286baa
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x, exec = run command (the rest of the line) using shell
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

使代码的提交记录更加规整 !!!

skills

- 配置 git hook
  - 代码分析
  - 代码格式优化
  - 单元测试
  - ...
- 别放大文件，删除真的很麻烦！
- 使用 .gitignore 忽略文件
- 建议使用 git pull - -rebase 拉取代码
- 借用 stash 特性随意切换分支
- 使用 reflog 解决误操作的重定向 HEAD
- git clean -fd
- git cherry-pick
- git commit --amend
- git clone --depth 限制拉取深度，加快克隆代码

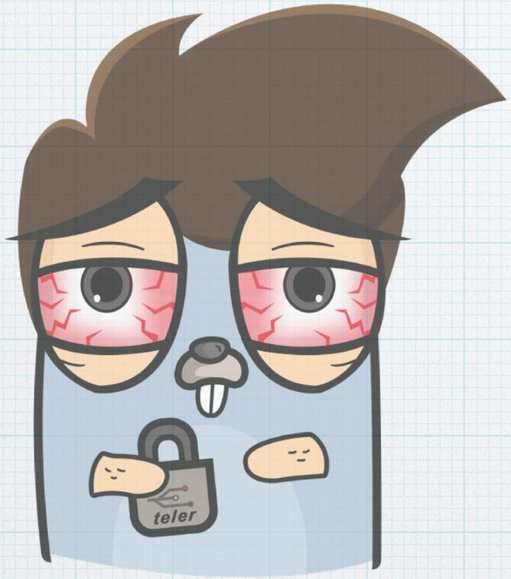
semver



* 主版本号 (major)
  * 当有重大升级改动.
  * 当做了不兼容的 API 修改.
* 次版本号 (minor)
  * 当做了向下兼容的功能性新增.
* 修订号 (patch)
  * 当做了向下兼容的问题修正.

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

* label
  * alpha: 内部测试版
  * beta: 公开测试版
  * rc: 候选版本, 不再增加新功能.

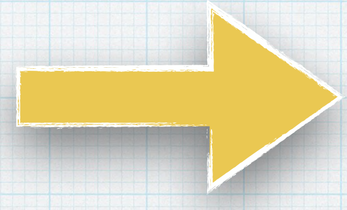


设计方法

单例模式

饿汉模式

```go
package singleton

type singleton struct {
}

var ins *singleton = &singleton{}

func GetInstance() *singleton {
    return ins
}
```



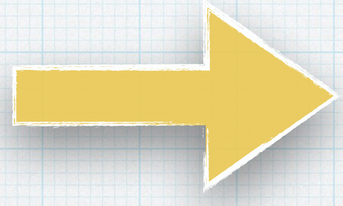
懒汉模式

```go
package singleton

import (
    "sync"
)

type singleton struct {
}

var (
    ins *singleton
    once sync.Once
)


func GetInstance() *singleton {
    once.Do(func() {
        ins = &singleton{}
    })
    return ins
}
```

工厂模式

```go
type Person interface {
    Greet()
}

type person struct {
    name string
    age  int
}

func (p person) Greet() {
    fmt.Printf("Hi! My name is %s", p.name)
}

func NewPerson(name string, age int) Person {
    return person{
        name: name,
        age:  age,
    }
}
```

简单工厂模式

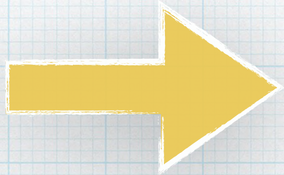


工厂方法模式

```go
type Person interface {
    Greet()
}

type person struct {
    name string
    age  int
}

func (p person) Greet() {
    fmt.Printf("Hi! My name is %s", p.name)
}

func NewPersonFactory(age int) func(name string) Person {
    return func(name string) Person {
        return Person{
            name: name,
            age:  age,
        }
    }
}

func main() {
    newBaby := NewPersonFactory(1)
    baby := newBaby("john")

    newTeenager := NewPersonFactory(16)
    teen := newTeenager("jill")
}
```

选项模式

```go
const (
    defaultTimeout = 10
    defaultCaching = false
)

type Connection struct {
    addr    string
    cache   bool
    timeout time.Duration
}

type Options struct {
    Caching bool
    Timeout time.Duration
}

// NewConnect creates a connection with options.
func NewConnect(addr string, opts *Options) (*Connection, error) {
    return &Connection{
        addr:    addr,
        cache:   opts.Caching,
        timeout: opts.Timeout,
    }, nil
}
```



```go
type Connection struct {
    addr    string
    cache   bool
    timeout time.Duration
}

const (
    defaultTimeout = 10
    defaultCaching = false
)

type options struct {
    timeout time.Duration
    caching bool
}

type optionFunc func(*options) error

func WithTimeout(t time.Duration) optionFunc {
    return func(o *options) error {
        o.timeout = t
        return nil
    }
}

func WithCaching(cache bool) optionFunc {
    return func(o *options) error {
        o.caching = cache
        return nil
    }
}

// Connect creates a connection.
func NewConnect(addr string, opts ...optionFunc) (*Connection, error) {
    options := options{
        timeout: defaultTimeout,
        caching: defaultCaching,
    }

    for _, o := range opts {
        err := o(&options)
        if err != nil {
            return nil, err
        }
    }

    return &Connection{
        addr:    addr,
        cache:   options.caching,
        timeout: options.timeout,
    }, nil
}
```

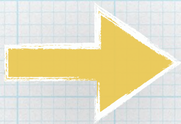
建设者模式

```
## before

p := new(product)
p.setName(...)
p.setOption(...)
p.setClient(...)
p.build()


## after

p := new(product)
p.Name(...).Option(...).Client(...).build()
```

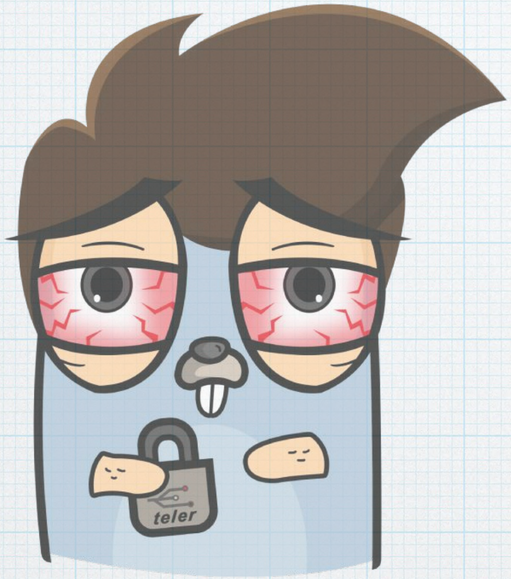
```
type Query struct {}

func (q *Query) Select() *Query {
    return q
}

func (q *Query) Where() *Query {
    return q
}

func (q *Query) OrderBy() *Query {
    return q
}

func (q *Query) GroupBy() *Query {
    return q
}

func (q *Query) Find() (interface{}, error) {
    return interface{}, nil
}

func main() {
    query := Query()
    query.Select().Where().OrderBy().GroupBy().Find()
}
```

面向接口编程

- 代码的扩展性
- 提高了代码的健壮性和稳定性
  - 调用方只能使用 **interface** 定义的方法集
  - 屏蔽内部的具体实现
- 解耦上下游的实现
  - 不关心你是如何 **speak** 的，你会 **speak** 就可以了
- 提高了代码可测性
  - 自定义 **mock / fake** 类型替换



```
type   AnimalSpeaker   interface {
    Speak() string
}

type Dog struct {
}

func (d *Dog) Speak() string {
    return "Woof!"
}

type Cat struct {
}

func (c *Cat) Speak() string {
    return "Meow!"
}

type Horse struct {
}

func (h *Horse) Speak() string {
    return "?????"
}
```



golang 一些技巧

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))
}
```

- defer
  - lock/unlock
  - conn pool get/release
  - sync.pool get/put
  - timer.stop
  - ... ...

```
// 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
}
```

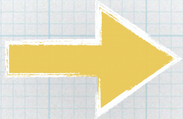
defer

* 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()
}
```

thread safe

```go
package main

import (
    "fmt"
    "sync"
)

var incr int32

func main() {
    wg := sync.WaitGroup{}

    for i := 0; i < 1000; i++ {
        wg.Add(1)
        go func() {
            defer wg.Done()
            incr++
        }()
    }

    wg.Wait()

    fmt.Println(incr)
}
```



```
933
973
983
971
893
881
979
```

num ++

1. get num
2. num = num + 1
3. set num

* thread safe
  * sync.Mutex
  * sync/Atomic

cow/rcu

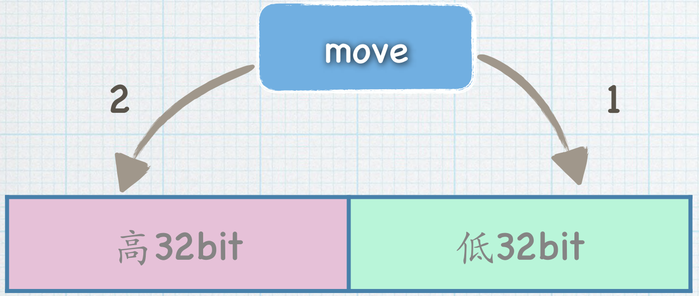
```
CMPL    runtime.writeBarrier(SB), $0
JNE     911
MOVQ    "".&m+64(SP), AX
MOVUPS  (AX), X0
MOVUPS  X0, "".dict(SB)
MOVUPS  16(AX), X0
MOVUPS  X0, "".dict+16(SB)
MOVUPS  32(AX), X0
MOVUPS  X0, "".dict+32(SB)
MOVUPS  48(AX), X0
MOVUPS  X0, “".dict+48(SB)
. . .
…
```

* sync.mutex
* atomic.Pointer

```
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
```

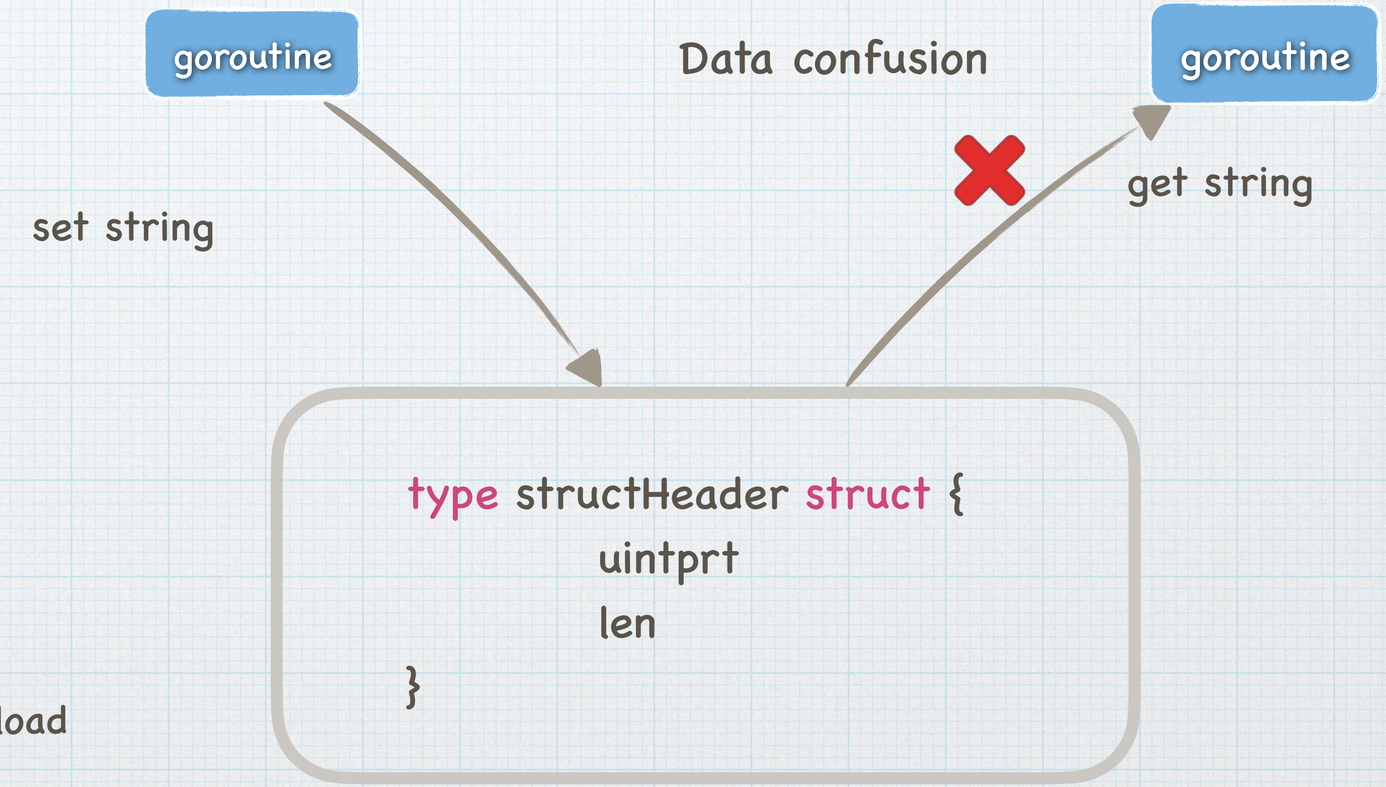
```
var dict = sync.Map{}

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

func main() {
    go func() {
        for {
            time.Sleep(1 * time.Second)
            overlay()
        }
    }()

    go func() {
        for {
            time.Sleep(1000 * 1000)
            _, _ = dict.Load("key")
        }
    }()
    ...
}
```

data race



* data race
  * int64 in 32bit system
  * string

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



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
```

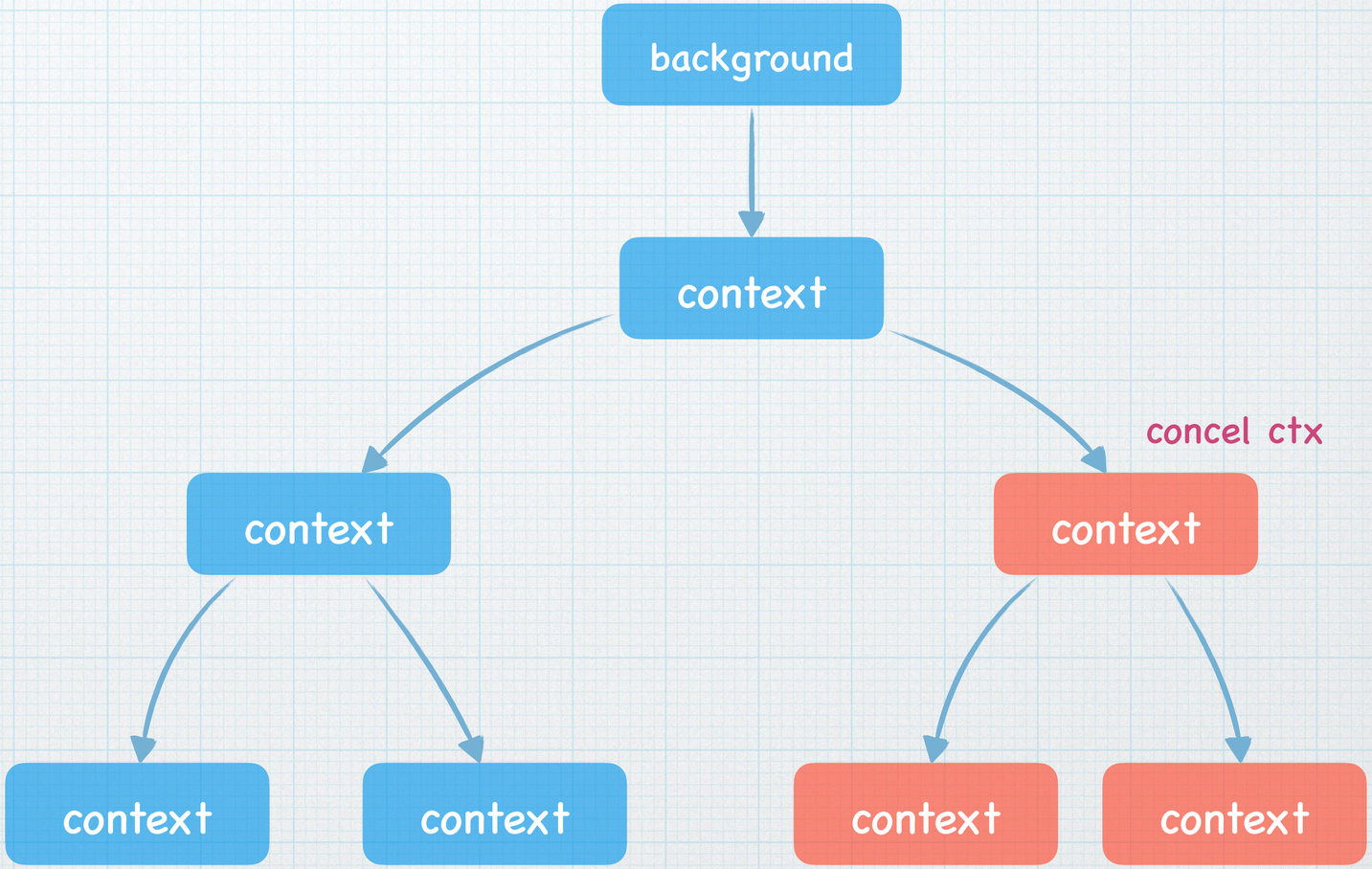
优雅退出

- 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)))
```

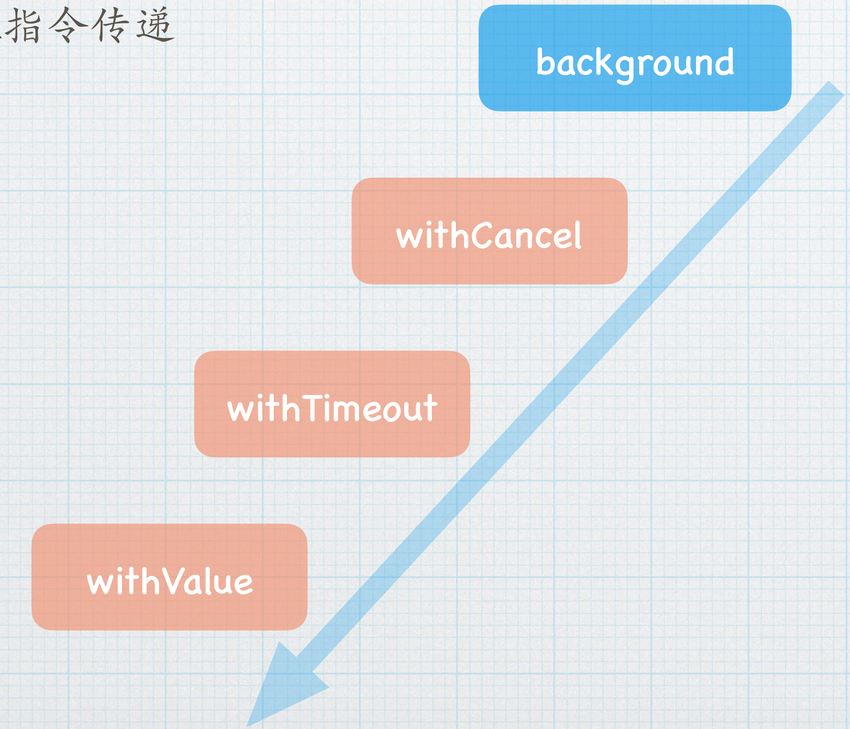
context

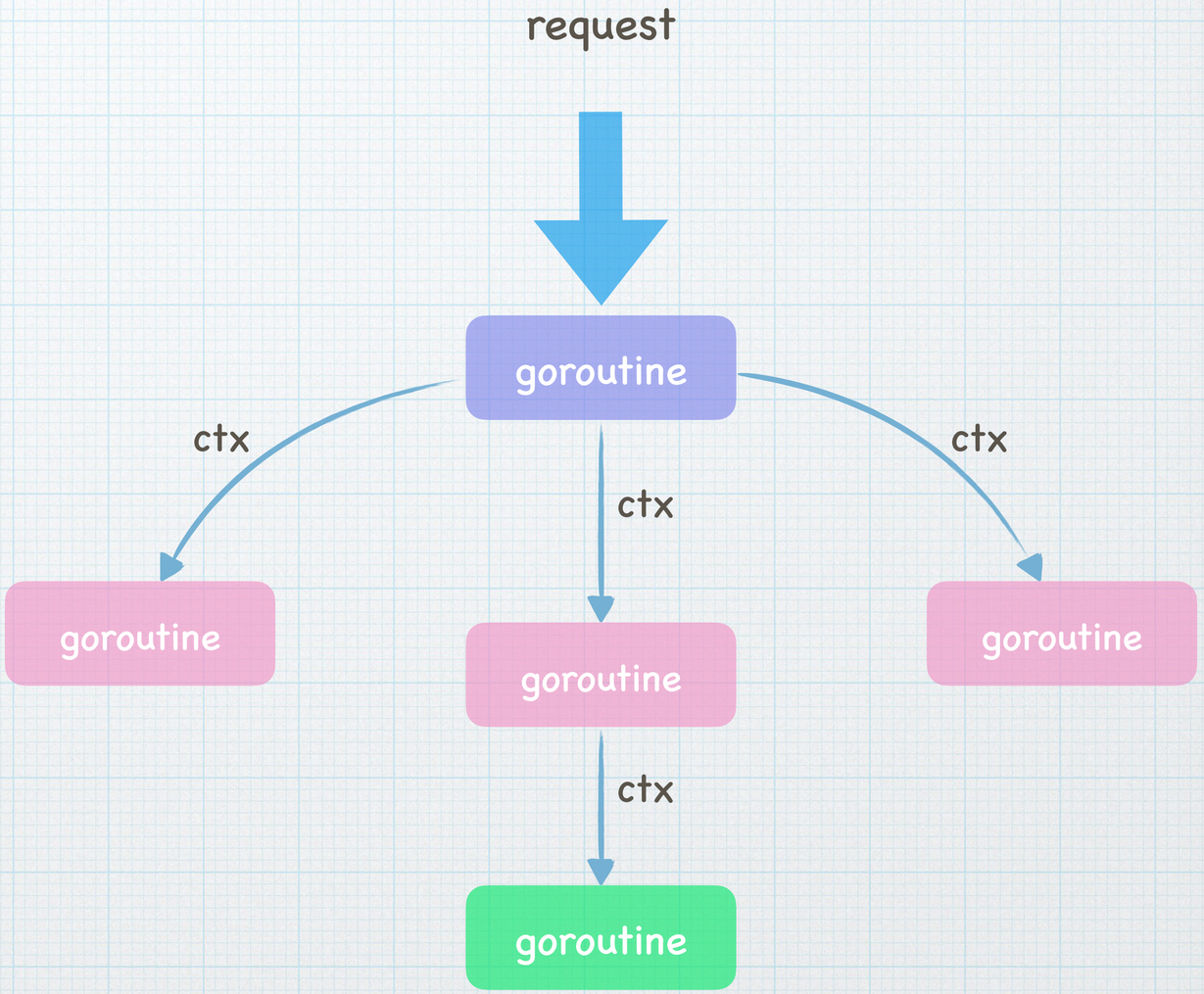
* context.Background()
  * context.WithCancel()
  * context.WithTimeout()
  * context.WithDeadline()
  * context.WithValue()



context

* ctx 并发安全
* 函数之间数据及信号传递
* 协程指令传递

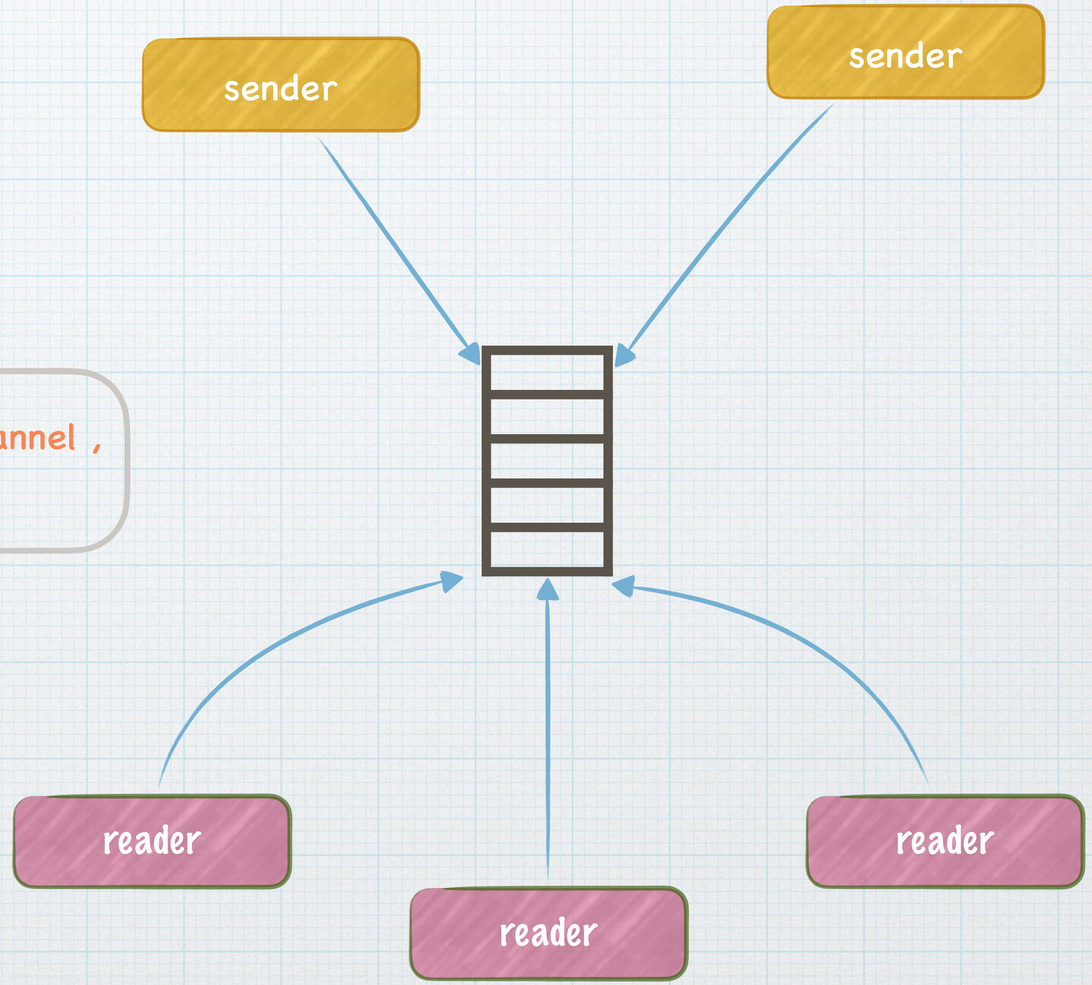




channel closing principle

* spsc
* spmc
* mpsc
* mpmc
  * context
  * stopChannel

在 多写多读 下不要直接关闭 数据channel，
而通过 ctx/chan 来通知退出.

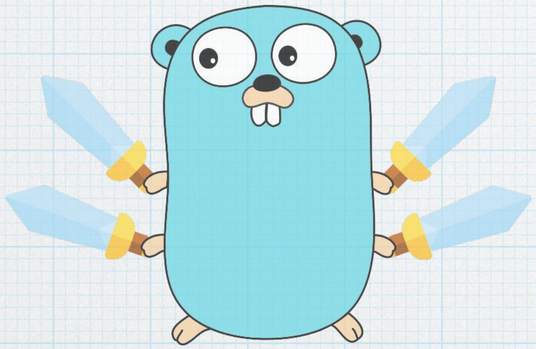


tips

- 如返回一个后面有竞争的容器结构，建议 deepcopy 返回
- 返回一个单向的 chan 对象
- 不能 copy 含有 lock 的结构
- 不能在 for 循环内执行 defer
- 尽量不要用 goto
- “for range map delete key” 安全操作
- 用来超时的timer，在函数退出时要 stop
- … … …

```go
// queue task queue
var queue chan interface{}

// Bad Code
func DequeueTimeout() (interface{}, error) {
    select {
    case <-time.NewTimer(10 * time.Second).C:
        return nil, errors.New("dequeue timeout")

    case task := <-queue:
        return task, nil
    }
}

// Good Code
func DequeueTimeout() (interface{}, error) {
    timeout := time.NewTimer(10 * time.Second)
    defer timeout.Stop()

    select {
    case <-timeout.C:
        return nil, errors.New("dequeue timeout")

    case task := <-queue:
        return task, nil
    }
}
```

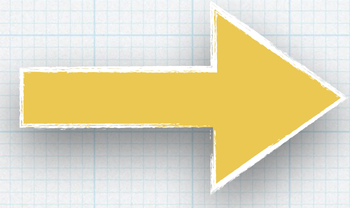
并发编程



- 同步原语
- 并发模型
- 性能调优
- 效率库包
- 经验

同步原语

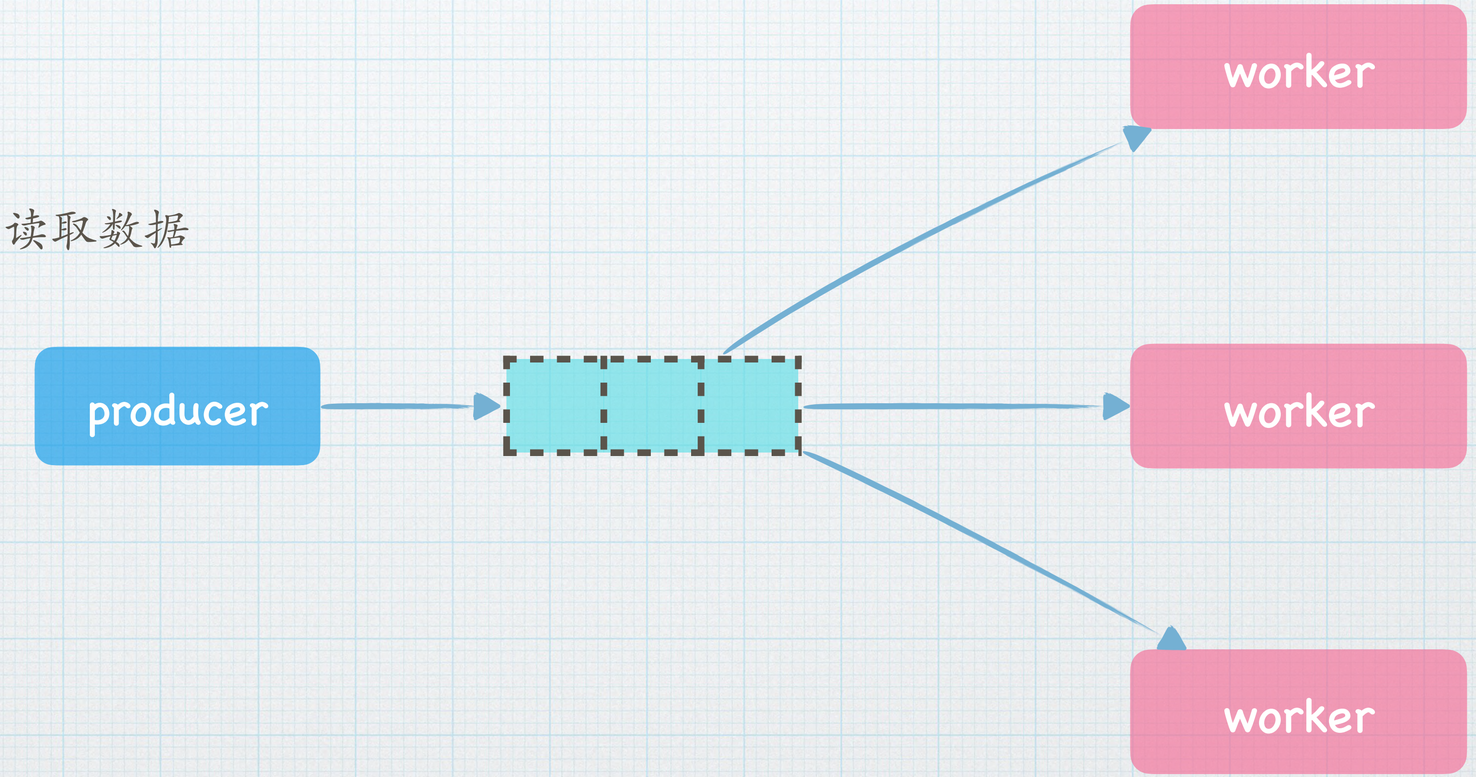
* Sync
  * mutex
  * rwmutex
  * once
  * cond
  * atomic



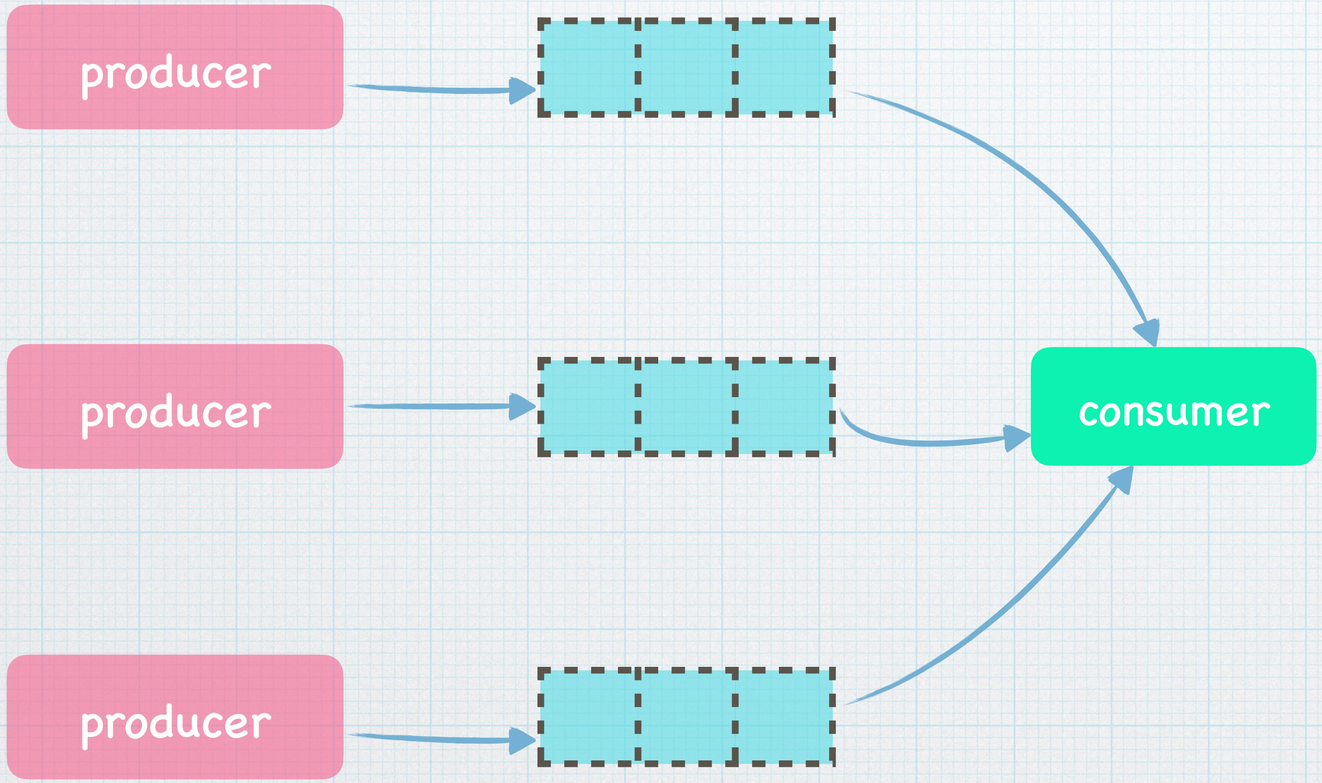
* Extend
  * ReentrantLock
  * Semaphore
  * SpinLock
  * Fslock
  * SingleFlight
  * NetLocker

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



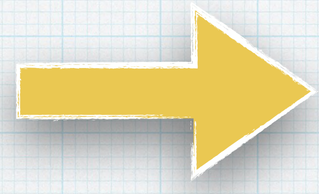
fan-in



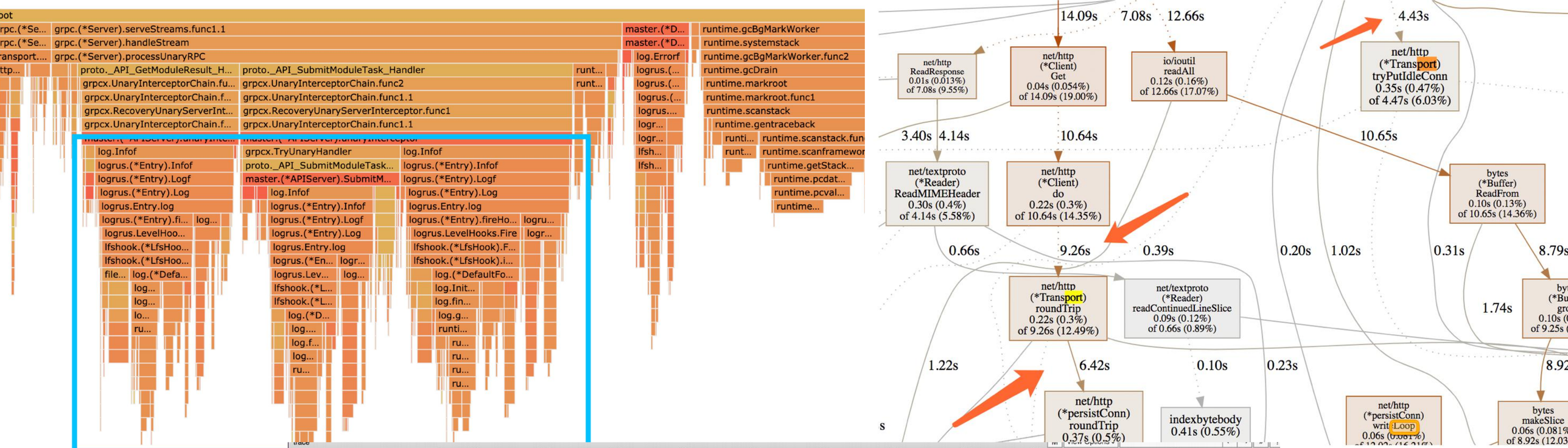
* FAN-IN模式
  * 1个goroutine从多个通道读取数据
  * 扇入

性能优化

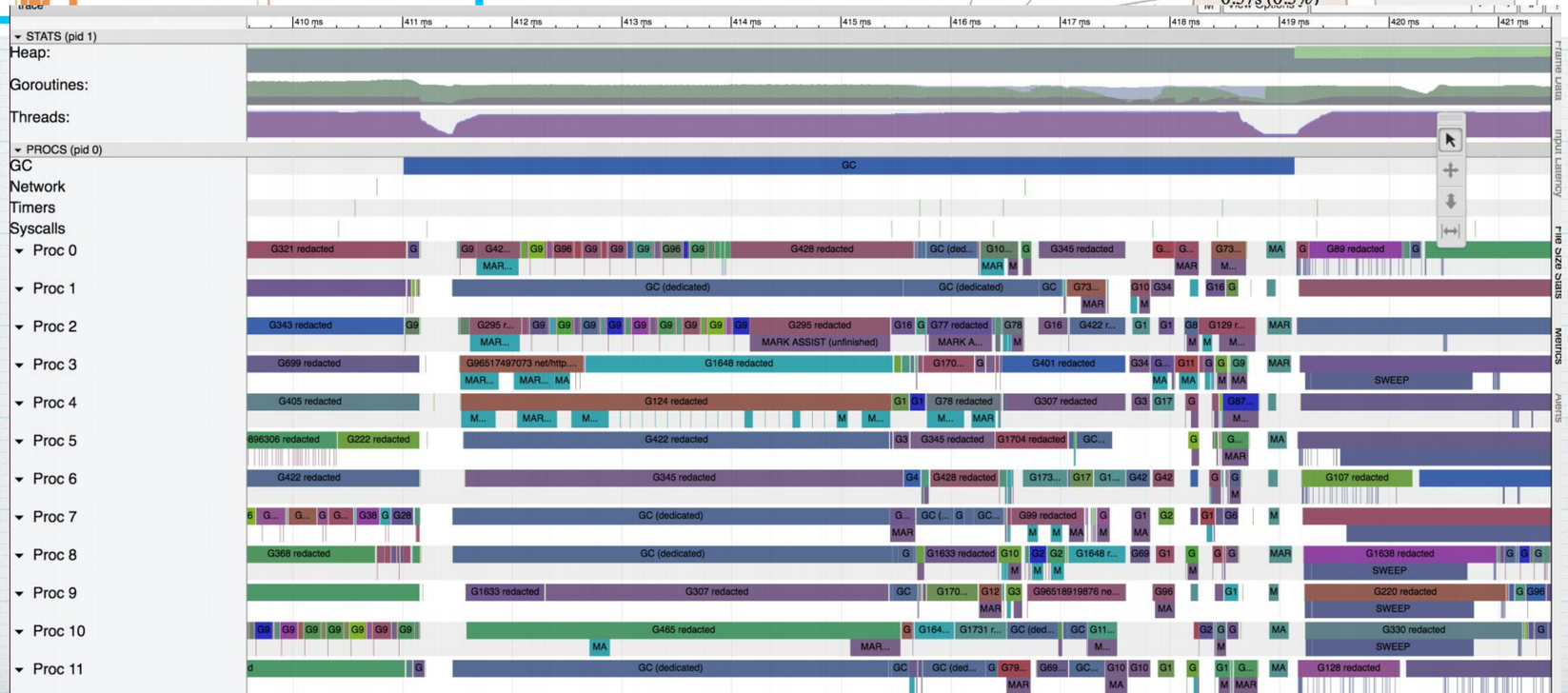
- pprof cpu on
- pprof cpu off
  - use fgprof
- pprof heap
- pprof trace



- 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

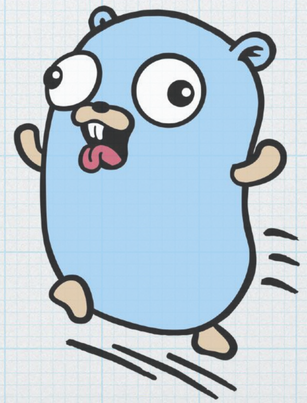

grpc.(*Server).serveStreams.func1.1
grpc.(*Server).handleStream
grpc.(*Server).processUnaryRPC
proto._API_GetModuleResult_H...
proto._API_SubmitModuleTask_Handler
grpcx.UnaryInterceptorChain.fu...
grpcx.UnaryInterceptorChain.func2
grpcx.UnaryInterceptorChain.f...
grpcx.UnaryInterceptorChain.func1.1
grpcx.RecoveryUnaryServerInt...
grpcx.RecoveryUnaryServerInterceptor.func1
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.(...
logrus....
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
4.43s
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%)
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
GC (dedicated)
MARK ASSIST (unfinished)
SWEEP

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
- ...



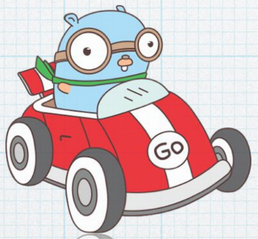
可靠性

- 限频策略
- 熔断策略
- 降级策略

- 容错机制
  - fallback 降级策略，一定要想好兜底 !!!
  - failfast 快速失败
  - failover 失败转义
  - backup requests 高延迟下的请求对冲
- 加入业务指标监控，关注时延，错误率和 QPS
- 尽量在所有请求及逻辑中加入可靠的超时逻辑，注意超时传递问题

效率库包

* ratelimiter
* circuit breaker
* cache
* gpool
* resty & req



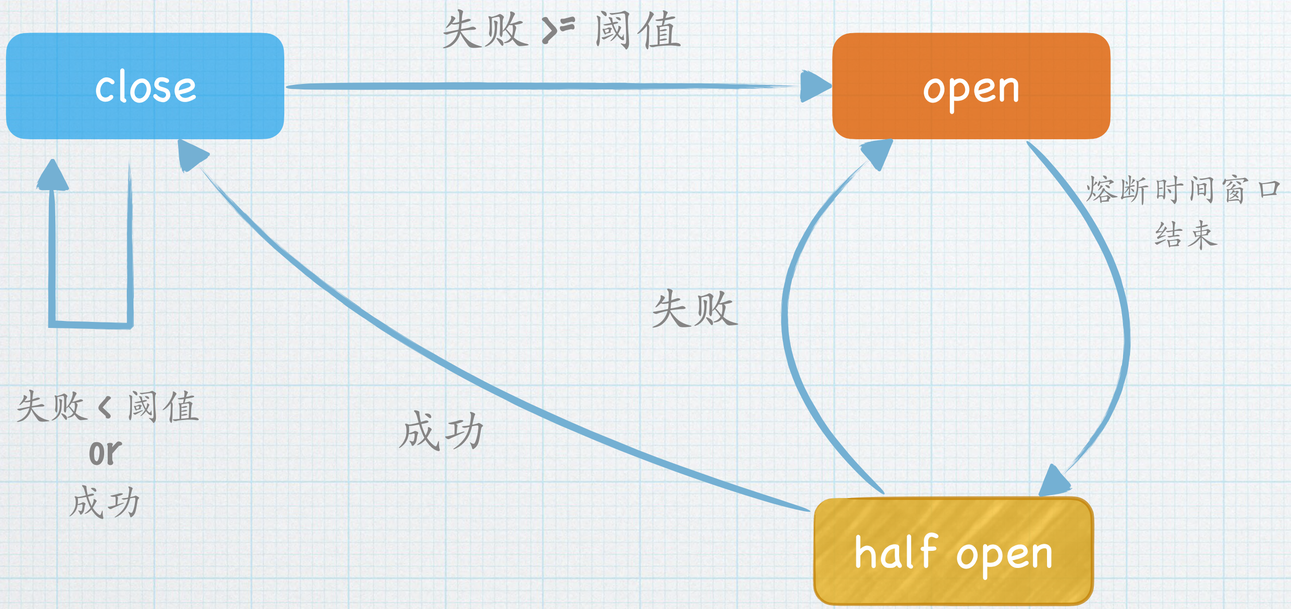
* backoff
* gjson
* gops
* cast
* retry
* ... ...

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

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))
}
```

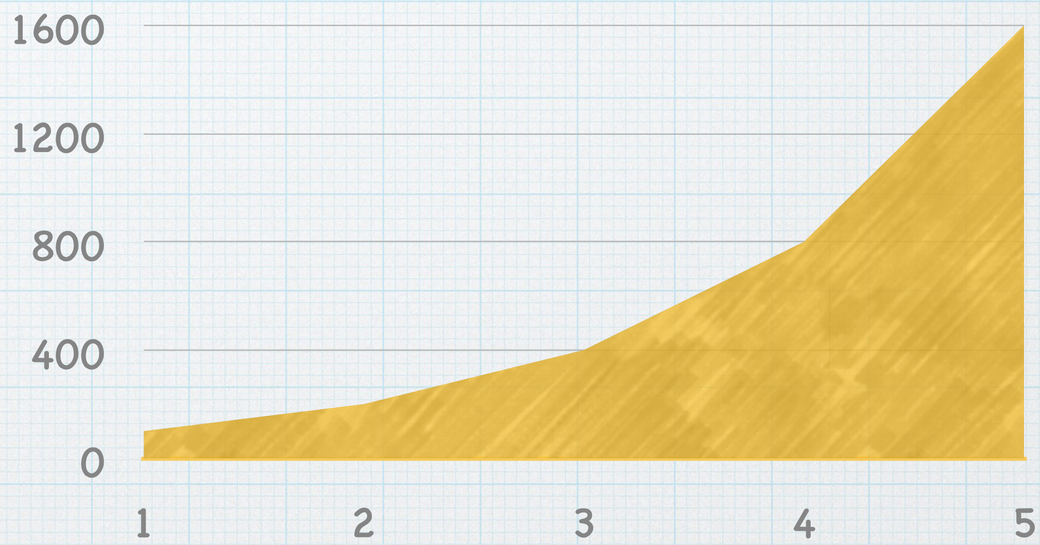
gpool 协程池

* 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()
}
```

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)
}
```



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)
    }
}
```

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


```

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

“Q&A!”

– 峰云就她了