sync.Pool 源码分析及适用场景

https://github.com/developer-learning/night-reading-go

sync.Pool 介绍

- A Pool is a set of temporary objects that may be individually saved and retrieved.
- Any item stored in the Pool may be removed automatically at any time without notification. If the Pool holds the only reference when this happens, the item might be deallocated.
- A Pool is safe for use by multiple goroutines simultaneously.

sync.Pool 的由来讨论

Brad Fizpatrick曾建议在sync包里加入一个公开的Cache类型。这个建议引发了一长串的讨论。Go 语言应该在标准库里提供一个这个样子的类型，还是应当将这个类型作为私下的实现? 这个实现应该真的释放内存么? 如果释放，什么时候释放? 这个类型应当叫做Cache，或者更应该叫做Pool?

https://my.oschina.net/u/115763/blog/282376
https://github.com/golang/go/issues/4720

gc (garbage collector)

- Go 是自动垃圾回收，减少了程序员的负担；
- GC 是一把双刃剑，给编程带来便利的同时也增加了运行时开销，使用不当甚至会严重影响程序的性能；
- 高性能场景下：不能任意产生太多的垃圾(GC 负担重，影响性能）；

如何解决GC负重的问题?

- 重用对象；
    - Go 官方团队认识到这个问题普遍存在，避免大家重复造轮子，就出了一个 Pool 包：它设计的目的是用来保存和复用临时对象，以减少内存分配，降低CG压力。
    - https://echo.labstack.com/guide/routing
      Echo 的路由使用了 sync pool 来重复利用内存并且几乎达到了零内存占用。

      直接看源代码：
      github.com/labstack/echo
    - gin的context通过pool来get和put，也就是使用了sync.Pool进行维护，见代码红色部分

Dave Cheney
@davecheney

正在关注

#golang top tip: sync.Pool is not a cache, that is why it isn't called sync.Cache.

翻译推文

上午5:19 - 2014年12月16日

sync.Pool 有两种使用方式

```
// 方法一
package main

import(
    "fmt"
    "sync"
)

func main() {
    // 如果我们不指定 New 函数的话，会返回nil
    p := &sync.Pool{
        New: func() interface{} {
            return 0
        },
    }

    a := p.Get().(int)
    p.Put(1)
    b := p.Get().(int)
    fmt.Println(a, b)
}
```

```
// 方法二
package main

import(
    "fmt"
    "sync"
)

func main() {
    p := &sync.Pool{}
    a := p.Get()
    if a == nil {
       a = func() interface{} {
           return 0
        }
    }
    p.Put(1)
    b := p.Get().(int)
    fmt.Println(a, b)
}
```

sync.Pool 定义的结构

```
type Pool struct {
    noCopy noCopy

    local     unsafe.Pointer // local fixed-size per-P pool, actual type is [P]poolLocal
    localSize uintptr        // size of the local array

    // New optionally specifies a function to generate
    // a value when Get would otherwise return nil.
    // It may not be changed concurrently with calls to Get.
    New func() interface{}
}

// Local per-P Pool appendix.
type poolLocalInternal struct {
    private interface{}   // Can be used only by the respective P.
    shared  []interface{} // Can be used by any P.
    Mutex                 // Protects shared.
}

type poolLocal struct {
    poolLocalInternal

    // Prevents false sharing on widespread platforms with
    // 128 mod (cache line size) = 0 .
    pad [128 - unsafe.Sizeof(poolLocalInternal{})%128]byte
}
```

sync.Pool 的特性

1. 池不可以指定大小，大小只受制于GC临界值；

2. 对象的最大缓存周期是GC 周期，当GC 调用时 没有被引用的对象都会被清理掉；（Pool 包在 init 会注册一个 poolCleanup 函数：清除所有 pool 里面的缓存对象，该函数注册之后会在每次 gc 之前都会调用）

如果我们在 Get() 之前执行 runtime.GC() ， 则我们获取到的值就是0；

3. Get方法返回时是返回池中任意一个对象，没有顺序；如果池中没有对象，那么调用New方法新生成一个；如果没有指定New方法，那么返回nil；

sync.Pool 的适用场景

当多个goroutine都需要创建同一个对象的时候，如果goroutine过多，可能导致对象的创建数目剧增。 而对象又是占用内存的，进而导致的就是内存回收的GC压力徒增。造成“并发大－占用内存大－GC缓慢－处理并发能力降低－并发更大”这样的恶性循环。

===>在这个时候，我们非常迫切需要有一个对象池，每个goroutine不再自己单独创建对象，而是从对象池中获取出一个对象（如果池中已经有的话）

Use sync.Pool

- https://stackoverflow.com/questions/38505830/how-to-implement-memory-pooling-in-golang
- http://www.akshaydeo.com/blog/2017/12/23/How-did-I-improve-latency-by-700-percent-using-syncPool/
- [译] CockroachDB GC优化总结 https://segmentfault.com/a/1190000004046212
- Golang 优化之路——临时对象池 http://blog.cyeam.com/golang/2017/02/08/go-optimize-slice-pool

其他

- https://github.com/golang/go/issues/23199
- https://github.com/golang/go/issues/22950

sync.Pool相关视频

- Mastering Go Programming : Syncs and Locks | packtpub.com
  https://youtu.be/Chys1QbqZDw

- justforfunc #37: sync.Pool from the pool
  https://youtu.be/rfXSrgIGrKo

  优化的情况：
  https://github.com/campoy/justforfunc/blob/master/37-sync-pool/perfs.txt

- SREcon17 Asia/Australia: Golang's Garbage
  https://youtu.be/q1h2g84EX1M

CodeReview

- 我们来看看 Google Go 团队是如何进行 CodeReview的 https://go-review.googlesource.com/c/go/+/8202