



Dive into arena package

~ Go 1.20 release party ~

The Go gopher was designed by Renee French.

Takuma Shibuya

Twitter/GitHub @sivchari

- golangci-lint
- Kubernetes
- Go Conference 2021 Autumn
- Go Conference 2022 Spring
- Go Conference mini 2022 Autumn IN SENDAI



Today's Talk

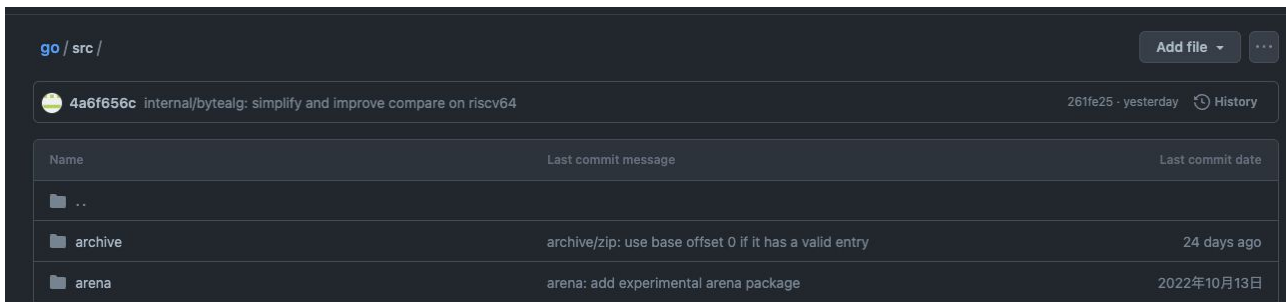
- What's arena package ?
- arena proposal & concurrent mark and sweep GC
- How to use the arena package ?
- Source code
- Appendix

Let's Go

What's arena package ?

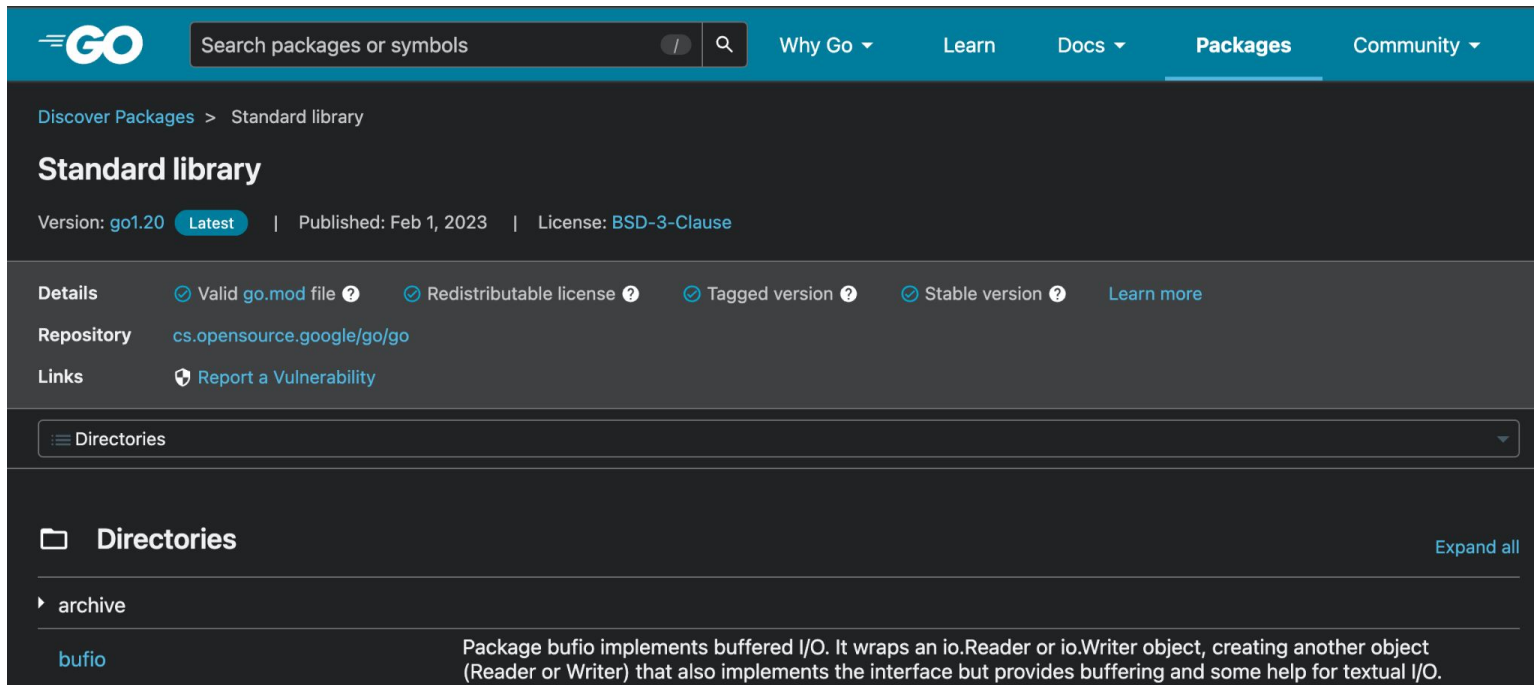
arenaはGo1.20からGo Teamが試験的にサポートを開始したパッケージ

Go1.20では標準パッケージに入っている



What's arena package ?

[Go Doc](#)には存在しない



The screenshot shows the Go Packages website interface. At the top is a navigation bar with the Go logo, a search bar, and links for 'Why Go', 'Learn', 'Docs', 'Packages' (which is highlighted), and 'Community'. Below the navigation bar, the breadcrumb 'Discover Packages > Standard library' is visible. The main heading is 'Standard library'. Below this, it shows 'Version: go1.20' with a 'Latest' badge, 'Published: Feb 1, 2023', and 'License: BSD-3-Clause'. A 'Details' section contains several status indicators: 'Valid go.mod file', 'Redistributable license', 'Tagged version', and 'Stable version', each with a checkmark and a help icon, followed by a 'Learn more' link. The 'Repository' section shows the URL 'cs.opensource.google/go/go'. The 'Links' section has a link to 'Report a Vulnerability'. Below these sections is a 'Directories' section with a dropdown menu currently showing 'Directories'. Under the 'Directories' heading, there is a list of directories, with 'archive' expanded to show the 'bufio' package. The 'bufio' package description is: 'Package bufio implements buffered I/O. It wraps an io.Reader or io.Writer object, creating another object (Reader or Writer) that also implements the interface but provides buffering and some help for textual I/O.'

GO

Search packages or symbols

Why Go ▾ Learn Docs ▾ **Packages** Community ▾

Discover Packages > Standard library

Standard library

Version: [go1.20](#) **Latest** | Published: Feb 1, 2023 | License: [BSD-3-Clause](#)

Details [✔ Valid go.mod file ?](#) [✔ Redistributable license ?](#) [✔ Tagged version ?](#) [✔ Stable version ?](#) [Learn more](#)

Repository [cs.opensource.google/go/go](#)

Links [🛡 Report a Vulnerability](#)

Directories ▾

Directories

[Expand all](#)

▸ archive

[bufio](#)

Package bufio implements buffered I/O. It wraps an io.Reader or io.Writer object, creating another object (Reader or Writer) that also implements the interface but provides buffering and some help for textual I/O.

What's arena package ?

1/18 [doc/go1.20: remove mention of arena goexperiment](#)

arena packageが他の APIに侵食される可能性がある

Googleの中でも極めてかぎられたケースで使われている

まだ実験的なものであり予告なしに変更、削除される可能性がある

Release noteに載せると実験的とはいえサポートしているように見えるため Release noteから削除

What's arena package ?

`src/internal/goexperiment/exp_arenas_on.go`

```
//go:build goexperiment.arenas
```

```
// +build goexperiment.arenas
```

```
package goexperiment
```

```
const Arenas = true
```

```
const ArenasInt = 1
```


What's arena package ?

- GOEXPERIMENTという環境変数を使う

GOEXPERIMENT=arenas go build main.go

- build constraintも存在する

go build -tags goexperiment.arenas main.go

```
package main
```

```
import "arena"
```

```
func main() {
```

```
    a := arena.NewArena()
```

```
}
```

What's arena package ?

- **GOEXPERIMENT=arenas godoc**

Packages

Standard library
Other packages
Sub-repositories
Community

Standard library ▼

Name	Synopsis
archive	
tar	Package tar implements access to tar archives.
zip	Package zip provides support for reading and writing ZIP archives.
arena	The arena package provides the ability to allocate memory for a collection of Go values and free that space manually all at once, safely.

arena proposal

[proposal: arena: new package providing memory arenas #51317](#)

2022/2/23

GoはGCが存在する言語だが、arenaはユーザーが自分でメモリを確保、利用、解放を行う

Google社内で実装され、GCのCPU時間とHeap使用量の削減を行った

社内の大規模アプリケーションでは CPUとメモリ使用時間が最大 15%削減できた

Proposalの例だと Protobufが挙げられている

arena proposal

Background

- GoはGCを持っているためユーザーがメモリ管理を気にする必要はない
- 大規模な Goのサービスは GCに多くの CPU時間をつかっている
- 大規模サービスの JSONやprotobufで入れ子のメッセージなどを扱う際に有用

arena proposal

Background

- 大規模サービスの *JSON* や *protobuf* で入れ子のメッセージなどを扱う際に有用

concurrent mark and sweep (GC)

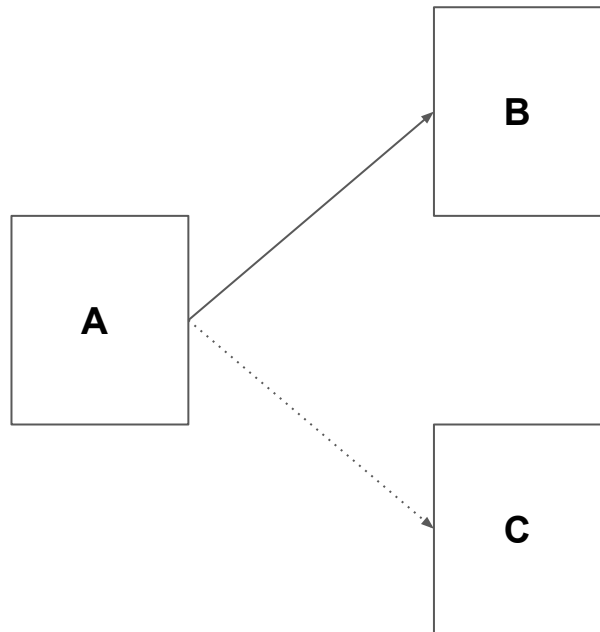
Initial Mark (STW)

Concurrent Mark

Mark Termination (STW)

Concurrent Sweep

Sweep Termination (STW)



concurrent mark and sweep (GC)

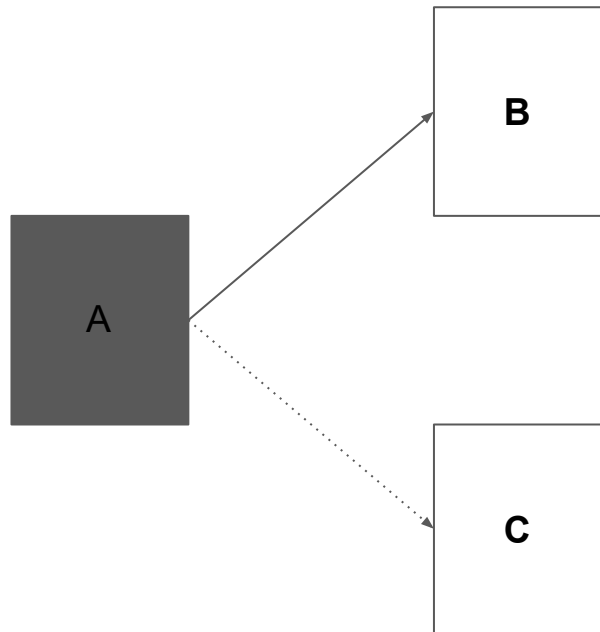
Initial Mark (STW)

Concurrent Mark

Mark Termination (STW)

Concurrent Sweep

Sweep Termination (STW)



concurrent mark and sweep (GC)

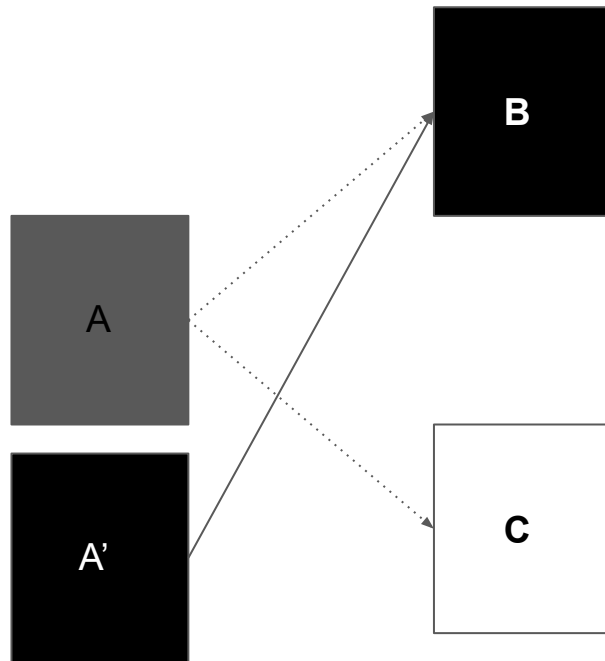
Initial Mark (STW)

Concurrent Mark

Mark Termination (STW)

Concurrent Sweep

Sweep Termination (STW)



concurrent mark and sweep (GC)

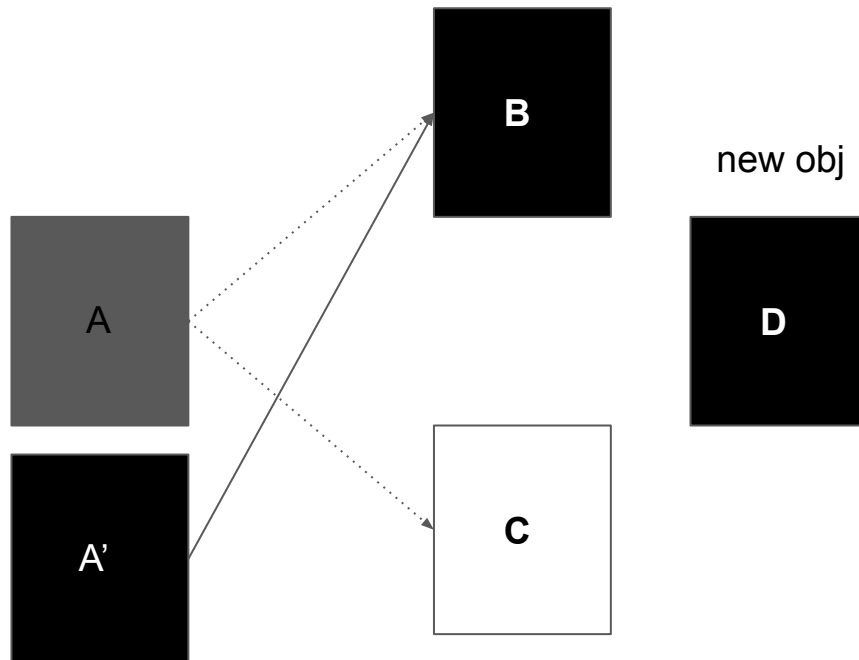
Initial Mark (STW)

Concurrent Mark

Mark Termination (STW)

Concurrent Sweep

Sweep Termination (STW)



concurrent mark and sweep (GC)

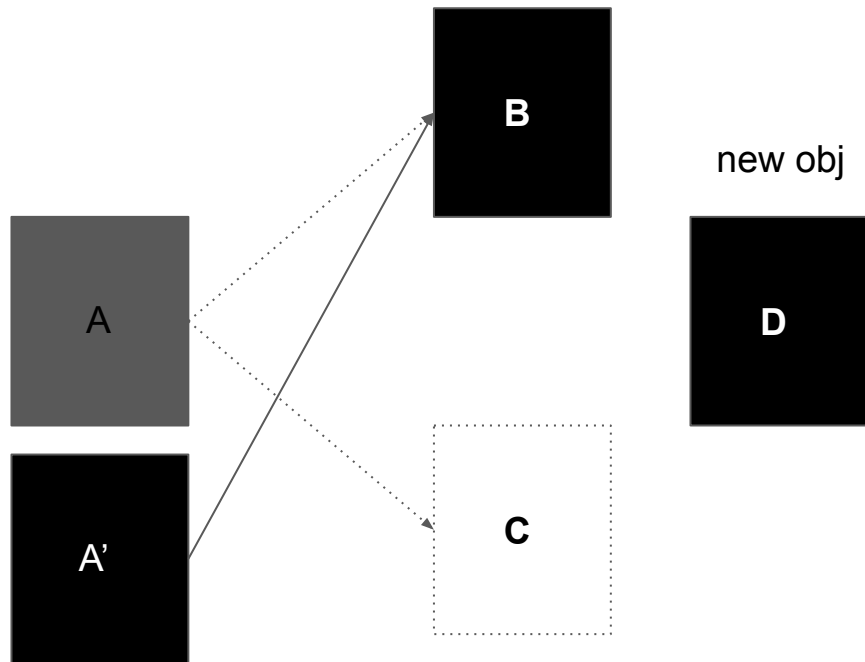
Initial Mark (STW)

Concurrent Mark

Mark Termination (STW)

Concurrent Sweep

Sweep Termination (STW)



concurrent mark and sweep (GC)

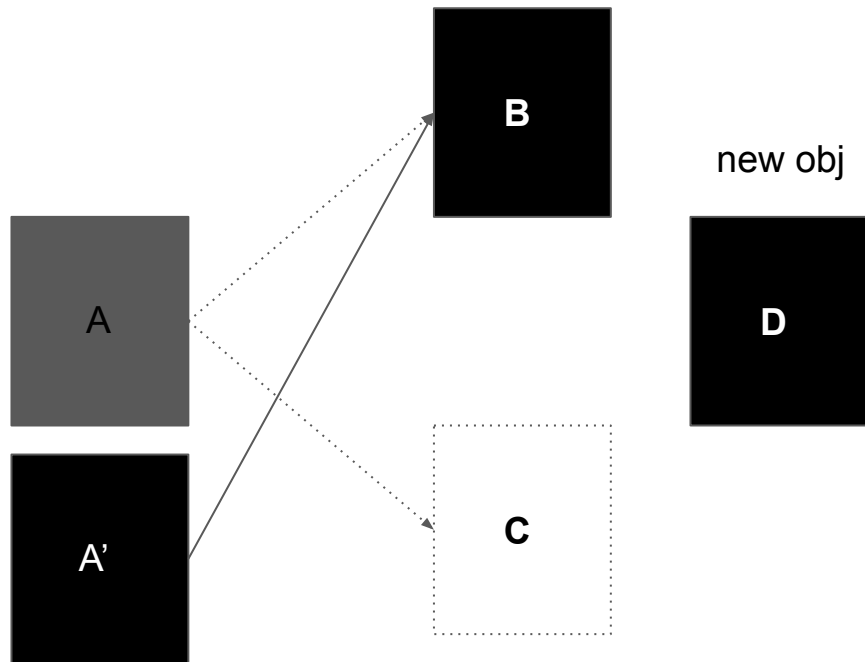
Initial Mark (STW)

Concurrent Mark

Mark Termination (STW)

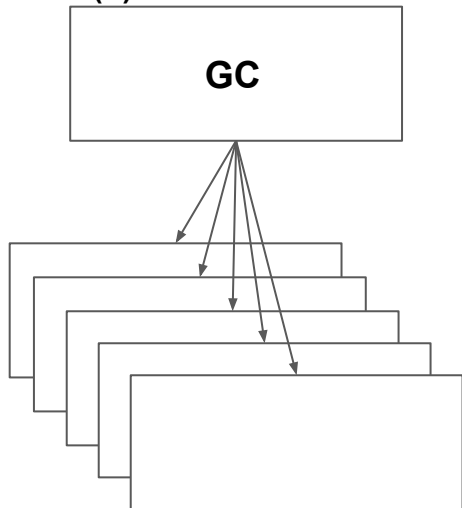
Concurrent Sweep

Sweep Termination (STW)

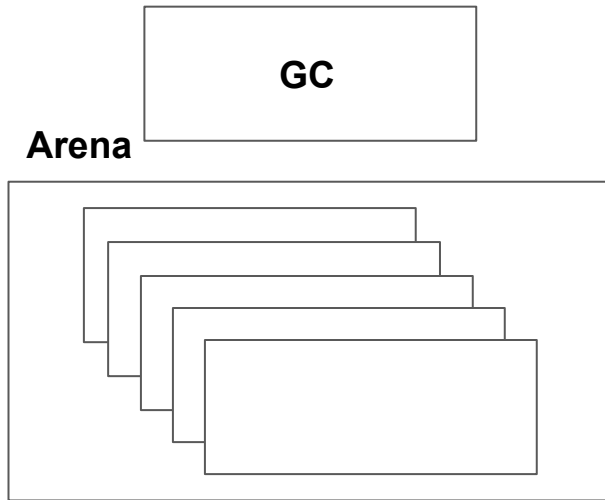


arena proposal

- 複雑なデータ構造 (e.g. JSON/protobuf)
 - Mark&Sweep == STWのためのGCのオーバーヘッド
 - $O(n)$



- アリーナなら？
 - メモリの一括解放
 - $O(1)$



How to use the arena package ?

API

- **func Clone[T any](s T) T**
- **func MakeSlice[T any](a *Arena, len, cap int) []T**
- **func New[T any](a *Arena) *T**
- **type Arena**
 - **func NewArena() *Arena**
 - **func (a *Arena) Free()**

How to use the arena package ?

Usage

```
package main

import (
    "arena"
)

func main() {
    a := arena.NewArena()
    i := arena.New[int](a)
    *i = 0
    println(*i) => 0
    *i = 1
    println(*i) => 1
    ii := arena.Clone(i)
    println(*ii) => 1
    slicei := arena.MakeSlice[int](a, 1, 1)
    slicei[0] = 1
    println(slicei[0]) => 1
    a.Free()
}
```

How to use the arena package ?

Usage

```
package main

import (
    "arena"
)

func main() {
    a := arena.NewArena() // allocates a new user arena.
    i := arena.New[int](a)
    *i = 0
    println(*i) => 0
    *i = 1
    println(*i) => 1
    ii := arena.Clone(i)
    println(*ii) => 1
    slicei := arena.MakeSlice[int](a, 1, 1)
    slicei[0] = 1
    println(slicei[0]) => 1
    a.Free()
}
```

How to use the arena package ?

Usage

```
package main

import (
    "arena"
)

func main() {
    a := arena.NewArena()
    i := arena.New[int](a) // creates a new *T in the provided arena. don't use the *T after the arena is freed.
    *i = 0
    println(*i) => 0
    *i = 1
    println(*i) => 1
    ii := arena.Clone(i)
    println(*ii) => 1
    slicei := arena.MakeSlice[int](a, 1, 1)
    slicei[0] = 1
    println(slicei[0]) => 1
    a.Free()
}
```


How to use the arena package ?

Usage

```
package main

import (
    "arena"
)

func main() {
    a := arena.NewArena()
    i := arena.New[int](a)
    *i = 0
    println(*i) => 0
    *i = 1
    println(*i) => 1
    ii := arena.Clone(i) // shallow copy.
    println(*ii) => 1
    slicei := arena.MakeSlice[int](a, 1, 1)
    slicei[0] = 1
    println(slicei[0]) => 1
    a.Free()
}
```

How to use the arena package ?

Usage

```
package main

import (
    "arena"
)

func main() {
    a := arena.NewArena()
    i := arena.New[int](a)
    *i = 0
    println(*i) => 0
    *i = 1
    println(*i) => 1
    ii := arena.Clone(i)
    println(*ii) => 1
    slicei := arena.MakeSlice[int](a, 1, 1) // creates a new []T with the provided capacity and length.
    slicei[0] = 1
    println(slicei[0]) => 1
    a.Free()
}
```

How to use the arena package ?

Usage

```
package main

import (
    "arena"
)

func main() {
    a := arena.NewArena()
    i := arena.New[int](a)
    *i = 0
    println(*i) => 0
    *i = 1
    println(*i) => 1
    ii := arena.Clone(i)
    println(*ii) => 1
    slicei := arena.MakeSlice[int](a, 1, 1)
    slicei[0] = 1
    println(slicei[0]) => 1
    a.Free() // free the arena and all objects allocated from the arena
}
```

How to use the arena package ?

Panic cases

- `(*Arena).Free()`のあとに同じ Arenaを参照する
 - goroutine safeではない 異なるライフタイムで参照してしまう
 - 2回Freeした場合
- `Clone()`でpointer, slice, string以外を渡す

How to use the arena package ?

Panic cases

- *(*Arena).Free()*のあとに同じ *Arena*を参照する
 - *goroutine safe*ではないため、異なるライフタイムで参照してしまう
 - 2回Freeした場合
- *Clone()*でpointer, slice, string以外を渡す

How to use the arena package ?

Panic cases

- *(*Arena).Free()*のあとに同じ *Arena*を参照する
 - *goroutine safe*ではないため、異なるライフタイムで参照してしまう
- *goroutine A*がループ内で *arena.New*で取得した **T*に対して値を書き込みプリントする
- *main goroutine*が *(*Arena).Free()*を行う
- *goroutine A*が再度取得しようすると *nil pointer reference*になる
- ループ外で先に取得したアドレスに書き込むと *(*Arena).Free()*のあとでも書き換えられる
 - *go build -asan*で検知できる

How to use the arena package ?

Panic cases

- *(*Arena).Free()*のあとに同じ *Arena*を参照する
 - goroutine safeではないため、異なるライフタイムで参照してしまう
 - 2回*Free*した場合
- *Clone()*でpointer, slice, string以外を渡す

How to use the arena package ?

Panic cases

- `(*Arena).Free()`のあとに同じ *Arena*を参照する
 - 2回Freeした場合

エラーメッセージは同期的、非同期的のパターンで異なる (実装自体は後述)

How to use the arena package ?

Panic cases

- `(*Arena).Free()`のあとに同じ Arenaを参照する
 - goroutine safeではないため、異なるライフタイムで参照してしまう
 - 2回Freeした場合
- `Clone()`で`pointer`, `slice`, `string`以外を渡す

How to use the arena package ?

Panic cases

- *Clone()* で *pointer, slice, string* 以外を渡す
 - 基本的に `arena.New[T any](s T)` の戻り値は `*T` になるので `dereference` しなければ OK
 - データ構造的に `pointer` 演算で `arena package` が `data` をとれる範囲がサポートされてそう (実体へのポインタを持っているデータ構造)

source code

```
package main

import "arena"

func main() {
    a := arena.NewArena()
    defer a.Free()
    i := arena.New[int](a)
    *i = 1
    println(*i)
    clonei := arena.Clone(i)
}
```

source code

```
// ...
```

```
// An Arena is automatically freed once it is no longer referenced, so it  
must be kept alive (see runtime.KeepAlive) until any memory allocated  
from it is no longer needed.
```

```
// An Arena must never be used concurrently by multiple goroutines.
```

```
type Arena struct {  
    a unsafe.Pointer  
}
```

source code

- Arenaは長い間参照されない場合 GCの対象になってしまうため参照され続けている必要がある (runtime.KeepAliveはおそらくその例)
- Arenaはgoroutine safedではないことがコメントでも書かれている

source code

arena	runtime
NewArena	arena_newArena
New	arena_arena_New
MakeSlice	arena_arena_Slice
Clone	arena_heapify
(*Arena).Free	arena_arena_Free

source code

```
package main

import "arena"

func main() {
    a := arena.NewArena()
    defer a.Free()
    i := arena.New[int](a)
    *i = 1
    println(*i)
    clonei := arena.Clone(i)
}
```

source code

arena	runtime
NewArena	arena_newArena
New	arena_arena_New
MakeSlice	arena_arena_Slice
Clone	arena_heapify
(*Arena).Free	arena_arena_Free

source code

```
func NewArena() *Arena {  
    return &Arena{a: runtime_arena_newArena()}  
}
```

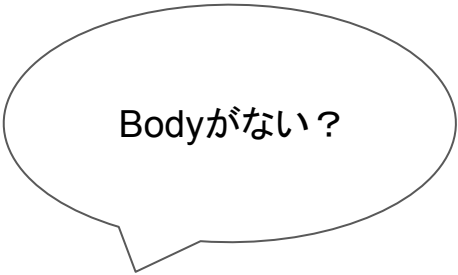
source code

```
func NewArena() *Arena {  
    return &Arena{a: runtime_arena_newArena()}  
}
```

source code

```
//go:linkname runtime_arena_new_Arena  
func runtime_arena_newArena(arena unsafe.Pointer, typ any) any
```

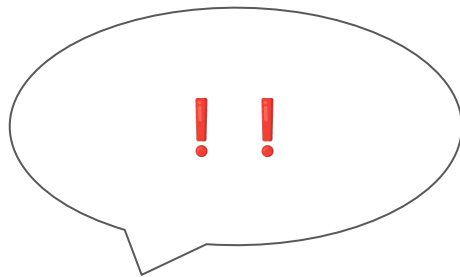
source code



Bodyがない？

```
//go:linkname runtime_arena_new_Arena  
func runtime_arena_newArena(arena unsafe.Pointer, typ any) any
```

source code



```
//go:linkname runtime_arena_new_Arena
```

```
func runtime_arena_newArena(arena unsafe.Pointer, typ any) any
```

source code

ref. [compiler command](#)

```
//go:linkname localname [importpath.name]
```

This special directive does not apply to the Go code that follows it. Instead, the `//go:linkname` directive instructs the compiler to use “importpath.name” as the object file symbol name for the variable or function declared as “localname” in the source code. If the “importpath.name” argument is omitted, the directive uses the symbol's default object file symbol name and only has the effect of making the symbol accessible to other packages. Because this directive can subvert the type system and package modularity, it is only enabled in files that have imported “unsafe”.

source code

ref. [compiler command](#)

```
//go:linkname localname [importpath.name]
```

雑にまとめると

- `//go:linkname localname [importpath.name]`とすると `localname`のBodyは無視されて `importpath.name`が実際のBodyになる
- `[importpath.name]`を省略するとシンボル名を外部に向けて公開できる

source code

リンク先は [go/src/runtime/arena.go](https://golang.org/src/runtime/arena.go)

```
//go:linkname arena_newArena arena.runtime_arena_newArena
func arena_newArena() unsafe.Pointer {
    return unsafe.Pointer(newUserArena())
}
```


source code

リンク先は [go/src/runtime/arena.go](https://golang.org/src/runtime/arena.go)

```
//go:linkname arena_newArena arena.runtime_arena_newArena
func arena_newArena() unsafe.Pointer {
    return unsafe.Pointer(newUserArena())
}
```

リンク元は [go/src/arena/arena.go](https://golang.org/src/arena/arena.go)

```
//go:linkname runtime_arena_newArena
func runtime_arena_newArena(arena unsafe.Pointer, typ any) any
```

source code

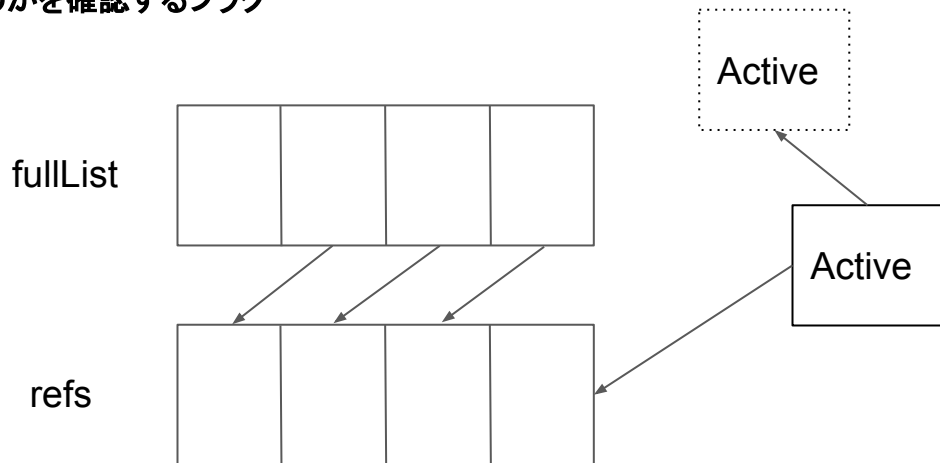
```
//go:linkname arena_newArena arena.runtime_arena_newArena  
func arena_newArena() unsafe.Pointer {  
    return unsafe.Pointer(newUserArena())  
}
```

source code

```
type userArena struct {  
    fullList *mspan  
  
    active *mspan  
  
    refs []unsafe.Pointer  
  
    defunct atomic.Bool  
}
```

source code

- userArenaの各メモリ領域は chunk
- fullListは空きメモリが十分でない chunkのリスト
- activeは現在使用しているアロケートする chunk
- refsは現在管理している fullListとactive全ての chunkのアドレスをもつ unsafe.Pointerのスライスで先頭は必ず active であり、fullListのheadがrefsの2番目になる
- defunctはfreeしたかどうかを確認するフラグ



source code

```
// newUserArena creates a new userArena ready to be used.
func newUserArena() *userArena {
    a := new(userArena)
    SetFinalizer(a, func(a *userArena) {
        // If arena handle is dropped without being freed, then call
        // free on the arena, so the arena chunks are never reclaimed
        // by the garbage collector.
        a.free()
    })
    a.refill()
    return a
}
```

source code

```
// newUserArena creates a new userArena ready to be used.
func newUserArena() *userArena {
    a := new(userArena)
    SetFinalizer(a, func(a *userArena) {
        // If arena handle is dropped without being freed, then call
        // free on the arena, so the arena chunks are never reclaimed
        // by the garbage collector.
        a.free()
    })
    a.refill()
    return a
}
```

source code

```
// newUserArena creates a new userArena ready to be used.
func newUserArena() *userArena {
    a := new(userArena)
    SetFinalizer(a, func(a *userArena) {
        // If arena handle is dropped without being freed, then call
        // free on the arena, so the arena chunks are never reclaimed
        // by the garbage collector.
        a.free()
    })
    a.refill()
    return a
}
```

source code

```
func (a *userArena) refill() *mspan {  
    s := a.active  
    var x unsafe.Pointer  
    if len(userArenaState.reuse) > 0 {  
        // Pick off the last arena chunk from the list.  
        n := len(userArenaState.reuse) - 1  
        x = userArenaState.reuse[n].x  
        s = userArenaState.reuse[n].mspan  
    }  
    if s == nil {  
        // Allocate a new one.  
        x, s = newUserArenaChunk()  
    }  
    a.refs = append(a.refs, x)  
    a.active = s  
    return s  
}
```


source code

```
package main

import "arena"

func main() {
    a := arena.NewArena()
    defer a.Free()
    i := arena.New[int](a)
    *i = 1
    println(*i)
    clonei := arena.Clone(i)
}
```

source code

arena	runtime
NewArena	arena_newArena
New	arena_arena_New
MakeSlice	arena_arena_Slice
Clone	arena_heapify
(*Arena).Free	arena_arena_Free

source code

```
func New[T any](a *Arena) *T {  
    return runtime_arena_arena_New(a.a, reflectlite.TypeOf((*T)(nil))).(*T)  
}
```

source code

```
func New[T any](a *Arena) *T {  
    return runtime_arena_arena_New(a.a, reflectlite.TypeOf((*T)(nil))).(*T)  
}
```

source code

```
//go:linkname runtime_arena_arena_New
```

```
func runtime_arena_arena_New(arena unsafe.Pointer, typ any) any
```

source code

```
//go:linkname arena_arena_New arena.runtime_arena_arena_New
func arena_arena_New(arena unsafe.Pointer, typ any) any {
    t := (*_type)(efaceOf(&typ).data)
    if t.kind&kindMask != kindPtr {
        throw("arena_New: non-pointer type")
    }
    te := (*ptrtype)(unsafe.Pointer(t)).elem
    x := ((*userArena)(arena)).new(te)
    var result any
    e := efaceOf(&result)
    e._type = t
    e.data = x
    return result
}
```

source code

```
//go:linkname arena_arena_New arena.runtime_arena_arena_New
func arena_arena_New(arena unsafe.Pointer, typ any) any {
    t := (*_type)(efaceOf(&typ).data)
    if t.kind&kindMask != kindPtr {
        throw("arena_New: non-pointer type")
    }
    te := (*ptrtype)(unsafe.Pointer(t)).elem
    x := ((*userArena)(arena)).new(te)
    var result any
    e := efaceOf(&result)
    e._type = t
    e.data = x
    return result
}
```

source code

```
// This operation is not safe to call concurrently with other operations on the same arena
func (a *userArena) new(typ *_type) unsafe.Pointer {
    return a.alloc(typ, -1)
}
```


source code

```
func (a *userArena) alloc(typ *_type, cap int) unsafe.Pointer {  
    s := a.active // active割当て  
    var x unsafe.Pointer  
    for {  
        // 割り当てるcapが負の数ならtyp通りに、そうでないならcap分確保する  
        // MakeSliceと共通で呼ばれる  
        x = s.userArenaNextFree(typ, cap)  
        if x != nil {  
            break  
        }  
        s = a.refill()  
    }  
    return x  
}
```

source code

```
func (s *mspan) userArenaNextFree(typ *_type, cap int) unsafe.Pointer {  
    size := typ.size  
    // userArenaChunkMaxAllocBytesはGOOSにより異なる  
    if size > userArenaChunkMaxAllocBytes {  
        // userArenaChunkMaxAllocBytesを超える場合heapにredirect  
        if cap >= 0 {  
            return newarray(typ, cap)  
        }  
        return newobject(typ)  
    }  
    // Prevent preemption M  
    mp.mallocing = 1
```

source code

```
func (s *mspan) userArenaNextFree(typ *_type, cap int) unsafe.Pointer {  
    // 末尾 or 先頭からsize分引いてアライメントする  
    if typ.ptrdata == 0 {  
        v, ok := s.userArenaChunkFree.takeFromBack(size, typ.align)  
        if ok {  
            ptr = unsafe.Pointer(v)  
        }  
    } else {  
        v, ok := s.userArenaChunkFree.takeFromFront(size, typ.align)  
        if ok {  
            ptr = unsafe.Pointer(v)  
        }  
    }  
    if ptr == nil {  
        // releasemを行い、preemptionを許可する  
        mp.mallocing = 0  
        releasem(mp)  
        return nil  
    }  
}
```

source code

```
package main

import "arena"

func main() {
    a := arena.NewArena()
    defer a.Free()
    i := arena.New[int](a)
    *i = 1
    println(*i)
    clonei := arena.Clone(i)
}
```

source code

arena	runtime
NewArena	arena_newArena
New	arena_arena_New
MakeSlice	arena_arena_Slice
Clone	arena_heapify
(*Arena).Free	arena_arena_Free

source code

```
func Clone[T any](s T) T {  
    return runtime_arena_heapify(s).(T)  
}
```

source code

```
func Clone[T any](s T) T {  
    return runtime_arena_heapify(s).(T)  
}
```

source code

```
//go:linkname runtime_arena_heapify  
func runtime_arena_heapify(any) any
```


source code

```
//go:linkname arena_heapify arena.runtime_arena_heapify
func arena_heapify(s any) any {
    var v unsafe.Pointer
    e := efaceOf(&s)
    t := e._type
    switch t.kind & kindMask {
    case kindString:
        v = stringStructOf((*string)(e.data)).str
    case kindSlice:
        v = (*slice)(e.data).array
    case kindPtr:
        v = e.data
    default:
        panic("arena: Clone only supports pointers, slices, and strings")
    }
    span := spanOf(uintptr(v))
    if span == nil || !span.isUserArenaChunk {
        // Not stored in a user arena chunk.
        return s
    }
}
```

source code

```
//go:linkname arena_heapify arena.runtime_arena_heapify
func arena_heapify(s any) any {
    // Heap-allocate storage for a copy.
    var x any
    switch t.kind & kindMask {
    case kindString:
        ..
    case kindSlice:
        ..
    case kindPtr:
        ..
    }
    return x
}
```

source code

```
package main

import "arena"

func main() {
    a := arena.NewArena()
    defer a.Free()
    i := arena.New[int](a)
    *i = 1
    println(*i)
    clonei := arena.Clone(i)
}
```

source code

arena	runtime
NewArena	arena_newArena
New	arena_arena_New
MakeSlice	arena_arena_Slice
Clone	arena_heapify
(*Arena).Free	arena_arena_Free

source code

```
//go:linkname runtime_arena_arena_Free  
func runtime_arena_arena_Free(arena unsafe.Pointer)
```

source code

```
func arena_arena_Free(arena unsafe.Pointer) {  
    ((*userArena)(arena)).free()  
}
```

source code

```
func arena_arena_Free(arena unsafe.Pointer) {  
    ((*userArena)(arena)).free()  
}
```

source code

```
func (a *userArena) free() {  
    // Check for a double-free.  
    if a.defunct.Load() {  
        panic("arena double free")  
    }  
    // Mark ourselves as defunct.  
    a.defunct.Store(true)  
    SetFinalizer(a, nil)  
}
```


source code

```
func (a *userArena) free() {  
    // Check for a double-free.  
    // 非同期に2つのgoroutineが解放した場合  
    // goroutine A がdefunctをtrueにするため、panicする  
    if a.defunct.Load() {  
        panic("arena double free")  
    }  
    // Mark ourselves as defunct.  
    a.defunct.Store(true)  
    SetFinalizer(a, nil)  
}
```

source code

```
func (a *userArena) free() {  
    // Free all the full arenas.  
    // fullListの2番目がrefsの先頭  
    s := a.fullList  
    i := len(a.refs) - 2  
    for s != nil {  
        a.fullList = s.next  
        s.next = nil  
        freeUserArenaChunk(s, a.refs[i])  
        s = a.fullList  
        i--  
    }  
    if a.fullList != nil || i >= 0 {  
        // fullListを全て解放しきれなかった場合はthrowされる  
        throw("full list doesn't match refs list in length")  
    }  
}
```

source code

```
func (a *userArena) free() {  
    // Free all the full arenas.  
    // fullListの2番目がrefsの先頭  
    s := a.fullList  
    i := len(a.refs) - 2  
    for s != nil {  
        a.fullList = s.next  
        s.next = nil  
        freeUserArenaChunk(s, a.refs[i])  
        s = a.fullList  
        i--  
    }  
    if a.fullList != nil || i >= 0 {  
        // fullListを全て解放しきれなかった場合はthrowされる  
        throw("full list doesn't match refs list in length")  
    }  
}
```

source code

```
func freeUserArenaChunk(s *mspan, x unsafe.Pointer) {
    mp := acquirem()
    // We can only set user arenas to fault if we're in the _GCoff phase.
    if gcphase == _GCoff {
        lock(&userArenaState.lock)
        faultList := userArenaState.fault
        userArenaState.fault = nil
        unlock(&userArenaState.lock)
        s.setUserArenaChunkToFault()
        for _, lc := range faultList {
            lc.mspan.setUserArenaChunkToFault()
        }
        // Until the chunks are set to fault, keep them alive via the fault list.
        KeepAlive(x)
        KeepAlive(faultList)
    } else {
        // Put the user arena on the fault list.
        lock(&userArenaState.lock)
        userArenaState.fault = append(userArenaState.fault, liveUserArenaChunk{s, x})
        unlock(&userArenaState.lock)
    }
    releasem(mp)
}
```

source code

```
// arena packageにより確保されているメモリはGCとは別でユーザーが管理するため_GCoffのときだけfaultにする
// _GCoffではないときはfault listで参照を残す
func freeUserArenaChunk(s *mspan, x unsafe.Pointer) {
    if gcphase == _GCoff {
        faultList := userArenaState.fault
        userArenaState.fault = nil
        s.setUserArenaChunkToFault()
        for _, lc := range faultList {
            lc.mspan.setUserArenaChunkToFault()
        }
        // Until the chunks are set to fault, keep them alive via the fault list.
        KeepAlive(x)
        KeepAlive(faultList)
    } else {
        userArenaState.fault = append(userArenaState.fault, liveUserArenaChunk{s, x})
    }
}
```

source code

```
func freeUserArenaChunk(s *mspan, x unsafe.Pointer) {
    s = a.active
    if s != nil {
        if raceenabled || msanenabled || asanenabled {
            // Don't reuse arenas with sanitizers enabled. We want to catch
            // any use-after-free errors aggressively.
            freeUserArenaChunk(s, a.refs[len(a.refs)-1])
        } else {
            lock(&userArenaState.lock)
            userArenaState.reuse = append(userArenaState.reuse, liveUserArenaChunk{s,
a.refs[len(a.refs)-1]})
            unlock(&userArenaState.lock)
        }
    }
    // nil out a.active so that a race with freeing will more likely cause a crash.
    a.active = nil
    a.refs = nil
}
```

source code

```
func freeUserArenaChunk(s *mspan, x unsafe.Pointer) {  
    s = a.active  
    if s != nil {  
        if raceenabled || msanenabled || asanenabled {  
            // Don't reuse arenas with sanitizers enabled. We want to catch  
            // any use-after-free errors aggressively.  
            freeUserArenaChunk(s, a.refs[len(a.refs)-1])  
        }  
    }  
}
```

source code

```
func freeUserArenaChunk(s *mspan, x unsafe.Pointer) {  
    s = a.active  
    if s != nil {  
        } else {  
            lock(&userArenaState.lock)  
            userArenaState.reuse = append(userArenaState.reuse,  
liveUserArenaChunk{s, a.refs[len(a.refs)-1]})  
            unlock(&userArenaState.lock)  
        }  
    }  
}
```


Appendix

Arenaのコードが初めてマージされたのは 2022/10/13

The screenshot shows the Go Change interface for a merged change. The top navigation bar includes 'Go', 'CHANGES', 'DOCUMENTATION', 'BROWSE', a search bar, and a 'Sign in' link. The change title is 'runtime: add safe arena support to the runtime' with a 'Merged' status and change number '423359'. The 'Change Info' section on the left lists the submitter as Michael Knyszek, reviewers as Cherry Mui and Gopher Robot, and the code owner as Paul Murphy. It also shows 'Submit Requirements' (Code-Review, No-Holds, Review-Enforcement) and 'Trigger Votes' (Run-TryBot, TryBot-Result). The main content area on the right contains the commit message, a detailed description of the change, and a 'Relation chain' of related merged changes.

Go CHANGES DOCUMENTATION BROWSE

Merged 423359 runtime: add safe arena support to the runtime

Change Info SHOW ALL

Submitted Oct 13

Owner Michael Knyszek

Reviewers Cherry Mui Gopher Robot

CC Paul Murphy

Repo | Branch go | master

Submit Requirements

- Code-Review +2
- No-Holds No votes
- Review-Enforcement Satisfied

Trigger Votes

- Run-TryBot +1
- TryBot-Result +1

runtime: add safe arena support to the runtime

This change adds an API to the runtime for arenas. A later CL can potentially export it as an experimental API, but for now, just the runtime implementation will suffice.

The purpose of arenas is to improve efficiency, primarily by allowing for an application to manually free memory, thereby delaying garbage collection. It comes with other potential performance benefits, such as better locality, a better allocation strategy, and better handling of

SHOW ALL

Comments 48 resolved

Checks No results

Relation chain

- [misc/cgo/test: add asan and msan arena tests](#) ✓ (Merged)
- [arena: add experimental arena package](#) ✓ (Merged)
- [runtime: add safe arena support to the runtime](#) ✓ (Merged)
- [runtime: make \(*mheap\).sysAlloc more general](#) ✓ (Merged)
- [runtime: factor out GC assist credit accounting](#) ✓ (Merged)
- [runtime: factor out mheap span initialization](#) ✓ (Merged)

Appendix

arena proposalは2022/08/21にlockされ2022/10/15にunlockされている



golang locked as **too heated** and limited conversation to collaborators on Aug 21, 2022



golang unlocked this conversation on Oct 15, 2022

Appendix

- 実装がマージされた間に discussionはできない状態になっていた
- sync.Poolとの比較は解決されておらず平行線のまま
- protobufのサンプルコードはどこにもないがパフォーマンスが上がったことだけは proposalに書かれている
- 例に上がっているのは Googleのみ

社内が必要になった?? 🤔