

# mrubyの省メモリ化 について考える

(事前公開版)

2018/08/31 SWEST 20

まつもと ゆきひろ

高橋征義

山根ゆりえ

# 今回のあらすじ

- これまでのあらすじ
- mrubyの「初期化時」における省メモリ化
- 省メモリ化する方法
- 省メモリ化できそうなところ
  - シンボルとメソッド表
- それ以外の手法について（参加者の方からご意見をいただきたいです）

# これまでのあらすじ

- マイコンでmrubyを広めるには
- 国内外で入手しやすいマイコン上で動かす
  - 価格面、入手容易性
- （実行速度等より）メモリ消費が問題
  - 特にRAM

# 今回検討する メモリ消費タイミング

- 初期化時メモリ消費 ← 今回は主にこちら
- 実行時メモリ消費

# mrubyの初期化時に おける省メモリ化

# 「初期化時」とは

```
#include <mruby.h>
#include <mruby/compile.h>
```

```
int main(void) {
    mrbc_state *mrbc = mrbc_open();
    if (!mrbc) { /* handle error */ }
    mrbc_load_string(mrbc, "puts 'hello world'");
    mrbc_close(mrbc);
    return 0;
}
```

このタイミングで  
なんとかしたい

(こちらは対象外)

helloworld.c

# mrblib\_open()の概略

```
{
```

```
  mrblib_gc_init(mrb, &mrb->gc); /* GC */  
  mrblib_init_symtbl(mrb); /* シンボル表 */
```

```
  mrblib_init_class(mrb); /* 基盤(Class) */
```

```
  mrblib_init_object(mrb); /* 基盤(Object) */
```

```
  mrblib_init_kernel(mrb); /* 基盤(Kernel) */
```

```
  mrblib_init_XXXX(mrb); /* クラスライブラリ (C) */
```

```
  mrblib_init_mrblib(mrb); /* クラスライブラリ (Ruby) */
```

```
  mrblib_init_mrbgems(mrb); /* 拡張ライブラリ */
```

```
}
```

# 最適化できるのでは🤔

- やるべきことは事前に（実行前に）分かっている
- クラスライブラリ、拡張ライブラリも分かっている



静的な情報はROMに置いたりできそう



# なぜそうしていいなのか

- あまり意識していなかった？
  - アプリケーション組込みだとあまり関係なさそう
- Rubyの言語的な性質による

# Rubyの特徴

- 宣言がない
  - 変数の宣言はなく、代入で初期化される
  - クラスやメソッド定義も「式」として実行される
- クラスもオブジェクト
  - クラスはClassクラスのインスタンス
  - オープンクラス（実行中にもメソッド追加可能）

# Rubyと宣言と式

- 変数宣言はなく、代入で初期化される
- クラスやメソッド定義も「式」として実行される
- →「初期化」と「実行」の区別があまりない
  - 現状の実装でも「初期化」時に「実行」してる

# クラスもオブジェクト

- String、IOなどの各クラスは、Classクラスのインスタンスとして実装されている
- 各クラスを生成する段階で、オブジェクトを作ることになる
- クラス定義後にメソッドを追加することも可能
- 「初期化」時に決まらない

# 初期化の範囲

{

```
mrbc_gc_init(mrb, &mrbc->gc); /* GC */  
mrbc_init_symtbl(mrb); /* シンボル表 */
```

```
mrbc_init_class(mrb); /* 基盤(Class) */  
mrbc_init_object(mrb); /* 基盤(Object) */  
mrbc_init_kernel(mrb); /* 基盤(Kernel) */
```

普通に初期化

```
mrbc_init_XXXX(mrb); /* クラスライブラリ (C) */  
mrbc_init_mrbllib(mrb); /* クラスライブラリ (Ruby) */  
mrbc_init_mrbgems(mrb); /* 拡張ライブラリ */
```

}

# 初期化の範囲

```
{
```

```
  mrb_gc_init(mrb, &mrb->gc); /* GC */  
  mrb_init_symtbl(mrb); /* シンボル表 */
```

やや特殊な処理

```
  mrb_init_class(mrb); /* 基盤(Class) */  
  mrb_init_object(mrb); /* 基盤(Object) */  
  mrb_init_kernel(mrb); /* 基盤(Kernel) */
```

```
  mrb_init_XXXX(mrb); /* クラスライブラリ (C) */  
  mrb_init_mrblib(mrb); /* クラスライブラリ (Ruby) */  
  mrb_init_mrbgems(mrb); /* 拡張ライブラリ */
```

```
}
```

# 初期化の範囲

```
{
```

```
  mrb_gc_init(mrb, &mrb->gc); /* GC */  
  mrb_init_symtbl(mrb); /* シンボル表 */
```

```
  mrb_init_class(mrb); /* 基盤(Class) */
```

```
  mrb_init_object(mrb); /* 基盤(Object)
```

```
  mrb_init_kernel(mrb); /* 基盤(Kernel)
```

実行と変わらない

```
  mrb_init_XXXX(mrb); /* クラスライブラリ (C) */
```

```
  mrb_init_mrblib(mrb); /* クラスライブラリ (Ruby) */
```

```
  mrb_init_mrbgems(mrb); /* 拡張ライブラリ */
```

```
}
```

# なぜmrubyは初期化時にも メモリを消費するのか

- Rubyには素朴な意味での「初期化」があまりない
- クラス・メソッド定義を含め、すべては実行文（式）
  - 実行時に評価されるのと同様の挙動になる
- 素朴に実装すると、mruby処理系内でメソッド定義式と同等のC関数が実行されることになる





**Miura Hideki**

@miura1729

フォローする



Rubyは何か最適化をしようとする就先回りしてその最適化を阻む言語仕様が用意されているので、全ての言語実装における最適化を研究して仕様が作られた可能性がある

20:34 - 2018年8月7日

30件のリツイート 96件のいいね



3



30



96



<https://twitter.com/miura1729/status/1026793694991593472>

省メモリ化  
できそうなこと

# もっとROMを使いたい

- ROMを活用する
  - 事前に決定している情報についてはROMに配置させるようにする
  - 実行時に追加・変更される可能性があっても、初期化時の置き場所と実行時の置き場所を分けて、うまいこと辻褄があうようにする

# mrubyとROM

- すでに使っているもの
  - 文字列リテラルの確保(コンパイルオプションで最適化)
  - バイトコードの実体
    - ただし、初期化時にはバイトコードに対応するirep構造体が作られ、実行時にはこちらが使われる

# mrubyとROM

- 新たにROMに置きそうなもの
  - シンボル
  - メソッドテーブル
  - 他には？

# Rubyとシンボル

- シンボルは元々処理系の内部表現だった
- 最近は「変更できない(immutableな)文字列」 として使われたりする
- 低コストで操作できる（？ 最近は文字列も効率化されている？）

# khash.h

- mrubyのhashmap/hashset構造体操作マクロ群
- いわゆるハッシュマップのデータを管理する
  - アクセスは速い（衝突がなければ  $O(1)$  ）
  - メモリ消費量は大きい（余計なbucketsが必要）
- mruby内では汎用Key-Value Store的に多用されている
  - シンボルやメソッドテーブルにも使われている

# シンボルのROM化

- ソースに埋め込まれているシンボルをROMに入れる
- 事前に判別できる→khashも不要
- mrubyではlexerの作成にgperfを使っている
- これをシンボルにも利用し、名前→シンボルのインデックスもROMに置くようにする



# シンボルの検出

- ソースコードをスキャンする
  - Cの解析はpreprocessorも絡むので厳密には困難なので、正規表現でざっくり解析する
- mrbファイルを作って抽出
  - Cで実装した部分については使えない
- mrubyを実際に起動して生成されたシンボルを動的に抽出
  - ターゲットが組み込みのmrubyでは困難

# メソッド表のROM化

- 各クラスにメソッドが定義されている
- 実装としては、各クラスオブジェクト構造体にメソッド表へのリンクがある
- メソッド表の実体はkhashで実装
- 実行時に追加されることもある
  - が、稀な場合も多いのでROM化したい

# メソッド表のROM化

- メソッド表：メソッド名（シンボル）とメソッド定義の  
実体（アドレス）
  - ↑このためにもシンボルをROM化する必要があった
- 単なる配列か、バランス木等で実装できそう

(続きは当日にて)