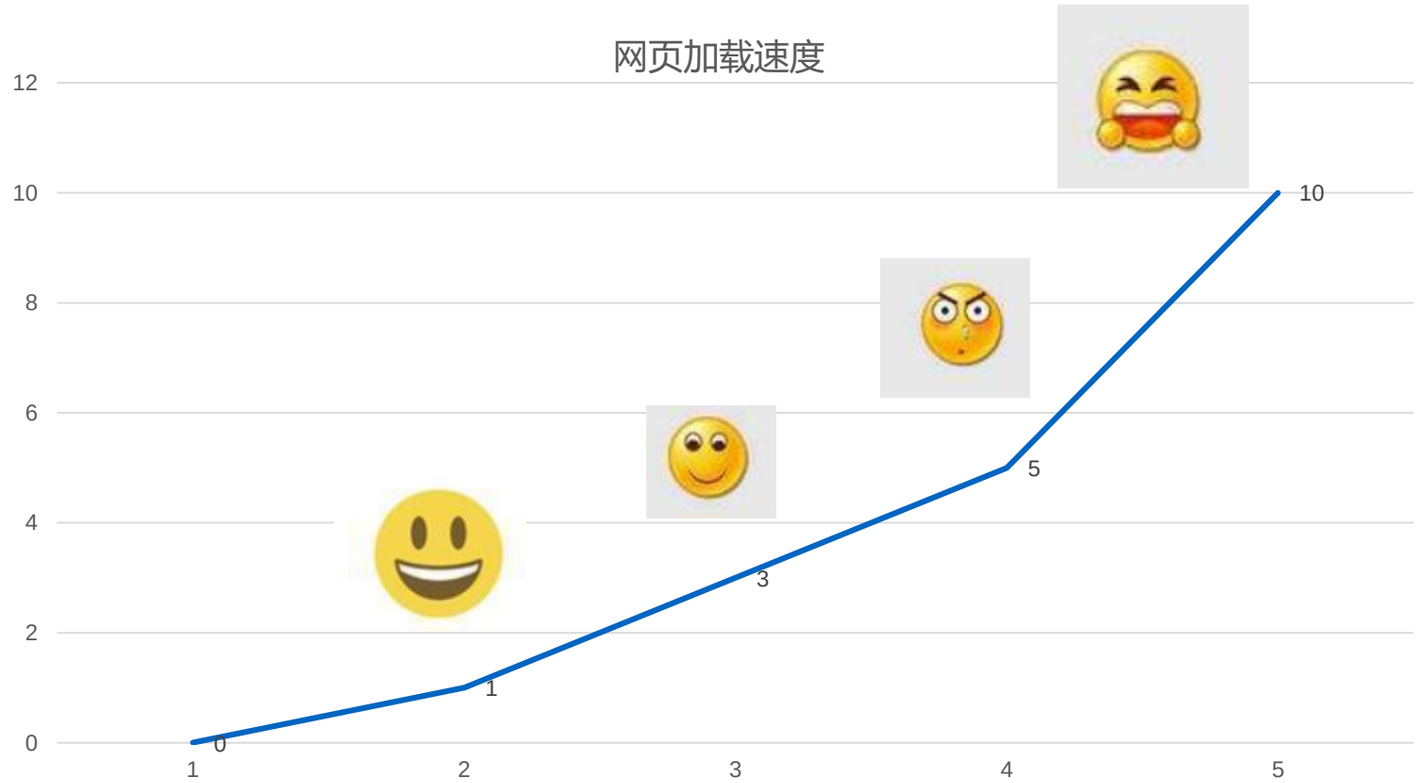
Go程序性能优化

作者：少林之巅

目录

性能优化背景以及原理

1. 性能优化背景



性能优化背景以及原理

2. 常见性能优化手段

A. 尽可能的减少 HTTP 的请求数。合并css和js以及图片。

B. 使用CDN系统，实现就近访问。

C. 启用gzip压缩，降低网页传输的大小。

D. 优化后端api服务的性能。

性能优化背景以及原理

3. Api服务性能优化的目标

A. 线上程序是一个黑盒

B. 通过性能优化，能够分析程序占用了多少资源?

C. 找到系统的瓶颈点

性能优化背景以及原理

4. Golang中的性能优化

A. Cpu维度的优化

B. Mem维度的优化

C. 锁竞争维度的优化

性能优化背景以及原理

5. 性能优化的原理

A. 要知道程序占用了多少资源，比如Cpu、内存等

B. 要知道程序的各个函数占用的资源比例

C. 有了A、B两个部分数据，我们就可以快速的定位到系统的性能瓶颈

D. 怎么达成这个目标?

当pprof开启后，每隔一段时间(10ms)收集下当前堆栈信息，获取各个函数占用的cpu以及内存资源；当pprof完成之后，通过对这些采样数据进行分析。形成一个性能分析报告。

CPU性能优化

6. CPU性能优化讲解

A. import (“runtime/pprof”)

B. 开始CPU性能分析： pprof.StartCPUProfile(w io.Writer)

C. 停止CPU性能分析: pprof.StopCPUProfile()

CPU性能优化

```
package main
import (
	"flag"
	"fmt"
	"os"
	"runtime/pprof"
	"time"
)
func logicCode() {
	var c chan int // = make(chan int, 1)
	for {
		select {
		case v := <-c:
			fmt.Printf("read from chan, v:%v\n", v)
		default:
		}
	}
}
func main() {
	var isCpuPprof bool
	flag.BoolVar(&isCpuPprof, "cpu", false, "turn cpu pprof on")
	flag.Parse()

	if isCpuPprof {
		file, err := os.Create("C:/tmp/cpu.pprof")
		if err != nil {
			fmt.Printf("create cpu pprof failed, err:%v\n", err)
			return
		}
		pprof.StartCPUProfile(file)
		defer pprof.StopCPUProfile()
	}

	for i := 0; i < 8; i++ {
		go logicCode()
	}
	time.Sleep(30 * time.Second)
}
```

CPU性能优化

7. CPU性能优化实战

A. 生成的采样数据，存储在当面目录下，./cpu.pprof

B. 使用命令进行分析, go tool pprof ./cpu_pprof_exam.exe ./cpu.pprof

C. 使用topN命令列出cpu消耗前几的函数，比如: top3

```
PS C:\project\src\github.com\pingguoxueyuan\gostudy\listen30\cpu_pprof_exam> go tool pprof .\cpu_pprof_exam.exe .\cpu.pprof
File: cpu_pprof_exam.exe
Type: cpu
Time: Sep 16, 2018 at 11:20pm (CST)
Duration: 3.12s, Total samples = 23.72s (759.86%)
Entering interactive mode (type "help" for commands, "o" for options)
(pprof) top3
Showing nodes accounting for 23.57s, 99.37% of 23.72s total
Dropped 24 nodes (cum <= 0.12s)
      flat  flat%   sum%        cum   cum%
    12.20s 51.43% 51.43%     20.31s 85.62%  runtime.selectnbrecv C:\Go\src\runtime\chan.go
     8.11s 34.19% 85.62%      8.11s 34.19%  runtime.chanrecv C:\Go\src\runtime\chan.go
     3.26s 13.74% 99.37%     23.57s 99.37%  main.logicCode c:\project\src\github.com\pingguoxueyuan\gostudy\listen30\cpu_pprof_exam\main.go
(pprof)
```

D. flat: 当前函数消耗的Cpu耗时， %flat： 当前函数消耗的Cpu耗时总占比。

E. sum%: 函数消耗的Cpu耗时的累计占比。

F. cum: 当前函数加上调用当前函数的函数消耗的Cpu耗时之和。 Cum%： 消耗的cpu总占比。

CPU性能优化

8. 图像化界面

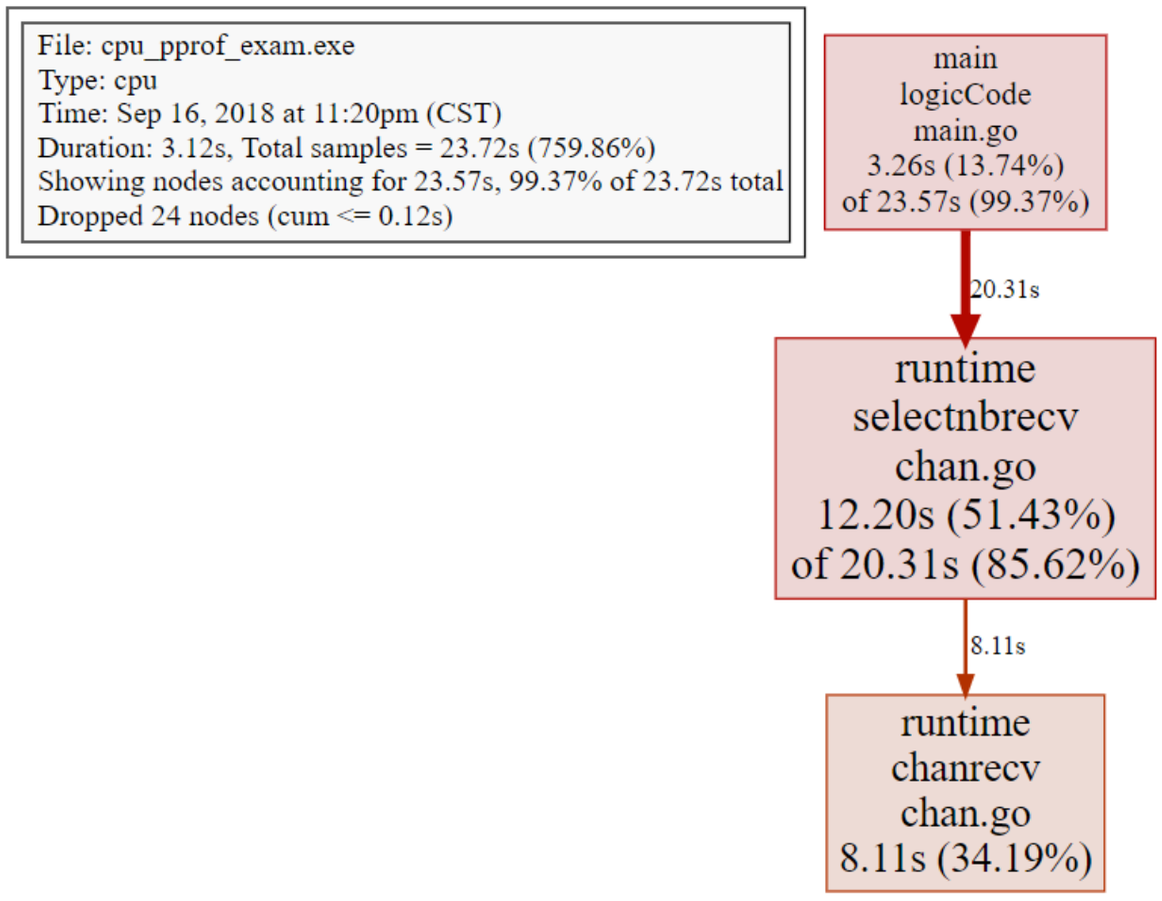
A. 安装graphviz图形化工具，下载地址

B. 安装完了之后，把安装路径加入到环境变量的PATH路径中。比如：C:\Program Files (x86)\Graphviz2.38\bin

C. go tool pprof ./cpu_pprof_exam.exe ./cpu.pprof，进入交互模式，敲入web。自动打开浏览器， 显示图像化界面。

CPU性能优化

9. 图像化界面



CPU性能优化

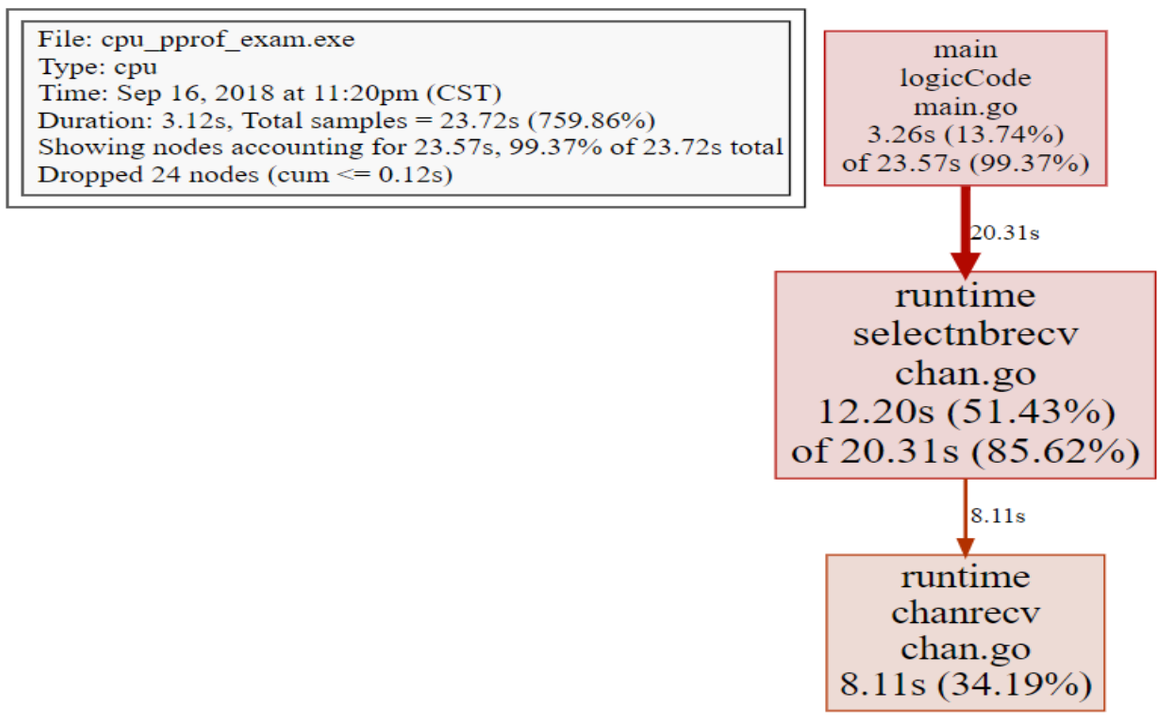
10. 图形说明

A. 每个框代表一个函数，理论上框的大小越大，占用的cpu资源越多。

B. 方框之间的线条，代表函数之间调用关系。

C. 线条上的数字，反映函数调用的次数。

D. 方框中的第一行数字，代表当前函数占用的cpu比例；第二行数字代表当前函数累计的比例。



CPU性能优化

11. Cpu pprof和单元测试结合

A. 每次都手动导入pprof比较麻烦，可以直接用go test压力测试时进行pprof。

B. 先编译压力测试程序，比如 go test –c 生成xxx_test可执行程序。

C. 分析cpu性能：./xxx.test –test.bench=xxxx –test.cpuprofile=./cpu.pprof。

D. 分析内存：./xxx.test –test.bench=xxxx –test.memprofile=./mem.pprof

MEM性能优化

11. 内存优化讲解

A. import (“runtime/pprof”)

B. 启用内存优化分析：pprof.WriteHeapProfile(w io.Writer)

C. 使用go tool pprof进行内存分析

tips go tool pprof默认是使用—inuse_space进行统计，可以使用—inuse-objects查看分配的数量。

MEM性能优化

12. 内存优化讲解练习

A. 代码来源：https://github.com/rsc/benchgraffiti/blob/master/havlak/havlak3.go

火焰图使用

13. 火焰图生成

A. 火焰图是Bredan Gregg创建一种性能分析图表。

B. 同样，go pprof的数据也可以转化成火焰图。

C. 使用Uber提供的： https://github.com/uber/go-torch 的工具。

火焰图使用

14. go-torch安装

A. go get github.com/uber/go-torch 。

B. git clone https://github.com/brendangregg/FlameGraph.git。

C. 把FlameGraph目录加入到操作系统的环境变量 PATH中。

D. 安装perl环境支持， http://www.perl.org/get.html

E. Windows的同学，需要把go-torch/render/flamegraph.go的GenerateFlameGraph**改成如下代码， 并在go-torch目录下执行go install就可以了：**

```
// GenerateFlameGraph runs the flamegraph script to generate a flame graph SVG.
func GenerateFlameGraph(graphInput []byte, args ...string) ([]byte, error) {
	flameGraph := findInPath(flameGraphScripts)
	if flameGraph == "" {
		return nil, errNoPerlScript
	}

	if runtime.GOOS == "windows" {
		return runScript("perl", append([]string{flameGraph}, args...), graphInput)
	}

	return runScript(flameGraph, args, graphInput)
}
```

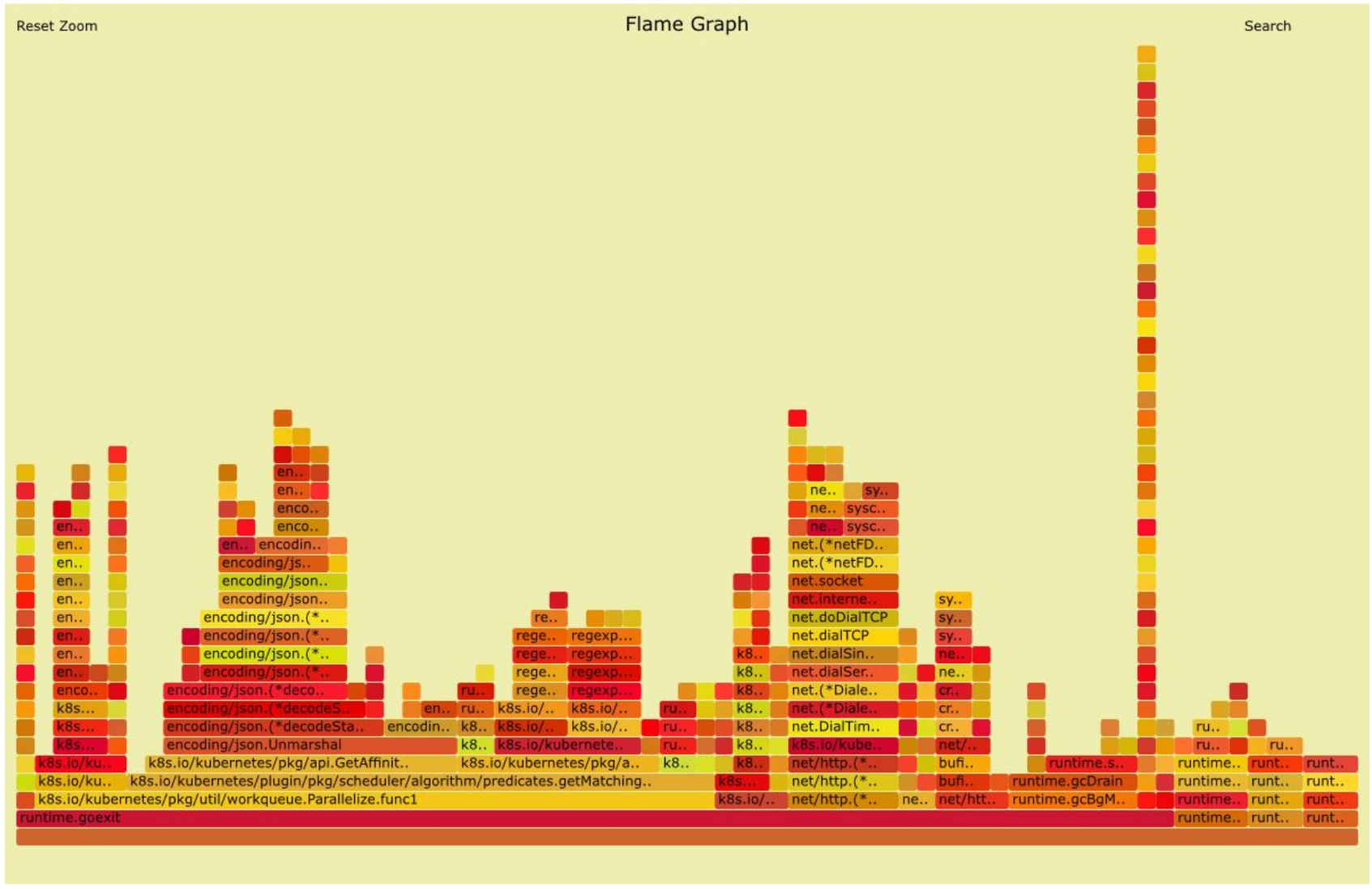
火焰图使用

15. go-torch使用

A. 把 go tool pprof 换成 go-torch就可以了。

火焰图使用

16. 火焰图举例



服务型程序pprof

17. 原声http以及tcp相关后台型程序pprof

A. 一直在后台运行，如果线上出现了性能问题怎么办?

B. Golang标准库：**import** _ "net/http/pprof"。

C. 打开url: http://localhost:xxx/debug/pprof/，就能看到pprof相关的信息。

服务型程序pprof

18. 基础使用

A. /debug/pprof/profile， 访问这个连接，会自动进行cpu pprof， 并把分析数据下载。

B. /debug/pprof/heap, 访问这个连接，会自动进行mem，pprof， 并把分析数据下载。

C. /debug/pprof/goroutines，访问这个连接，会把当前程序的goroutes信息打印出来。

D. /debug/pprof/threadcreate，访问这个连接，会把当前程序的使用的操作系统线程数量打印出来。

服务型程序pprof

19. 使用gin框架的后台型程序pprof

A. import (“github.com/DeanThompson/ginpprof”)

B. go get github.com/DeanThompson/ginpprof

C. 打开url: http://localhost:xxx/debug/pprof/，就能看到pprof相关的信息。

案例分析

20. 使用pprof优化 sudoku游戏

A. 游戏代码地址： https://github.com/paddie/godoku

B. 游戏介绍: https://baike.baidu.com/item/%E6%95%B0%E7%8B%AC/74847?fr=aladdin