ミニGCの作り方

中村 成洋

ネットワーク応用通信研究所

今日の話

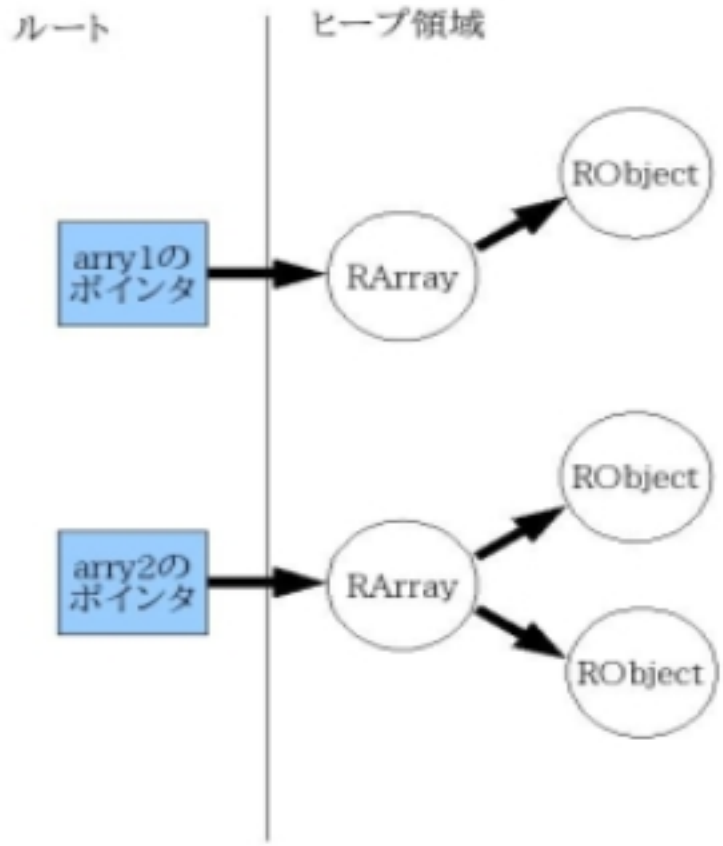
今日の話

- 簡単なGCの仕組み
- GC 分かれ道
- MiniGC の紹介
- malloc&free の実装
  - 省メモリ化の話
- mark&sweep の実装
  - 高速化の話

簡単なGCの仕組み

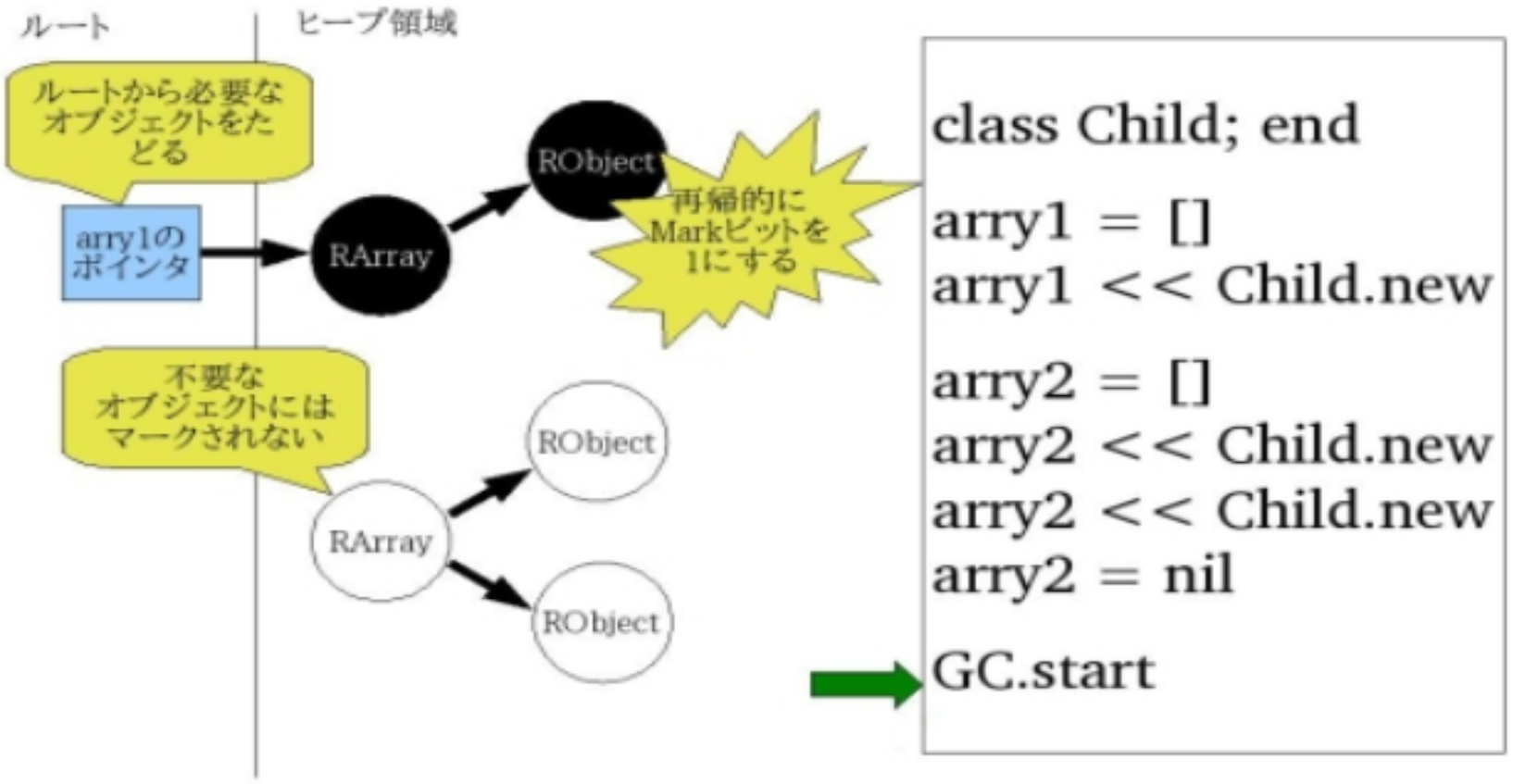
Mark&Sweep

ルート

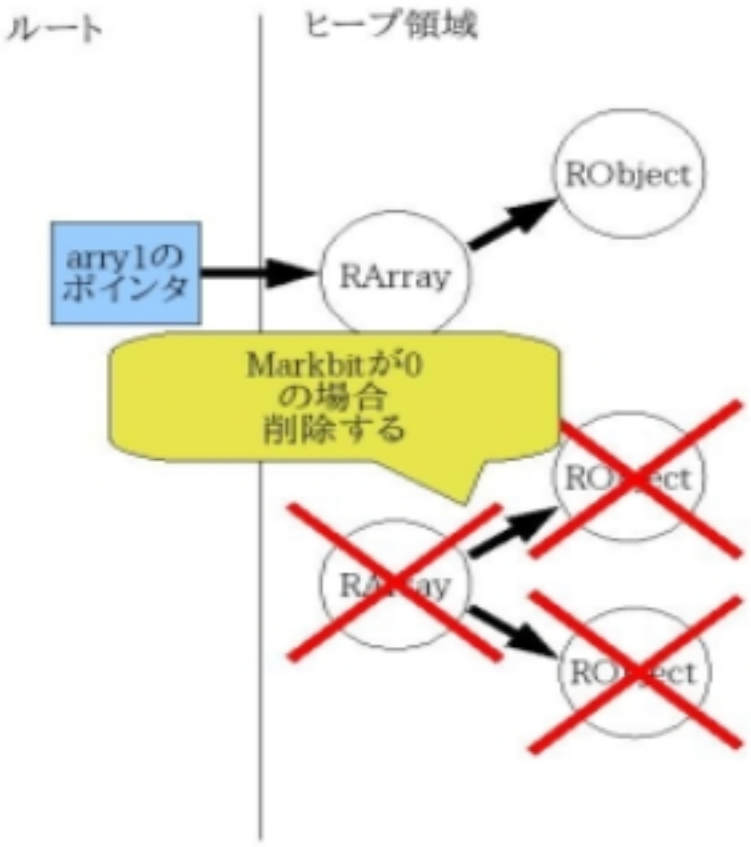



```
class Child; end

arry1 = []
arry1 << Child.new

arry2 = []
arry2 << Child.new
arry2 << Child.new
arry2 = nil

GC.start
```

Mark


ルート
ヒープ領域
ルートから必要なオブジェクトをたどる
arry1のポインタ
RArray
RObject
再帰的にMarkビットを1にする
不要なオブジェクトにはマークされない
RArray
RObject
RObject
class Child; end
arry1 = []
arry1 << Child.new
arry2 = []
arry2 << Child.new
arry2 << Child.new
arry2 = nil
GC.start

Sweep




```
class Child; end

arry1 = []
arry1 << Child.new

arry2 = []
arry2 << Child.new
arry2 << Child.new
arry2 = nil

GC.start
```

GC分かれ道

GC分かれ道

- GC対象領域の見つけ方
- メモリ領域の管理

GC対象領域の見つけ方

GC対象領域の見つけ方

- ConservativeGC
  - 茨の道
- PreciseGC
  - GCを作るのは簡単

ConservativeGC

- ユーザプログラムが現在直接さわれるメモリ領域(レジスタ、スタック、大域変数)をルートにする
  - a = [] <- こんなのがルートに入る
- 泥臭い事をやらないといけない
  - レジスタの値を取るためにsetjump使うとか
  - レジスタウィンドウ(SPARCとか)があったんだったとか

PreciseGC

- ルートは自分で完全に管理
  - 地道に追加
  - a = [] <- この時にaをルートへ追加
- ルートにはオブジェクトを指すポインタしかなくなる
- precise(正確)なGC

GCを作るのは簡単

- 不明瞭なルートがない(ambiguous roots)
  - CopyingGCが作れる
  - 空間効率もいい
- 泥臭い事をしなくてもいい

メモリ領域の管理

メモリ領域の管理

- 既存のmalloc使う
  - 簡単
- 自作malloc使う
  - 茨の道

既存のmallocを使う

- Rubyがやっている
- mallocでがばっと確保
- 確保したメモリ領域をGC対象に
- ゴミを見つけたらfree

この場合

- ライブラリのmalloc&freeの性能を信頼
- GCでもメモリ管理をするので二重管理
  - glibcでのフリーリスト
  - gc上でのフリーリスト
- GC屋は嫌うそうです（え，そうだったのか）
- 簡単でそこそこの性能

自作mallocを使う

- bdw-gc(boehm gc)他多くのGCライブラリが実装
- GC_MALLOCでは内部の自作mallocが動いてメモリ確保
- GCでは上記で確保した領域が対象となる

この場合

- mallocで管理している領域とGCで管理している領域が同じ
  - 二重管理ではない
- 性能は自作malloc次第
  - GC用に特別にチューンする事が可能

でも茨の道すぎ

- ライブラリのmallocはとても性能がいい
- 太刀打ちできない
  - 無理無理
- 環境依存関係の処理が膨大に
  - CPU
  - OS
  - ifdef地獄

MiniGCの紹介

MiniGCとは

- とても小さいGC
- ソースコード350行くらい
- 一応ConservativeなGC
- mallocも自作
  - GC茨道
- mark&sweepアルゴリズム

用途

- 読んで貰うためのGC

あと

- 私が作りました!

malloc&free の実装

- K&R malloc の焼き直し
- Headerにマークフラグを追加

K&R mallocの復習

- 前の発表の別のスライドへ

GC対象メモリ領域（GCヒープ）の作成

- sbrkしたものをリンクリストで保持

今後出来る省メモリ化

- Headerを小さく
- マークフラグをsizeの下位3bitに入れてしまう
  - 1Headerあたり4バイト削減
- free_listのリンクは割り当てたものから消してしまう
- コードが死ぬほど読みにくくなるのでやりませんが！

ヘッダを潰すのは良くやる手法

- minixのmallocはヘッダを潰しています
  - 実質のヘッダはポインタが一つだけ

mark and sweep の実装

特徴

- 関数スタックだけを走査
- 大域変数とかレジスタはやりません
  - やりたかったら add_rootsで指定

関数スタック上位と下位の取得

- プログラムを使う前にmini_gc_init()を呼び出す.
- gc前にスタックの一番上を取得
- スタックがどっちに伸びてるかは完全に環境依存
  - 適宜入れ替え

mark

- 以下の領域をルートとする
  - 関数スタック
  - add_rootsで定義した領域

sweep

- GCヒープ内を走査
- Headerを見ながらmini_gc_freeで解放していく

高速化

- マークの際のヘッダ探索
  - 今はリニアサーチ
  - O(n)
- 改善方法
  - treeを組む
  - ポインタで探索
  - O(1)でいける

終わり