

内建异步的 Go 语言

2011.10.10

关于“我”

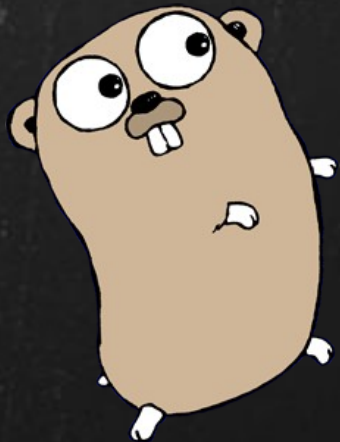
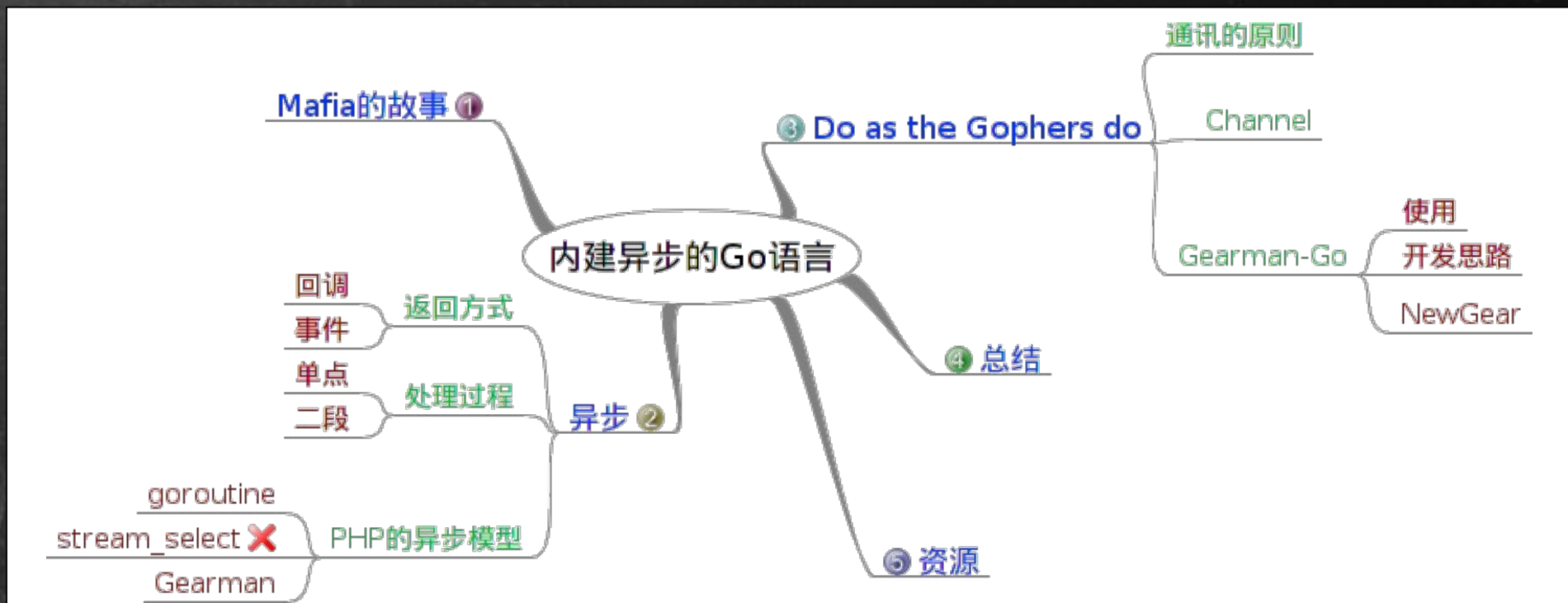
邢兴 39 健康网

mikespook@gmail.com

<http://mikespook.com>

@mikespook





主要内容

不会涉及的

- *Golang* 的基础知识
- 性能和基准测试
- *GC*、内存模型
- 实际应用案例
- *Golang* 的未来



Mafia 的故事

- 故事是这样开始的.....
 - 剧情 A——“大佬：到货场取阿一的货，送去给他。然后回来，再取阿二的货，送去给他.....”
 - 剧情 B——“大佬：到货场取上货，然后分别送去给阿一、阿二、阿三。这是地址.....”
 - 剧情 C——“大佬：到货场取上货，然后通知阿一、阿二、阿三，约定地方取货.....”



Mafia 的故事

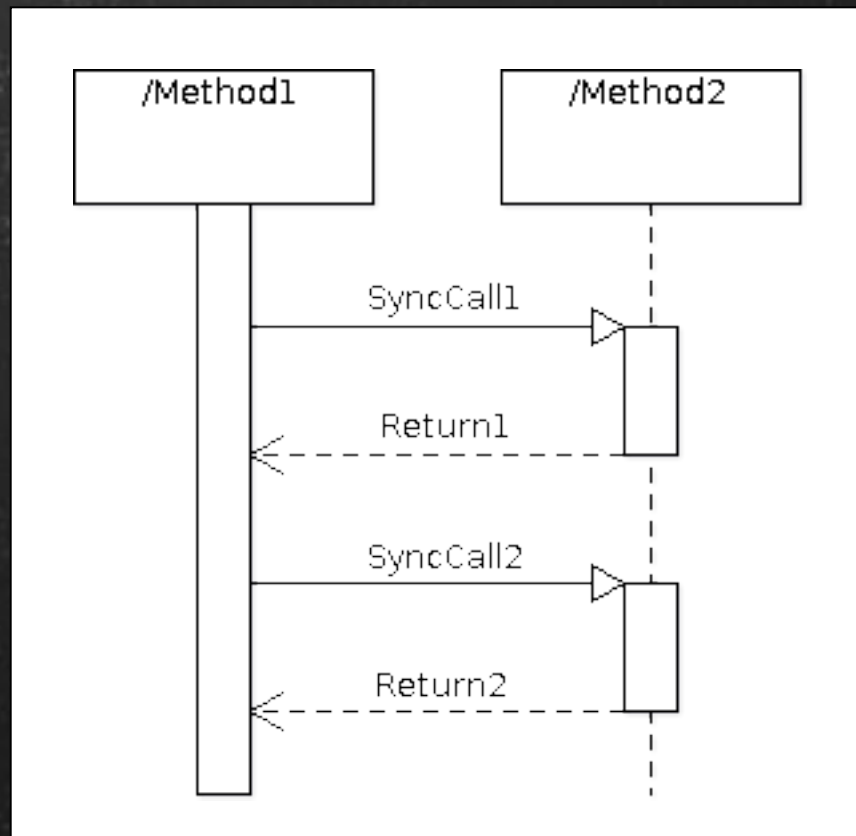
• 续.....

- 剧情 D——“大佬：到货场取上货，集中到咱们的仓库。全部货到齐了，一起送去给阿零.....”
- 剧情 E——“大佬：到货场取上货，集中到咱们的仓库。全部货到齐了，通知阿零到仓库取货.....”

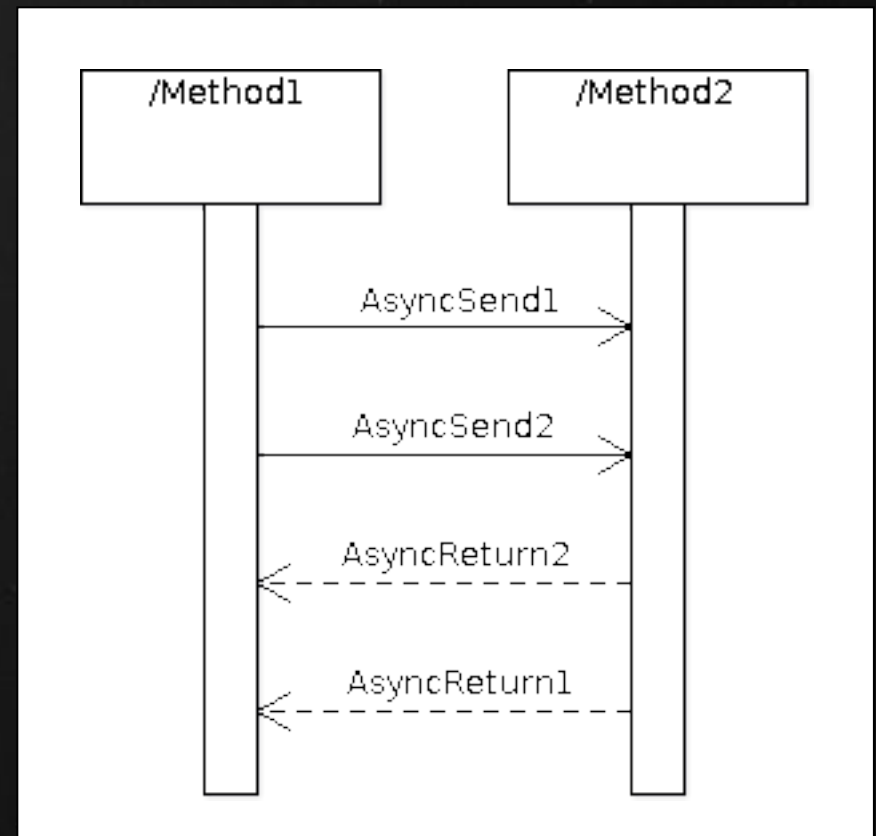


异步 == 非同步

同步



异步



异步

- 返回方式
 - 回调 (callback)
 - 事件 (event)
- 处理过程
 - 单点 (once)
 - 二段 (Begin/End)

- Mafia 的故事
 - 剧情 A—— 同步
 - 剧情 B—— 单点回调
 - 剧情 C—— 单点事件
 - 剧情 D—— 二段回调
 - 剧情 E—— 二段事件

	回调	事件
单点	剧情B	剧情C
二段	剧情D	剧情E

异步

```
2 // 获取数据
3 $userInfo = getUserInfo();
4 $newsList = getNewsList();
5 $topRateNewsList = getNewsList('DESC `rate`');
6 // 创建模板，绑定变量
7 $tmp = CreateTemplate();
8 $tmp->bind('userInfo', $userInfo);
9 $tmp->bind('newsList', $newsList);
10 $tmp->bind('topRateNewsList', $topRateNewsList);
11 // 渲染模板，输出
12 $tmp->render();
13 echo $tmp;
```

- PHP 的异步模型

- 白日梦：引入 go-lang 的关键字 go 到 php 中。
- 用原生 stream_select 实现
- RPC Queue——Gearman
- Libevent for PHP

Goroutine

- It is a function executing in parallel with other goroutines in the same address space.

-- From "Effective Go"

```
8 func Announce(message string, delay int64) {  
9     go func() {  
10         time.Sleep(delay)  
11         fmt.Println(message)  
12     }() // 注意括号，必须调用这个函数。  
13 }
```

Goroutine

```
15 func main() {  
16     Announce("Test 1", 10000000000)  
17     Announce("Test 2", 100000000)  
18     Announce("Test 3", 1000000000)  
19     time.Sleep(50000000000)  
20 }
```

```
mikespook@mikespook-laptop:~/Desktop$ 8g foobar.go  
mikespook@mikespook-laptop:~/Desktop$ 8l -o foobar foobar.8  
mikespook@mikespook-laptop:~/Desktop$ ./foobar  
Test 2  
Test 3  
Test 1  
mikespook@mikespook-laptop:~/Desktop$
```



- 是什么 (What) ?

Gearman provides a generic application framework to farm out work to other machines or processes that are better suited to do the work.

- 为什么 (Why) ?

It allows you to do work in parallel, to load balance processing, and to call functions between languages.

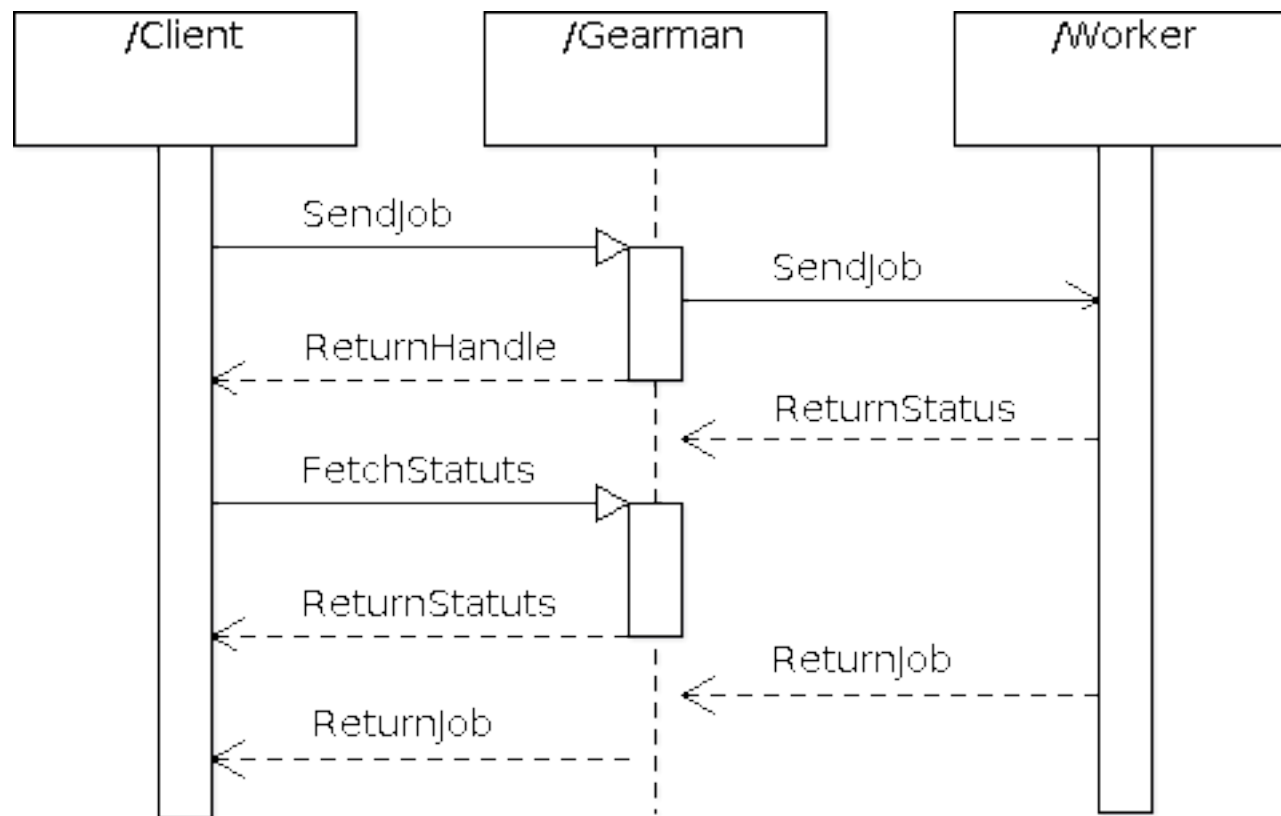
Gearman

- 如何做 (How) ?



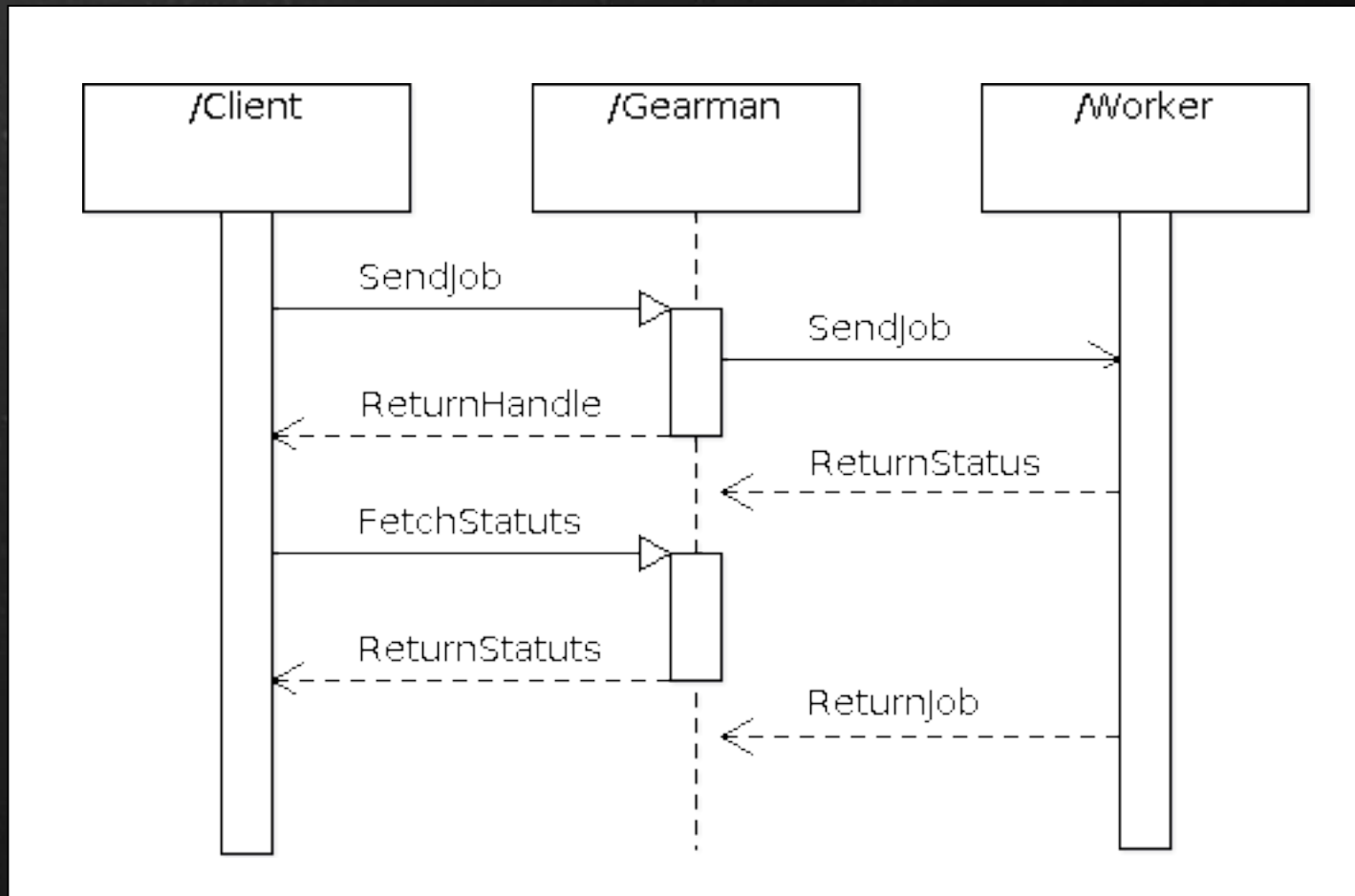
Gearman

- Normal Job



Gearman

- Background Job





When in Rome,
do as the Romans do.

When in Golang,
do as the Gophers do.



Do not communicate by
sharing memory; instead,
share memory by
communicating.

-- From "Effective Go"

Channel


- Channels combine communication—the exchange of a value—with synchronization—guaranteeing that two calculations (goroutines) are in a known state.
-- From "Effective Go"

Channel, PIPE or Message Queue?

- 信号量

```
9      var sem = make(chan int, MaxOutstanding)
10
11     func handle(r *Request) {
12         sem <- 1    // 等待队列腾出空间
13         process(r)  // 可能需要一些时间
14         <-sem       // 完成；让下一个请求可执行。
15     }
16
17     func Serve(queue chan *Request) {
18         for {
19             req := <-queue
20             go handle(req) // 不等待 handle 完成执行。
21         }
22     }
```

Gearman-Go

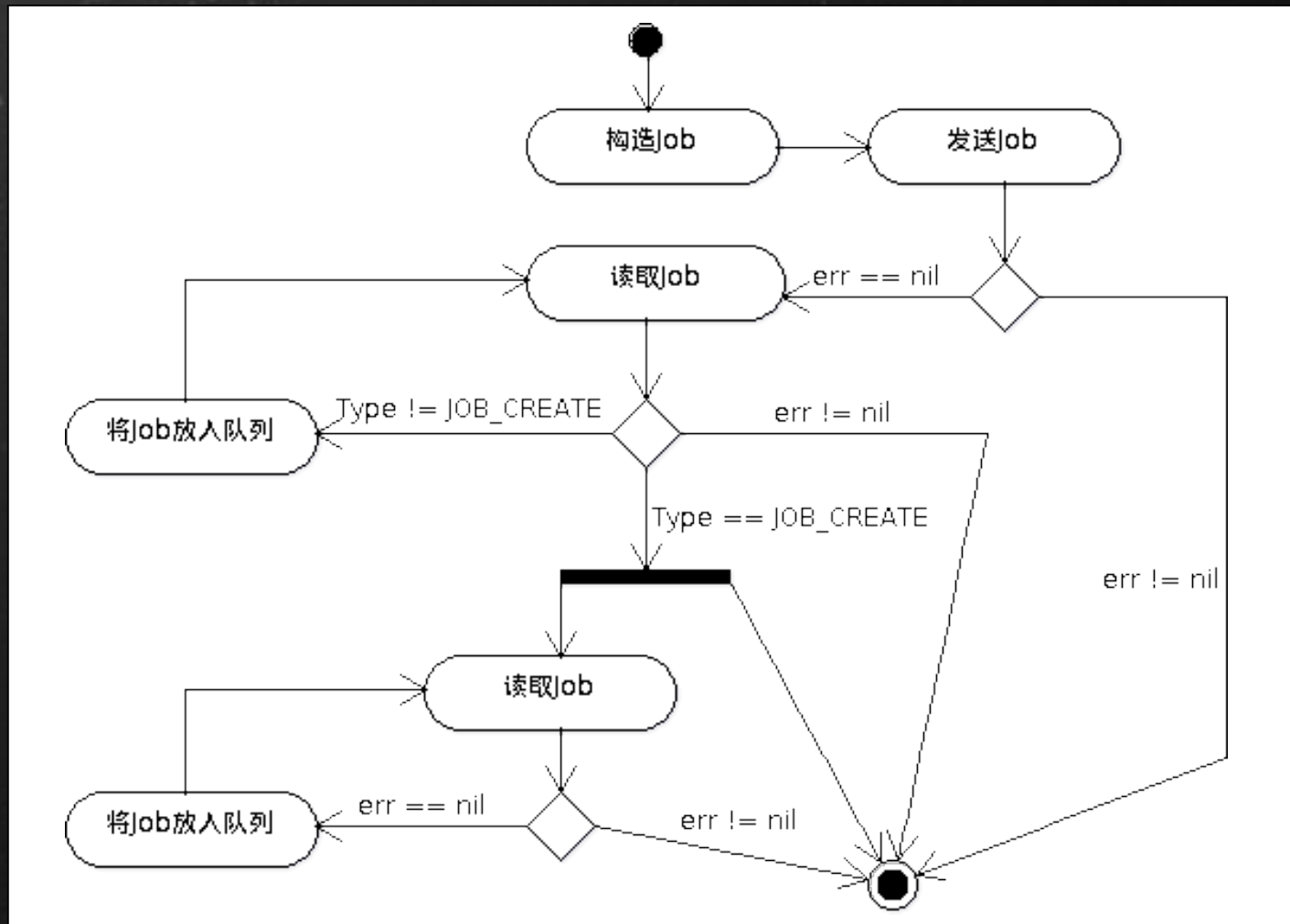
- 目的
 - 学习 Go 语言
- 目标
 - 一个可用的 Go 语言包，能替换现有的工作环境中的其他编程语言实现的 worker。
- 实现
 - cgo binding : cgo 无法处理 c callback。 
 - Go 语言的原生实现。

Gearman-Go

- 使用 client

```
9  client := gearman.NewClient()
10 defer client.Close()
11 if err := client.AddServer("127.0.0.1:4730"); err != nil {
12     log.Fatalln(err)
13 }
14 handle, err := client.Do("ToUpper", []byte("Hello\x00 world"), gearman.JOB_NORMAL)
15 if err != nil {
16     log.Fatalln(err)
17 } else {
18     log.Println(handle)
19     job := <-client.JobQueue
20     if data, err := job.Result(); err != nil {
21         log.Fatalln(err)
22     } else {
23         log.Println(string(data))
24     }
25 }
```

- client 实现



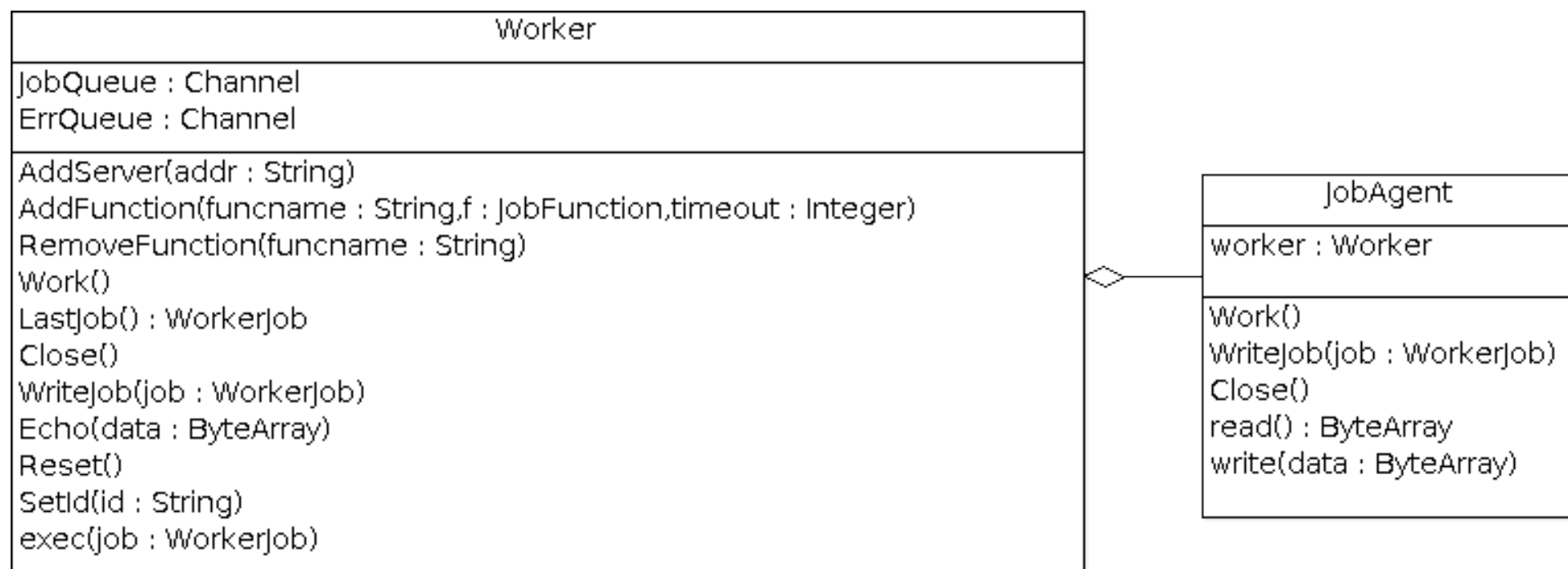
Gearman-Go

- 使用 worker

```
9 func ToUpper(job *gearman.WorkerJob) ([]byte, os.Error) {
10     data := []byte(strings.ToUpper(string(job.Data)))
11     return data, nil
12 }
13
14 func main() {
15     worker := gearman.NewWorker()
16     worker.AddServer("127.0.0.1:4730")
17     worker.AddFunction("ToUpper", ToUpper, 0)
18     worker.Work()
19 }
```

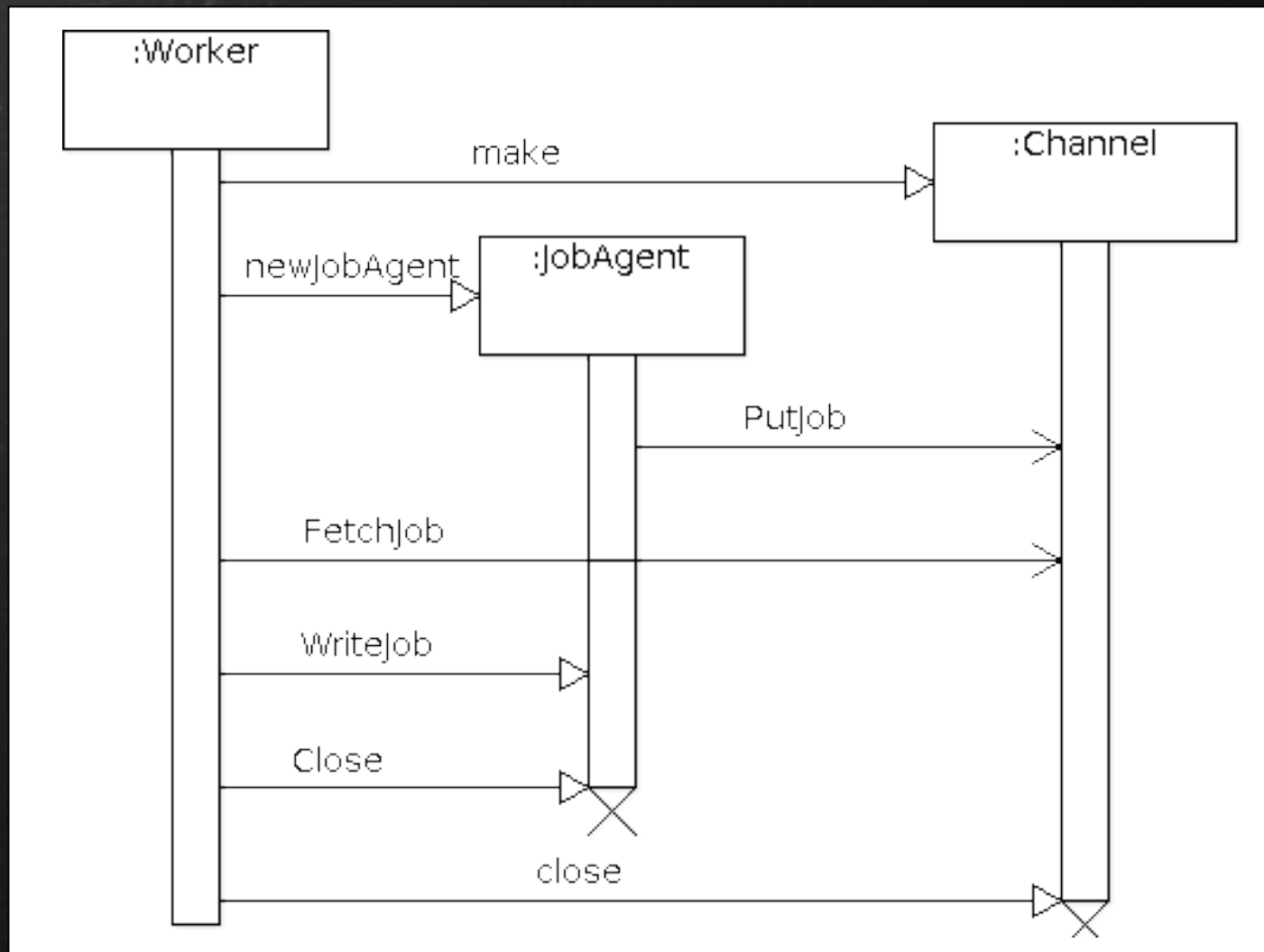
Gearman-Go

- worker 实现



Gearman-Go

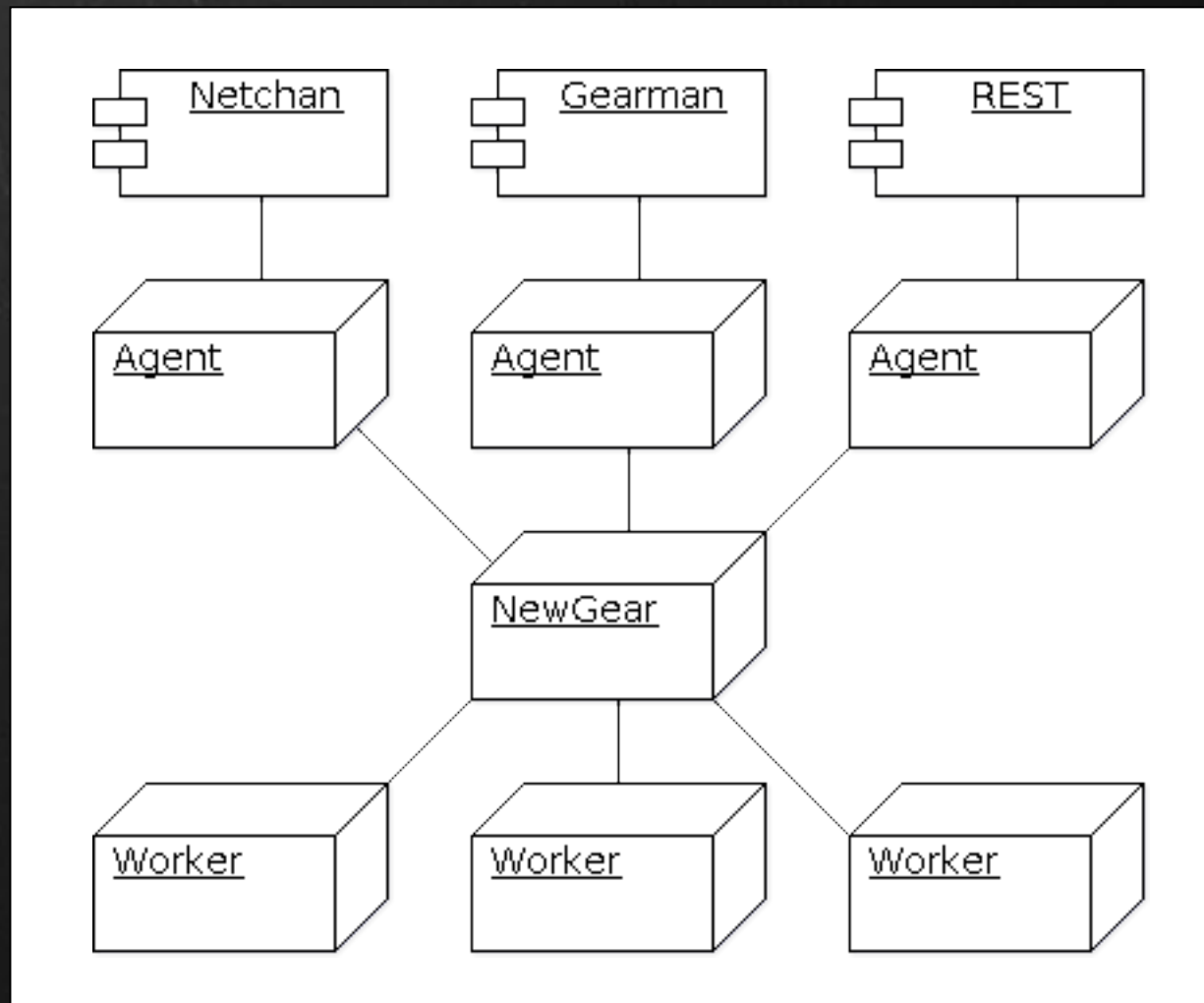
- worker 实现



- 目的
 - 实践 Go 语言开发思路，形成 Go 开发的基础框架工具集。
- 目标
 - 一个可用的 RPC Queue 组件集合，可替换现有的工作环境中的 Gearman，也可作为 worker 接入 Gearman。
- Golang 的原生实现

NewGear

- pkg: gob, netchan



可以通过 Go 语言内置的并发和通讯功能实现异步。
让开发者可以从架构层面解决问题的系统编程语言。

资源

1. Golang 官方网站

<http://golang.org/>

2. Effective Go

http://golang.org/doc/effective_go.html

3. Gearman 官方网站

<http://gearman.org/>

4. Gearman-Go 项目

<http://bitbucket.org/mikespook/gearman-go>

5. "Learning Go"

<http://miek.nl/cgi-bin/gitweb.cgi?p=gobook.git;a=summary>

6. 《学习 Go 语言》

<http://www.mikespook.com/learning-go/>

7. 《Go 指南》

<http://go-tour-zh.appspot.com>

THANK YOU!

mikespook@gmail.com