大型 Go 系统生产环境问题定位

×

Xargin

曾经是某厂技术专家

目 录

第一部分

那些令人头疼的生产环境问题

“

你的老板

为什么有几个实例半夜四点的时候有一个 CPU 尖刺？

“

你的队友

为什么我的新代码上线了以后 OOM 了，还是随机的？

“

你的同事

你好，客户这里上线以后在线上 goroutine 数有暴涨，怎么办？

“

底层团队

你们的系统部署到新的容器实现里线程数炸了，解释一下？

第二部分

业界有没有可以借鉴的思路

业界先驱给我们的思路

Google 2010 的老论文：
https://research.google/pubs/pub36575/
网友的科普：https://www.opsian.com/blog/what-is-continuous-profiling/GCE上的付费服务：https://medium.com/google-cloud/continuous-profiling-of-go-programs-96d4416af77b
付费服务的客户端代码：https://github.com/googleapis/google-cloud-go/tree/master/profiler

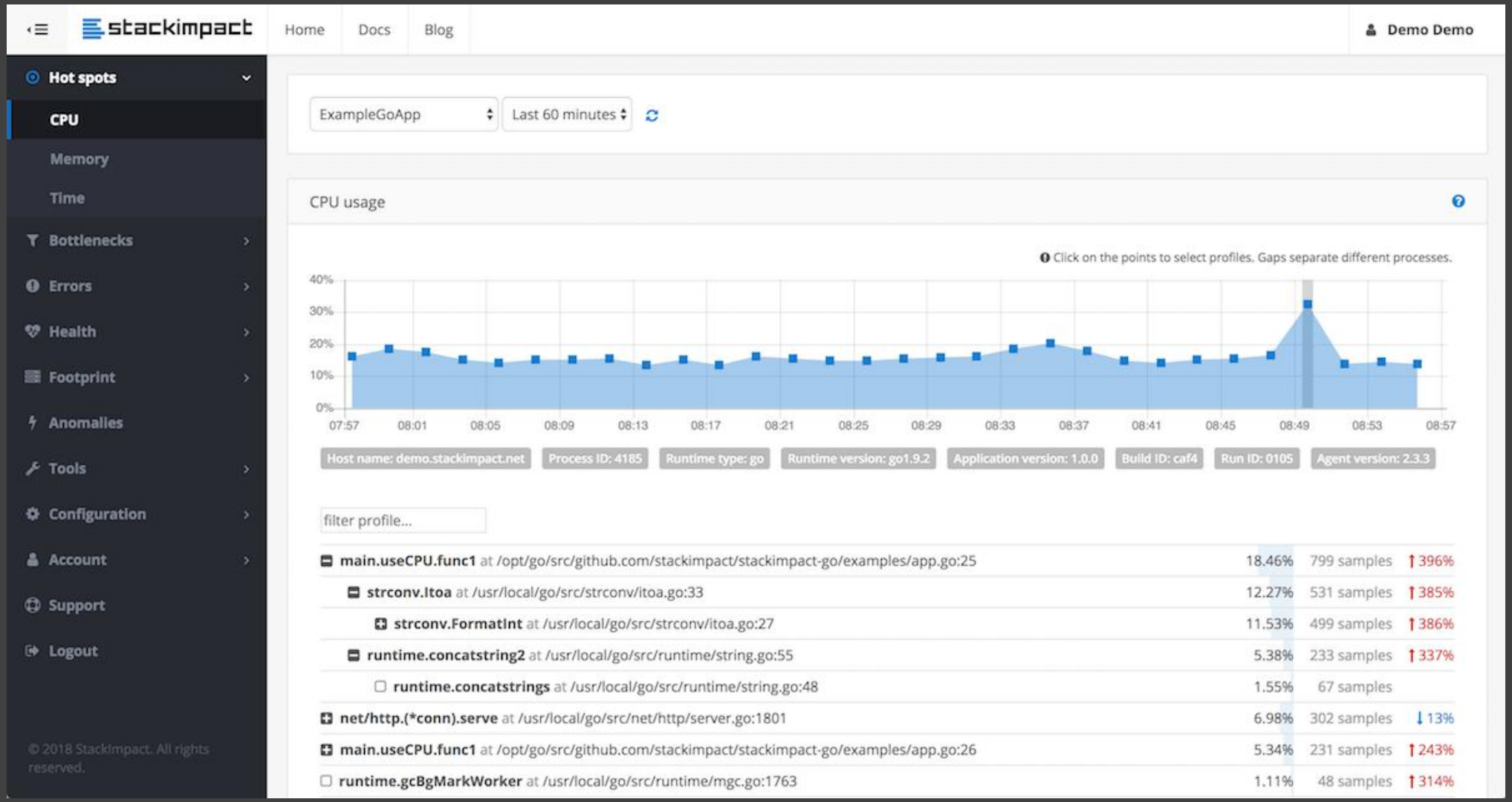
业界先驱给我们的思路

- 缩短性能问题反馈周期
- 节省服务器成本
- 改进开发者的工作流程

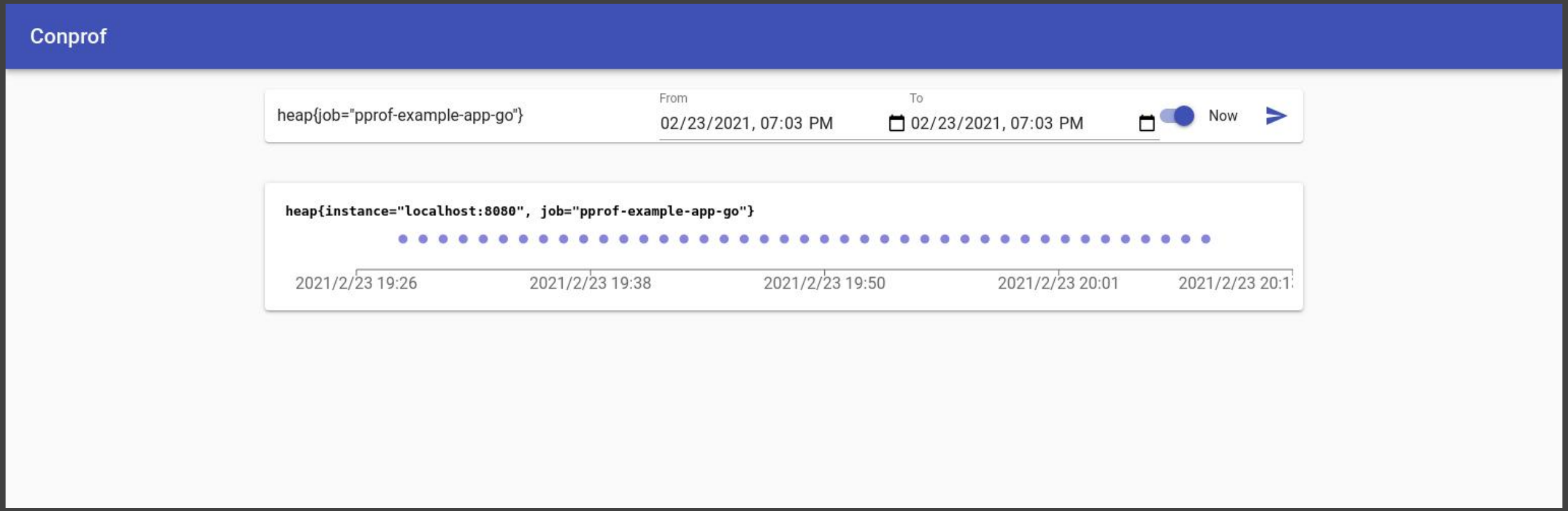
第三部分

相关开源产品

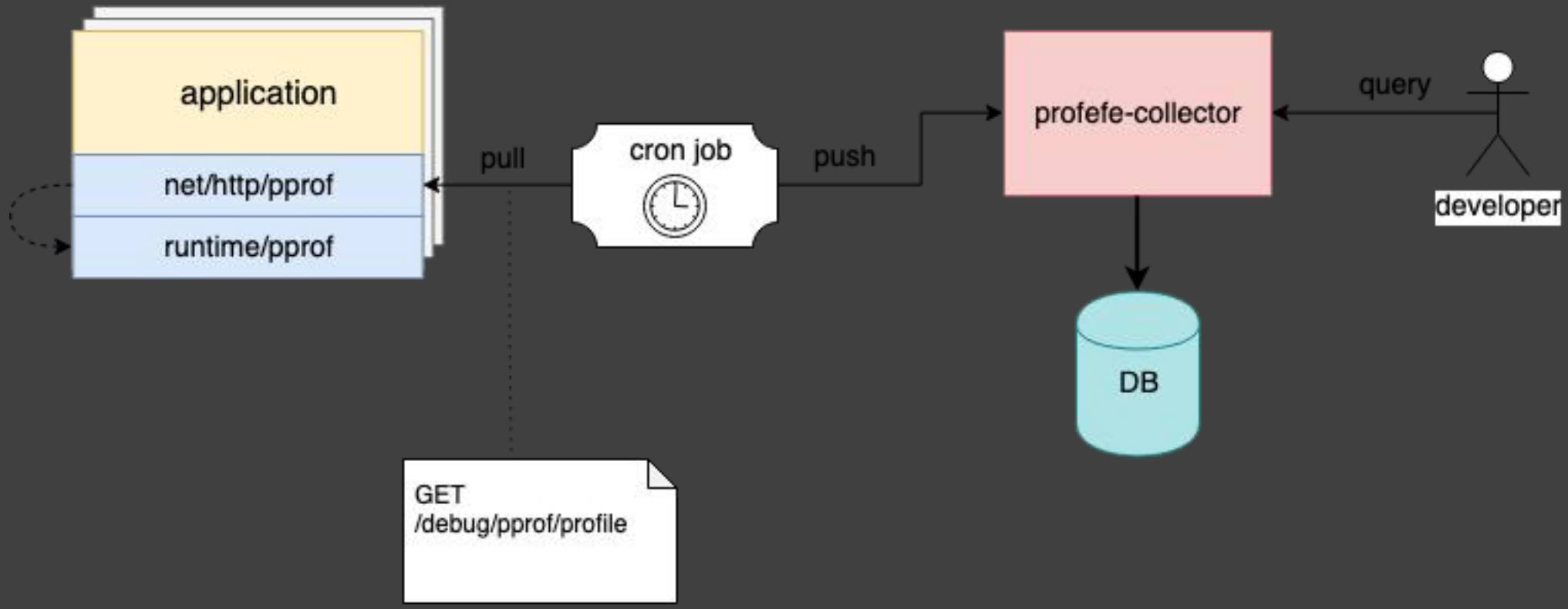
开源产品：stackimpact-go



开源产品：conprof



开源产品：profefe



“

你

这多简单？找监控团队来支持一下就好了嘛！

第四部分

眼前的现状

我们的现状(添油加醋版)

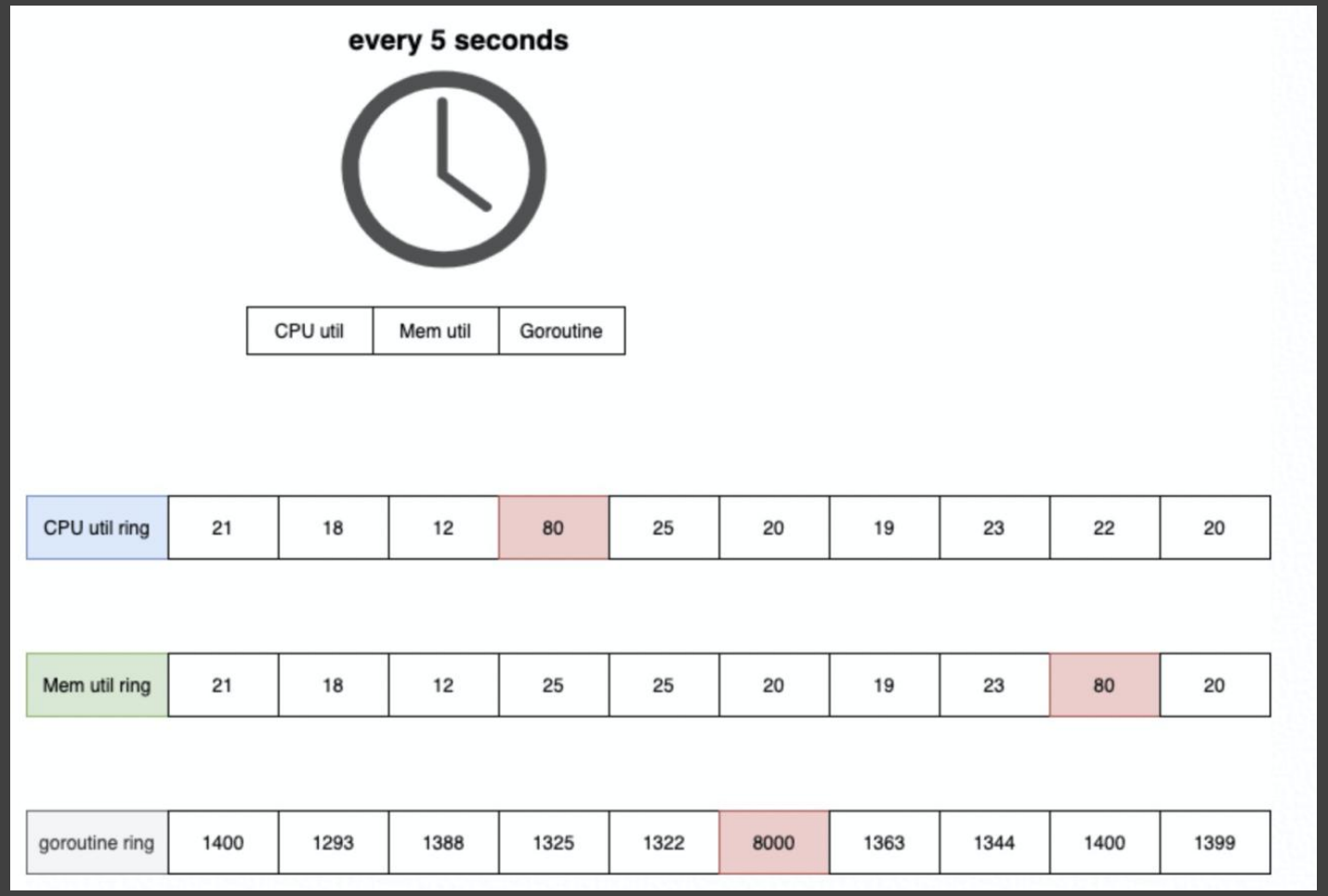
- 基础设施对基础设施的需求在公司内一般不是第一优先级
- 你谁啊你，凭什么支持你的需求而不是去做老板的
- 前端资源没有，你看着办吧
- 这个需求我不想做，先答应着，后面开溜

第五部分

我们的解决方案

解决方案

- Moving Average



线上的场景

- Huge slice make

```
func make1gbslice(wr http.ResponseWriter, req *http.Request) {
        var a = make([]byte, 1073741824)
        _ = a
}
```

线上的场景

- Massive memory allocation

```
func alloc(wr http.ResponseWriter, req *http.Request) {
        var m = make(map[string]string, 102400)
        for i := 0; i < 1000; i++ {
                m[fmt.Sprint(i)] = fmt.Sprint(i)
        }
        _ = m
}
```

线上的场景

- CPU Peak

```
25
26  func cpuex(wr http.ResponseWriter, req *http.Request) {
27          go func() {
28                  for {
29                          time.Sleep(time.Millisecond)
30                  }
31          }()
32  }
```

线上的场景

- Goroutine leak

```
func leak(wr http.ResponseWriter, req *http.Request) {
        taskChan := make(chan int)
        consumer := func() {
                for task := range taskChan {
                        _ = task // do some tasks
                }
        }

        producer := func() {
                for i := 0; i < 10; i++ {
                        taskChan <- i // generate some tasks
                }
                // forget to close the taskChan here
        }

        go consumer()
        go producer()
}
```

线上的场景-这个是吹牛逼的

- Deadlock

```
func req(wr http.ResponseWriter, req *http.Request) {
        l1.Lock()
        defer l1.Unlock()
}

func lockorder1(wr http.ResponseWriter, req *http.Request) {
        l1.Lock()
        defer l1.Unlock()

        time.Sleep(time.Minute)

        l2.Lock()
        defer l2.Unlock()
}

func lockorder2(wr http.ResponseWriter, req *http.Request) {
        l2.Lock()
        defer l2.Unlock()

        time.Sleep(time.Minute)

        l1.Lock()
        defer l1.Unlock()
}
```

线上的场景-这个也是吹牛逼的

- Thread Block

```
func leak(wr http.ResponseWriter, req *http.Request) {
        go func() {
                for i := 0; i < 1000; i++ {
                        go func() {
                                str := "hello cgo"
                                //change to char*
                                cstr := C.CString(str)
                                C.output(cstr)
                                C.free(unsafe.Pointer(cstr))

                        }()
                }
        }()
}
```

谢谢！