# API 性能分析

作为开发，我们一般都局限在功能上的单元测试，对一些性能上的细节我们往往没有去关注，如果我们在上线的时候对项目整体性能没有一个全面的了解的话，当流量越来越大时，可能会出现各种各样的问题，比如 CPU 占用高、内存使用率高等。为了避免这些性能瓶颈，我们在开发的过程中需要通过一定的手段来对程序进行性能分析。

Go 语言已经为开发者内置配套了很多性能调优监控的好工具和方法，这大大提升了我们 profile 分析的效率，借助这些工具我们可以很方便地来对 Go 程序进行性能分析。在 Go 语言开发中，通常借助于内置的 pprof 工具包来进行性能分析。

## 本节核心内容

- 如何用 pprof 工具对 API 程序进行性能分析

> 本小节源码下载路径：[demo16 (https://github.com/lexkong/apiserver_demos/tree/mas](https://github.com/lexkong/apiserver_demos/tree/mas)
>
> 可先下载源码到本地，结合源码理解后续内容，边学边练。
>
> 本小节的代码是基于 [demo15 (https://github.com/lexkong/apiserver_demos/tree/mas](https://github.com/lexkong/apiserver_demos/tree/mas) 来开发的。

## pprof 是什么

PProf (https://github.com/google/pprof) 是一个 Go 程序性能分析工具，可以分析 CPU、内存等性能。Go 在语言层面上集成了 profile 采样工具，只需在代码中简单地引入 runtime/ppro 或者 net/http/pprof 包即可获取程序的 profile 文件，并通过该文件来进行性能分析。

> runtime/pprof 还可以为控制台程序或者测试程序产生 pprof 数据。
>
> 其实 net/http/pprof 中只是使用 runtime/pprof 包来进行封装了一下，并在 HTTP 端口上暴露出来。

# 使用 pprof

在 gin 中使用 pprof 功能，需要用到 github.com/gin-contrib/pprof middleware，使用时只需要调用 pprof.Register() 函数即可。本例中，通过在router/router.go中添加如下代码来实现（详见 demo16/router/router.go (https://github.com/lexkong/apiserver_demos/blob/master/

```
package router

import (
    "github.com/gin-contrib/pprof"
    ....
)

// Load loads the middlewares, routes, handlers.
func Load(g *gin.Engine, mw ...gin.HandlerFunc)
*gin.Engine {
    // pprof router
    pprof.Register(g)
    ....
}
```

## 编译

1. 下载 apiserver_demos 源码包（如前面已经下载过，请忽略此步骤）

```
$ git clone
https://github.com/lexkong/apiserver_demos
```

2. 将 apiserver_demos/demo16 复制为 $GOPATH/src/apiserver

```
$ cp -a apiserver_demos/demo16/
$GOPATH/src/apiserver
```

3. 在 apiserver 目录下编译源码

```
$ cd $GOPATH/src/apiserver
$ make
```

# 获取 profile 采集信息

通过 `go tool pprof http://127.0.0.1:8080/debug/pprof/profile`，可以获取 profile 采集信息并分析。

也可以直接在浏览器访问 `http://localhost:8080/debug/pprof` 来查看当前 API 服务的状态，包括 CPU 占用情况和内存使用情况等。

> 执行命令后，需要等待 30s，pprof 会进行采样。

# 性能分析

在上一小节我们介绍函数性能测试时已经介绍过性能分析的一部分知识，为了使内容完整，我们这里再次介绍下相关知识。

通过上一部分我们已经获取到了程序的 profile 信息，并且进入到了 pprof 的交互界面，在交互界面执行 topN 可以获取采样信息。

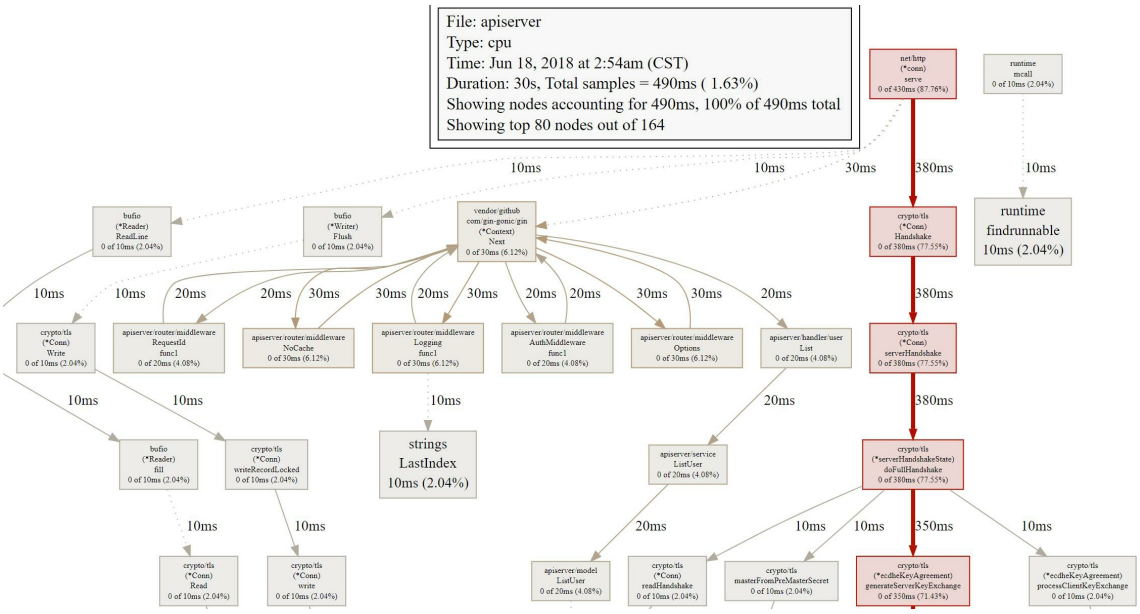
```
[api@centos src]$ go tool pprof http://127.0.0.1:8080/debug/pprof/profile
Fetching profile over HTTP from http://127.0.0.1:8080/debug/pprof/profile
Saved profile in /home/api/pprof/pprof.apiserver.samples.cpu.011.pb.gz
File: apiserver
Type: cpu
Time: Jun 18, 2018 at 2:54am (CST)
Duration: 30s, Total samples = 490ms ( 1.63%)
Entering interactive mode (type "help" for commands, "o" for options)
(pprof) top10
Showing nodes accounting for 350ms, 71.43% of 490ms total
Showing top 10 nodes out of 164
      flat  flat%   sum%        cum   cum%
      90ms 18.37% 18.37%      220ms 44.90%  math/big.nat.montgomery
      80ms 16.33% 34.69%       80ms 16.33%  math/big.addMulVVW
      50ms 10.20% 44.90%       50ms 10.20%  runtime.memmove
      40ms  8.16% 53.06%      110ms 22.45%  math/big.nat.divLarge
      20ms  4.08% 57.14%       30ms  6.12%  math/big.getNat
      20ms  4.08% 61.22%       20ms  4.08%  math/big.mulAddVWW
      20ms  4.08% 65.31%       20ms  4.08%  runtime.futex
      10ms  2.04% 67.35%       10ms  2.04%  crypto/sha256.block
      10ms  2.04% 69.39%       10ms  2.04%  internal/poll.convertErr
      10ms  2.04% 71.43%      120ms 24.49%  math/big.nat.div
```

通过 topN 的输出可以分析出哪些函数占用 CPU 时间片最多，这些函数可能存在性能问题。性能分析详细防范请参考：

如果觉得不直观，可以直接生成函数调用图，通过调用图来判断哪些函数耗时最久，在 pprof 交互界面，执行 svg 生成 svg 文件。

```
(pprof) svg
Generating report in profile001.svg
```

用浏览器打开 profile001.svg：




框框最大的函数耗时比较久，说明函数可能存在性能问题。

> 确保系统已经安装 graphviz 命令。

# 小结

本节展示了如何对 API 服务进行性能分析，这里只是介绍了如何添加性能分析入口和基本的流程。