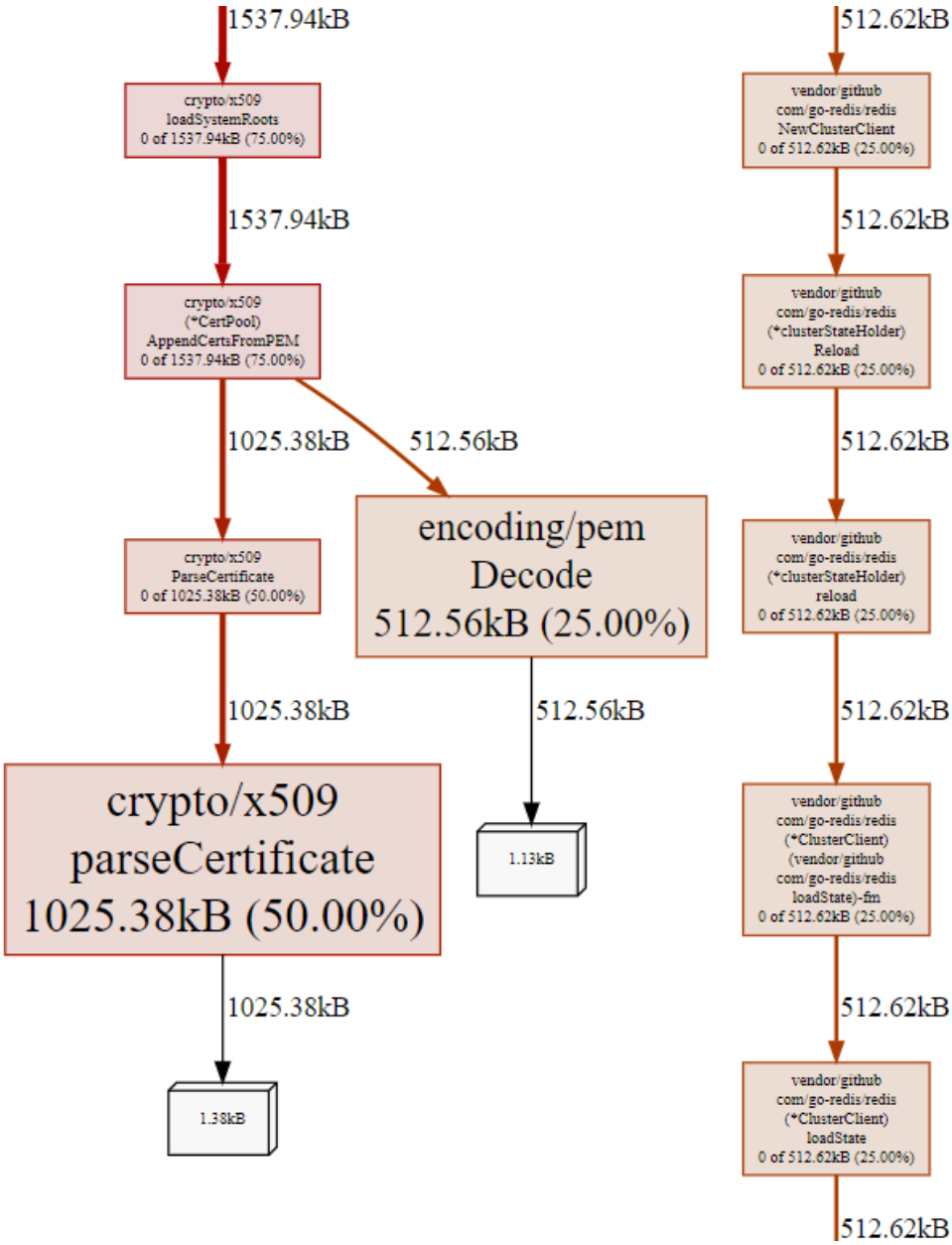
Go Profiling Using pprof

如果想要细致调优一个应用，不能靠直觉，要有详细的数据做支持，否则大概率瞎调就是浪费时间。获取运行时数据的一个重要途径就是做profiling。

Go应用里面常用的工具是pprof，pprof 是谷歌的一个可视化和分析C++ profiling数据的工具，go已经集成了pprof，在go中使用很简单。

达到的效果：

```
(pprof) top 10
Showing nodes accounting for 2050.56kB, 100% of 2050.56kB total
Showing top 10 nodes out of 36
      flat  flat%   sum%        cum   cum%
 1025.38kB 50.00% 50.00%  1025.38kB 50.00%  crypto/x509.parseCertificate
  512.62kB 25.00% 75.00%   512.62kB 25.00%  vendor/github.com/go-redis/redis/internal/pool.N
  512.56kB 25.00%   100%   512.56kB 25.00%  encoding/pem.Decode
         0     0%   100%  1537.94kB 75.00%  crypto/tls.(*Conn).Handshake
         0     0%   100%  1537.94kB 75.00%  crypto/tls.(*Conn).clientHandshake
         0     0%   100%  1537.94kB 75.00%  crypto/tls.(*clientHandshakeState).doFullHandsha
         0     0%   100%  1537.94kB 75.00%  crypto/tls.(*clientHandshakeState).handshake
         0     0%   100%  1537.94kB 75.00%  crypto/x509.(*CertPool).AppendCertsFromPEM
         0     0%   100%  1537.94kB 75.00%  crypto/x509.(*Certificate).Verify
         0     0%   100%  1025.38kB 50.00%  crypto/x509.ParseCertificate
(pprof) top5 -cum
Showing nodes accounting for 0, 0% of 2MB total
Showing top 5 nodes out of 36
      flat  flat%   sum%        cum   cum%
         0     0%     0%     1.50MB 75.00%  crypto/tls.(*Conn).Handshake
         0     0%     0%     1.50MB 75.00%  crypto/tls.(*Conn).clientHandshake
         0     0%     0%     1.50MB 75.00%  crypto/tls.(*clientHandshakeState).doFullHandsha
         0     0%     0%     1.50MB 75.00%  crypto/tls.(*clientHandshakeState).handshake
         0     0%     0%     1.50MB 75.00%  crypto/x509.(*CertPool).AppendCertsFromPEM
```



打开profiling

Benchmark

最简单的用法，pprof在testing包里已经默认打开了，做benchmark时可以直接加参数输出profile：

```
go test -cpuprofile cpu.prof -memprofile mem.prof -bench .
```

做完benchmark会输出cpu和mem profile。

在做一些热点的benchmark的时候通过做profile可以大幅度优化整体程序。

一般应用

触发CPU profiling：

```
f, err := os.Create(*cpuprofile)
if err != nil {
    log.Fatal("could not create CPU profile: ", err)
}
defer f.Close()
if err := pprof.StartCPUProfile(f); err != nil {
    log.Fatal("could not start CPU profile: ", err)
}
// 这个例子在main里面用defer，意思是到进程结束的时候才结束
// 自己实现时可以根据自己的应用选择结束profiling的时机
// 例如定时10s后结束
defer pprof.StopCPUProfile()
```

触发内存profiling，很类似上面CPU profiling，但是注意区别：

```
f, err := os.Create(*memprofile)
if err != nil {
    log.Fatal("could not create memory profile: ", err)
}
defer f.Close()
runtime.GC() // 保证数据实时性
// 注意，和CPU profiling不同，
// 这里是获取当前瞬间heap 状态，CPU profiling需要跑一段时间
if err := pprof.WriteHeapProfile(f); err != nil {
    log.Fatal("could not write memory profile: ", err)
}
```

除了这两种最常用的profile类型，还有

类型	说明
goroutine	所有当前瞬间的goroutines的stack trace
heap	当前存活的对象的内存分配样本
allocs	所有过去内存分配的样本（新版才支持）
threadcreate	引起新建系统线程的stack trace
block	引起阻塞的同步原语的stack trace
mutex	互斥锁的所有者stack traces

详细的pprof 接口见 https://golang.org/pkg/runtime/pprof/ 。

Web 应用

对于一些远程部署的web 应用，有更方便的打开profiling的方法，直接使用标准库中内置的内置了pprof操作的http handler即可。

最方便的用法是直接引入这个包

```
import _ "net/http/pprof"
```

作为副作用，会注册 /debug/pprof/的路由到你的http server。

如果要改变这个路由，可以使用它里面的http handler自己去注册，例如我们使用的beego：

```
nsAdm := beego.NewNamespace("/adm",
    // 对外环境注意加鉴权
    beego.NSBefore(admCtrl.Auth),
    beego.NSHandler("/pprof/heap", pprof.Handler("heap")),
    beego.NSHandler("/pprof/goroutine", pprof.Handler("goroutine")),
)
```

CPU profile没有http handler接口，我们自己套一个即可：

```
type cpuProfHandler string
func (h cpuProfHandler) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    pprof.Profile(w, r)
}

//...
// 注册到NS
beego.NSHandler("/pprof/cpu", new(cpuProfHandler))
```

这个CPU profile默认跑30s，如果需要更改，访问时URL带上query：`?seconds=5`

使用pprof 分析

用以上任意方式得到profile之后可以用pprof分析，Go内置了命令行工具：

```
> go tool pprof cpu.prof
File: thesis_styling
Build ID: 037fddd4a2d21018acf9e46d847197edfd5bf9e3
Type: cpu
Time: Mar 27, 2019 at 9:27am (CST)
Duration: 10.01s, Total samples = 0
Entering interactive mode (type "help" for commands, "o" for options)
(pprof)
```

接下来就进入了交互模式，你可以敲命令，常用的命令：

- top 10

```
(pprof) top 10
Showing nodes accounting for 591.75kB, 100% of 591.75kB total
Showing top 10 nodes out of 11
      flat  flat%   sum%        cum   cum%
  591.75kB   100%   100%   591.75kB   100%  crypto/elliptic.initTable
         0     0%   100%   591.75kB   100%  crypto/elliptic.(*p256Point
         0     0%   100%   591.75kB   100%  crypto/elliptic.GenerateKey
         0     0%   100%   591.75kB   100%  crypto/elliptic.p256Curve.S
         0     0%   100%   591.75kB   100%  crypto/tls.(*Conn).Handshak
         0     0%   100%   591.75kB   100%  crypto/tls.(*Conn).clientHa
         0     0%   100%   591.75kB   100%  crypto/tls.(*clientHandshak
```

- top 5 -cum（累计）

```
         0     0%   100%   591.75kB   100%  net/http.(*persistCo
(pprof) top 5 -cum
Showing nodes accounting for 591.75kB, 100% of 591.75kB total
Showing top 5 nodes out of 11
      flat  flat%   sum%        cum   cum%
         0     0%     0%   591.75kB   100%  crypto/elliptic.(*p2
         0     0%     0%   591.75kB   100%  crypto/elliptic.Gene
  591.75kB   100%   100%   591.75kB   100%  crypto/elliptic.init
         0     0%   100%   591.75kB   100%  crypto/elliptic.p256
         0     0%   100%   591.75kB   100%  crypto/tls.(*Conn).H
```

- web（用svg生成一个网状的profile数据并在浏览器打开）
- list {symbol}（列出具体分配内存的地方）

```
(pprof) list elliptic.initTable
Total: 591.75kB
ROUTINE ======================== crypto/elliptic.initTable in /usr/local/go/src
  591.75kB   591.75kB (flat, cum)   100% of Total
         .          .    433:   d = (d >> 1) + (d & 1)
         .          .    434:   return int(d), int(s & 1)
         .          .    435:}
         .          .    436:
         .          .    437:func initTable() {
  591.75kB   591.75kB    438:   p256Precomputed = new([37][64 * 8]uint64)
         .          .    439:
         .          .    440:   basePoint := []uint64{
         .          .    441:           0x79e730d418a9143c, 0x75ba95fc5fedb601,
         .          .    442:           0xddf25357ce95560a, 0x8b4ab8e4ba19e45c,
         .          .    443:           0x0000000000000001, 0xffffffff00000000,
```

其它的交互命令可以通过help 来学习。

另外由于我是在没有GUI的环境中，不能直接用web命令生成SVG并打开，怎么办？可以直接在命令行生成：

```
go tool pprof -svg profile > heap.svg
```

`-svg`也可以换成其它格式，例如png。在有GUI的地方把svg拖到浏览器，或用图片浏览器就好，就可以看到示例中很直观的网状图。

注意，生成图片是go调用了gv生成了xdot文件并转换，所以需要安装Graphviz，不同的系统安装方式不同，不再赘述。

参考

- go doc - net/http/pprof: https://golang.org/pkg/net/http/pprof
- go doc - runtime/pprof: https://golang.org/pkg/runtime/pprof/
- go blog: https://blog.golang.org/2011/06/profiling-go-programs.html