由一个简单程序图解Go语言内存分配和管理

原创 yoko Go语言中文网 2019-11-21

点击上方蓝色“Go语言中文网”关注我们，领全套Go资料，每天学习 Go 语言

题图

本文基于 Go 1.13

Go 程序的内存从申请阶段到不再使用后的释放阶段都由 Go 标准库自动管理。尽管管理工作不需要开发者参与，但是 Go 对内存管理的底层实现做了非常好的优化，里面充满了有意思的知识点，还是值得我们学习的。

从堆上申请内存

Go 内存管理的设计目标是在并发环境下保持高性能，并且集成垃圾回收器。让我们从一个简单的例子开始：

```
package main

type smallStruct struct {
        a, b int64
        c, d float64
}
```

```
func main() {
        smallAllocation()
}

//go:noinline
func smallAllocation() *smallStruct {
        return &smallStruct{}
}
```

`//go:noinline` 这行注释可以禁止编译时的内联优化，从而避免编译时把 `smallAllocation` 这个函数调用直接优化没了。

运行逃逸分析命令 `go tool compile "-m" main.go` ，得到内存申请情况：

```
main.go:14:9: &smallStruct literal escapes to heap
```

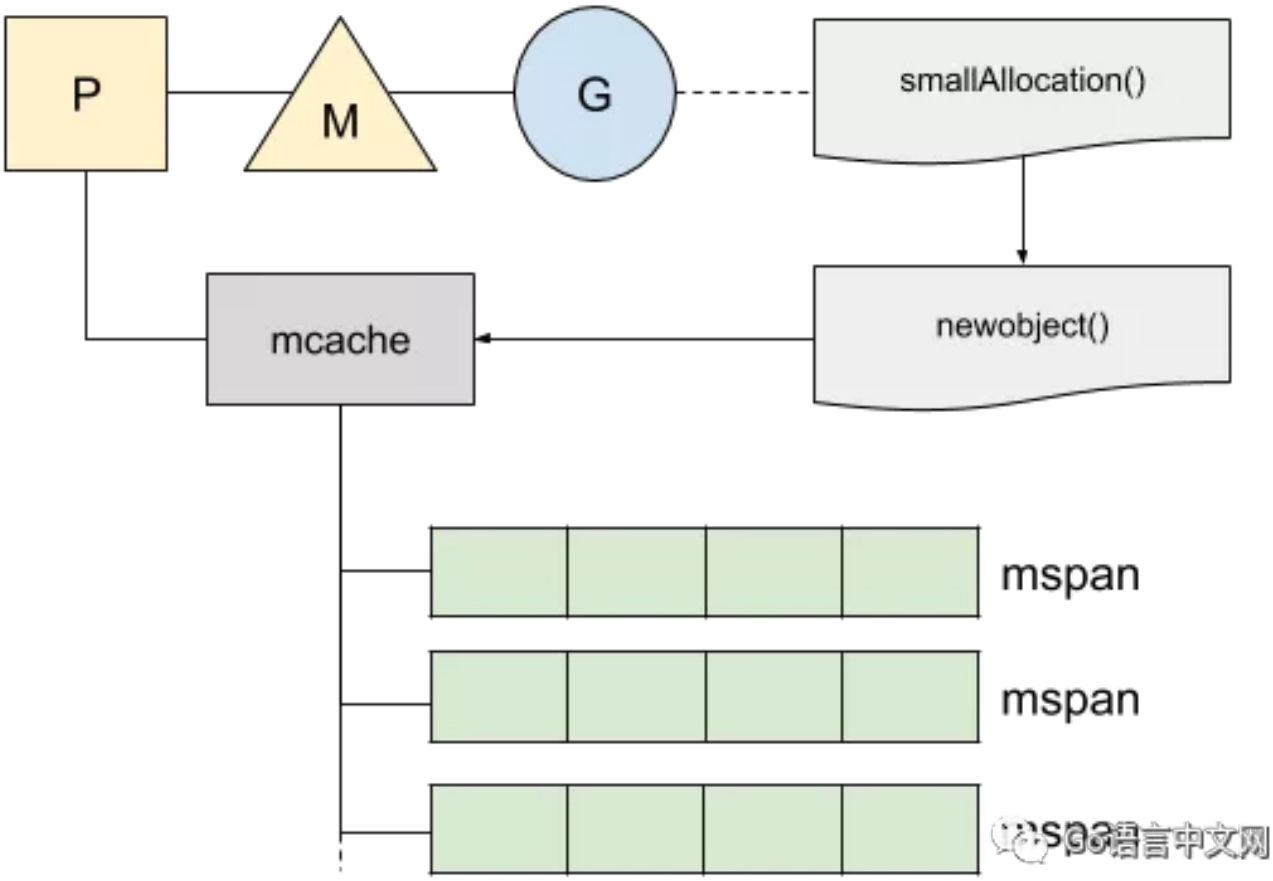
运行 `go tool compile -S main.go` 命令，获取程序的汇编代码，可以更清晰的查看内存申请情况：

```
0x001d 00029 (main.go:14)   LEAQ   type."".smallStruct(SB), AX
0x0024 00036 (main.go:14)  PCDATA $0, $0
0x0024 00036 (main.go:14)  MOVQ   AX, (SP)
0x0028 00040 (main.go:14)  CALL   runtime.newobject(SB)
```

`newobject` 是用于申请内存的内建函数， `newobject` 是 `mallocgc` 的代理， `mallocgc` 是管理堆内存的函数。Go 分配内存有两种策略：小块内存申请和大块内存申请。

小块内存申请

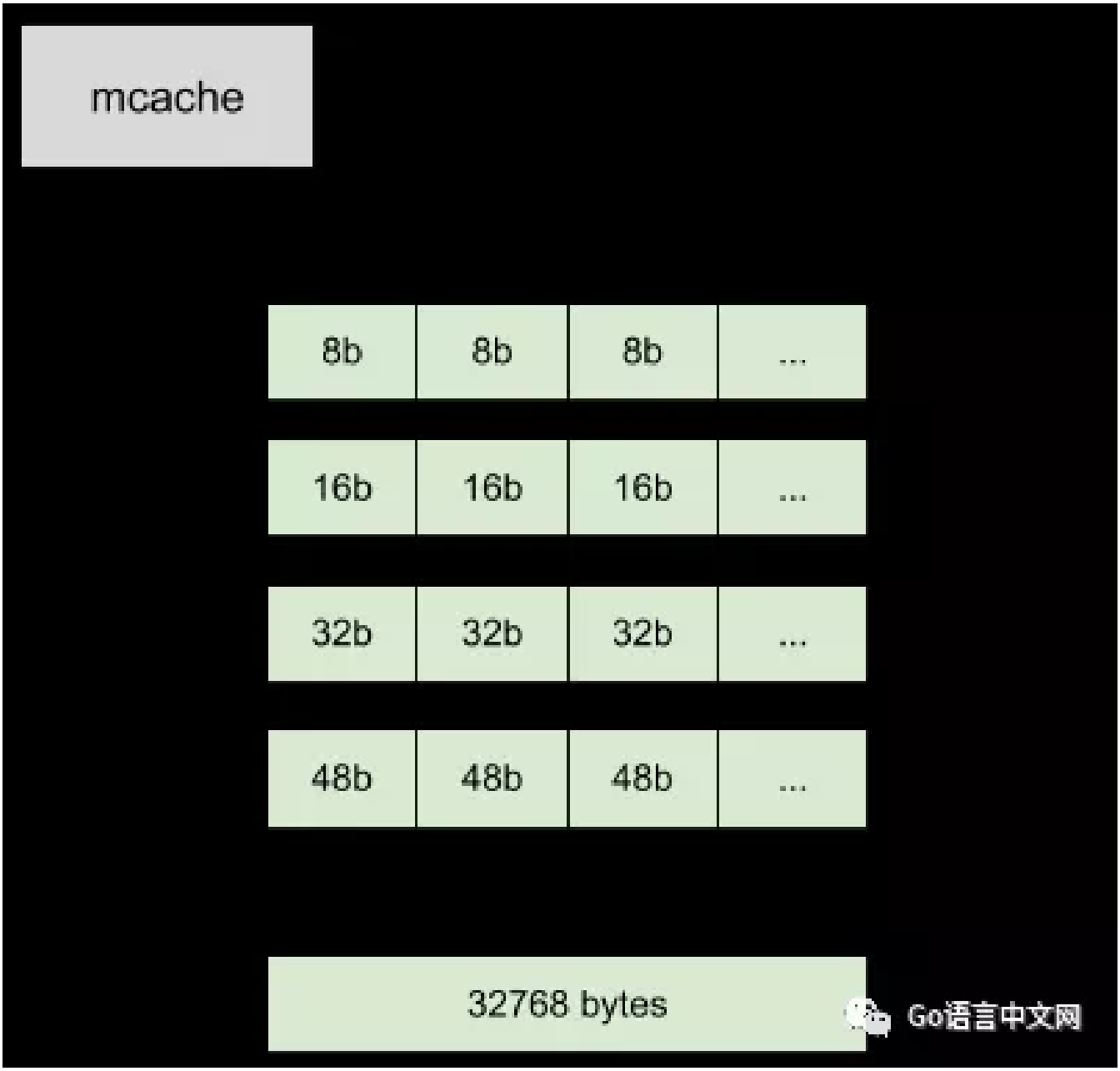
对于 32KB 以下的小块内存申请，Go 会尝试从本地缓存 `mcache` 中获取内存。 `mcache` 包含了一系列被称为 `mspan` 的 `span` 列表， `mspan` 包含了可供分配使用的内存：



从mcache申请

Go 的线程调度模型中，每个系统线程 M 和一个上下文 P 挂钩，在一个指定时间点最多只能处理一个协程 G 。申请内存时，当前协程会首先在所属 M 的本地缓存中的 span 列表中查找可用的内存块。使用本地缓存的好处是不用加锁，更高效。

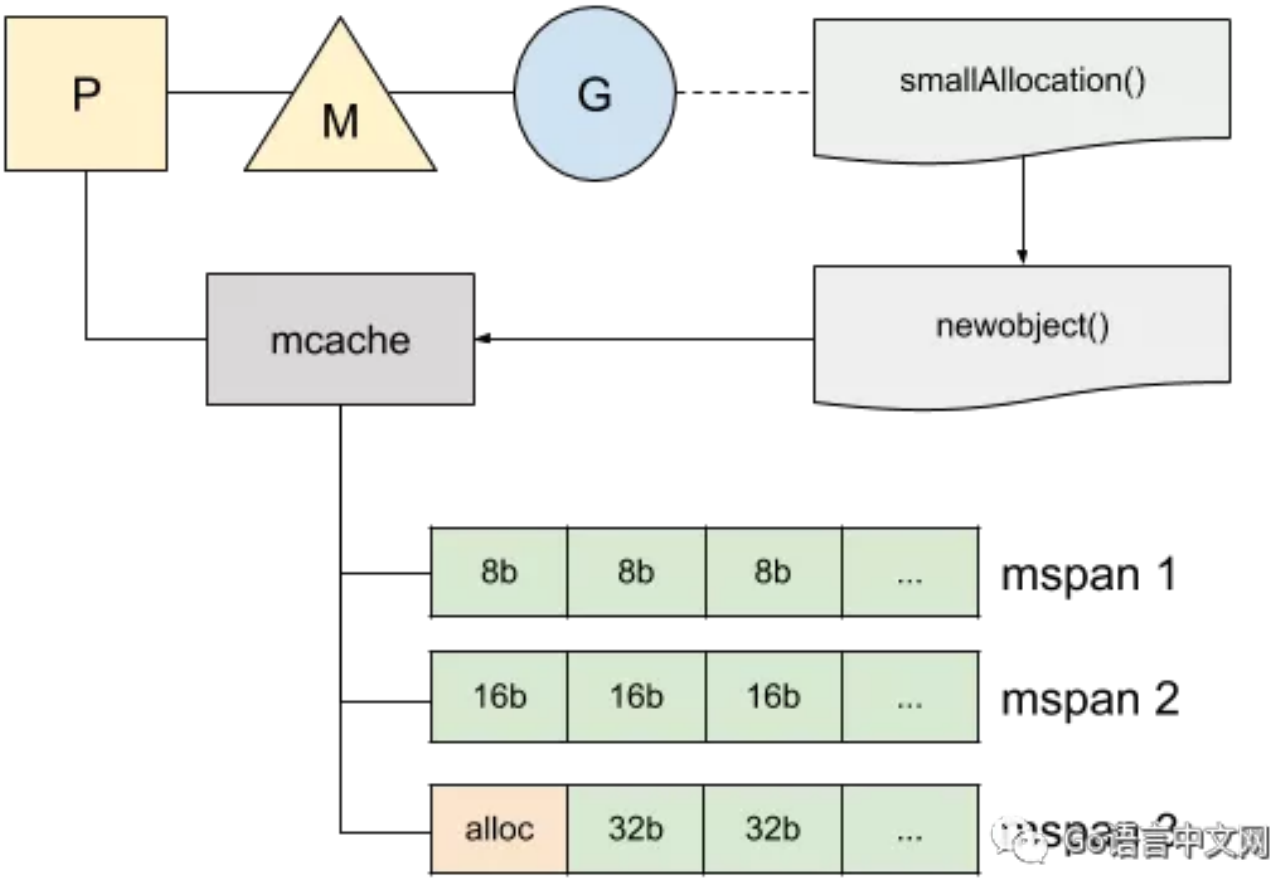
span 列表按大小被划分为大约 70 个等级，大小从 8 字节到 32K 字节不等，不同等级存储不同大小的内存块：



span大小等级

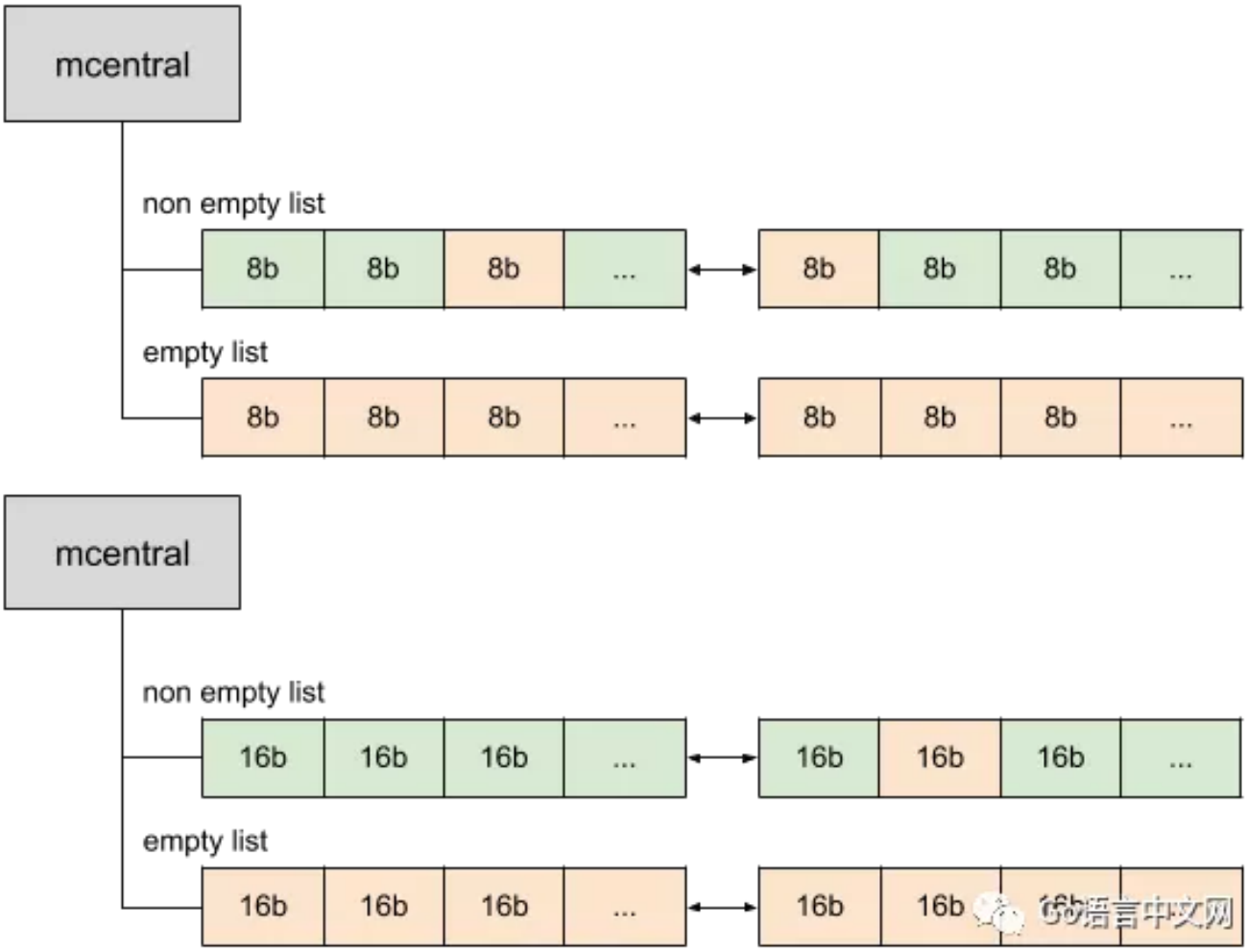
每个等级的 span 会存在两份：一个用于存储内部不包含指针的对象，另一个用于存储内部包含指针的对象。这么的好处是垃圾回收时更高效，因为不需要扫描不包含指针的那个 span 。

在我们前面的例子，结构体的大小为 32 字节，所以使用 32 字节的 span ：



mcache

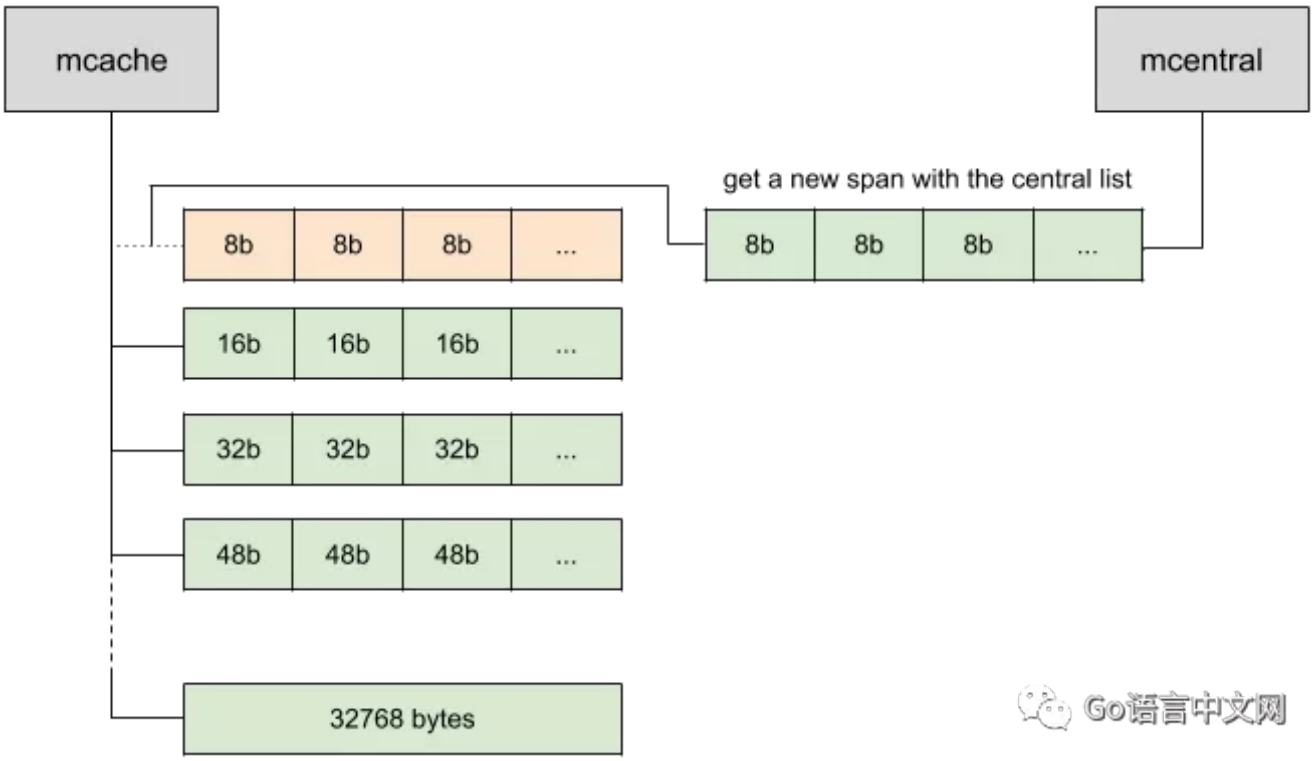
现在，你可能会奇怪如果 `mcache` 上没有空闲的内存块可供分配该怎么办。Go 另外还维护了全局的 `span` 列表，同样也按大小分成多个级别，叫做 `mcentral` 。 `mcentral` 包含两种链表，一张包含空闲内存块，一张包含已使用内存块：



中心列表

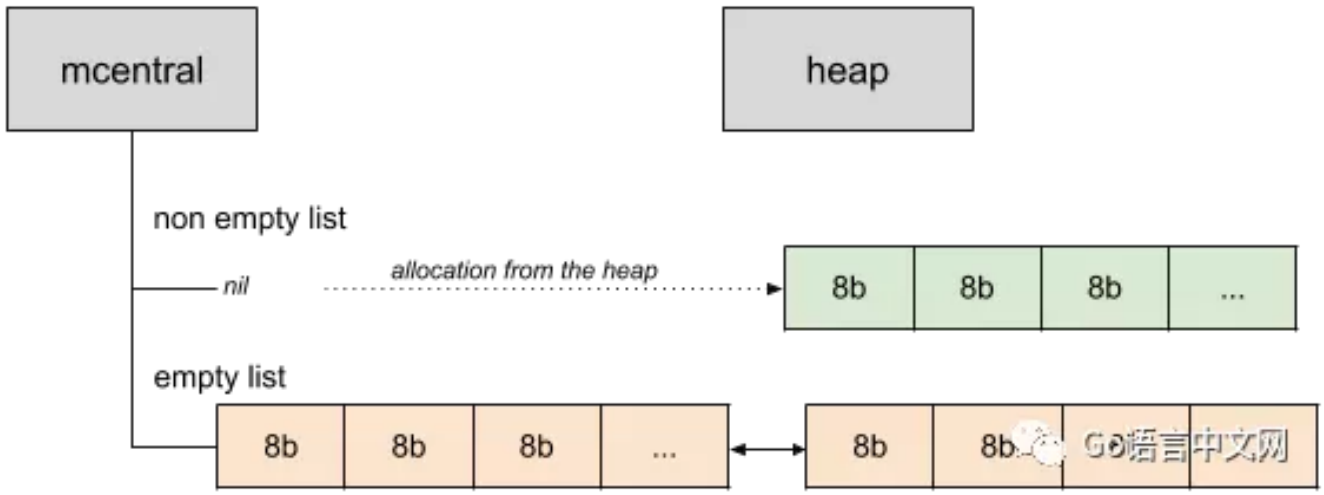
`mcentral` 维护了两张 `span` 链表。一张链表为 `non-empty` 类型，包含了可供分配的 `span` （由于一个 `span` 可能包含多个 `object` ，只要有一个或一个以上的 object 可供分配即表示该 `span` 可供分配），一张为 `empty` 类型，包含已分配完毕的 `span` 。当 Go 执行垃圾回收时，如果 `span` 中的内存块被标记为可供分配， `span` 会重新加入到 `non-empty` 链表中。

从 `mcentral` 获取 `span` 的流程图如下：



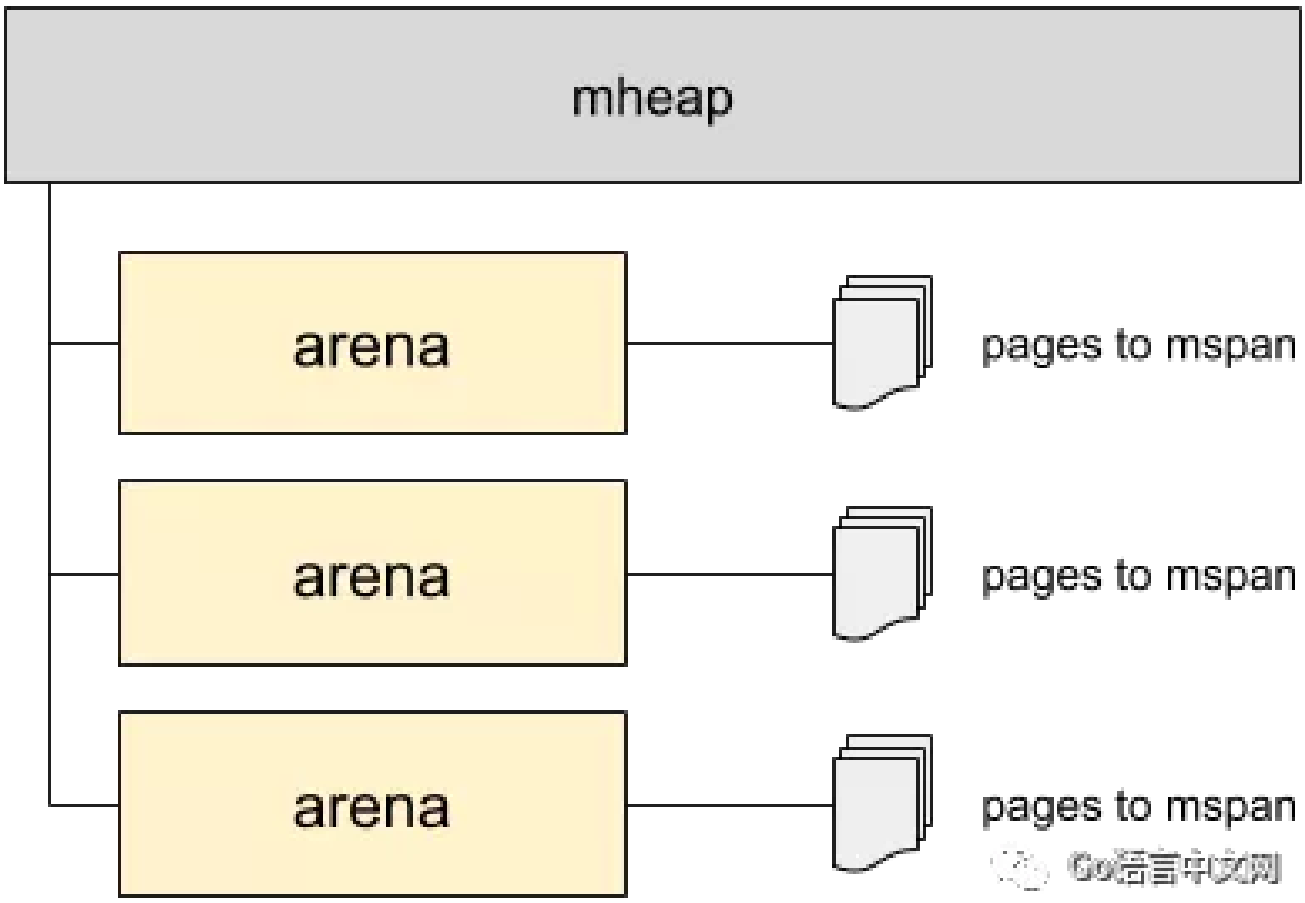
从mcentral获取span

当 `mcentral` 中也没有可供分配的 `span` 时，Go 会从堆上申请新的 `span` 并将其放入 `mcentral` 中：



从堆上获取span

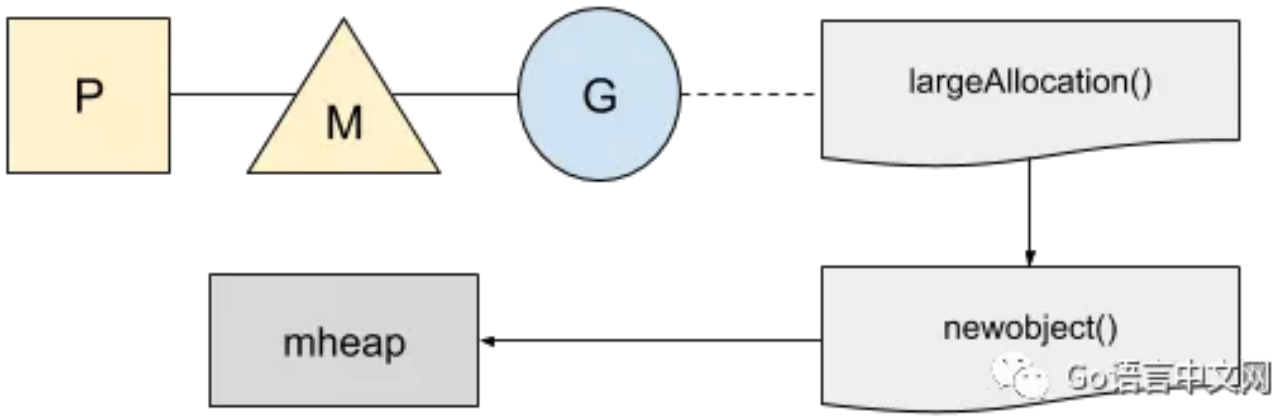
堆在必要时向操作系统申请内存。它会申请一块大内存，被称为 `arena` ，在 64 位系统下为 64MB，其它大部分系统为 4MB，申请的内存同样用 `span` 管理：



arena

大块内存申请

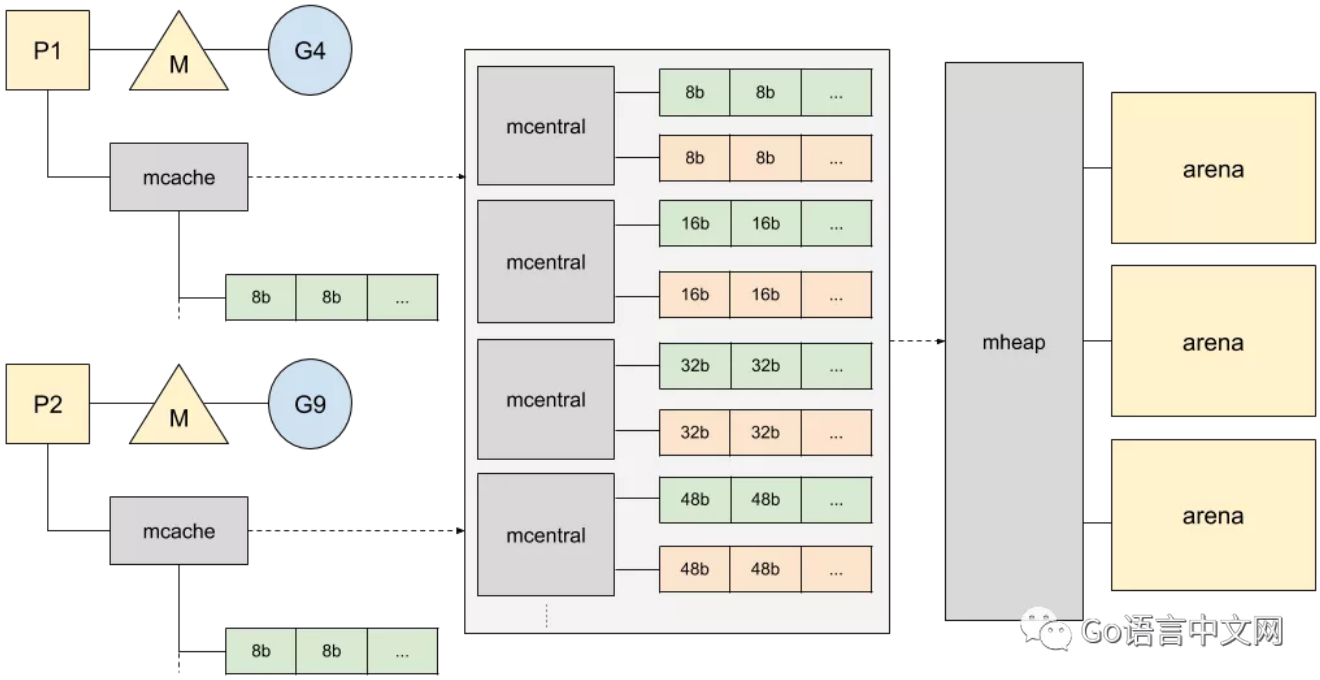
Go 申请大于 32KB 的大块内存不使用本地缓存策略，而是将大小取整到页大小整数倍后直接从堆上申请。



从堆上申请大内存

全局图

现在我们在一个较高层次上，对 Go 的内存分配有了一个大致了解。让我们将所有的组件集合到一起来绘制一张全局图：



全局图

设计灵感

Go 内存分配器的设计基于 TCMalloc，TCMalloc 是由 Google 专门为并行环境优化的内存分配器。TCMalloc 的文档[1]很值得一读，在文档里你也能找到本文中讲解到的一些概念。

> 英文原文地址：**https://medium.com/a-journey-with-go/go-memory-management-and-allocation-a7396d430f44**[2]

> 原文链接：**https://pengrl.com/p/38720/**[3]
>
> 原文出处：**yoko blog**[4] (**https://pengrl.com**[5])
>
> 原文作者： yoko
>
> 版权声明： 本文欢迎任何形式转载，转载时完整保留本声明信息（包含原文链接、原文出处、原文作者、版权声明）即可。本文后续所有修改都会第一时间在原始地址更新。

推荐阅读