

Lua の VM

Tsukuba.pm #3

びしょ〜じょ

May 14, 2016

こんにちは、びしょ〜じょです。

- ▶ ここの大学に4年滞在中の3年生
- ▶ 大学に来てからプログラミング始めたマン



Figure: 白い大きな花を咲かせている百合（ユリ）のイラストです。

こんにちは、びしょ〜じょです。

- ▶ ここの大学に 4 年滞在中の 3 年生
- ▶ 大学に来てからプログラミング始めたマン
- ▶ **Lua/MoonScript** をよく書く



Figure: 白い大きな花を咲かせている百合（ユリ）のイラストです。

Lua の VM

自己紹介

Lua とは

Lua/MoonScript とは

Lua とは?!

Lua VM の調査

Lua VM について

バイトコード

読み方

header block

function block

VM instructions

example

解析

まとめ

参考文献

Lua の VM

自己紹介

Lua とは

Lua/MoonScript とは

Lua とは?!

Lua VM の調査

Lua VM について

バイトコード

読み方

header block

function block

VM instructions

example

解析

まとめ

参考文献

▶ Lua

弱い動的型付けなスクリプト言語

ブラジル産!! 軽量!! 関数もファーストクラスで扱えるぞ!!

文法が簡単、予約語も 22 個と少ない

▶ MoonScript

JS に対する CoffeeScript みたいなやつ

end地獄の解消、リスト内包表記、クラスベース OOP など文法の強化

▶ Lua

弱い動的型付けなスクリプト言語 ← Perl では ?
ブラジル産!! 軽量!! 関数もファーストクラスで扱えるぞ!!
文法が簡単、予約語も 22 個と少ない

▶ MoonScript

JS に対する CoffeeScript みたいなやつ
end地獄の解消、リスト内包表記、クラスベース OOP など文法の強化

Perlでは.....

Lua の VM

自己紹介

Lua とは

Lua/MoonScript とは

Lua とは?!

Lua VM の調査

Lua VM について

バイトコード

読み方

header block

function block

VM instructions

example

解析

まとめ

参考文献

●Lua とは?!

- ▶ 軽量
- ▶ 関数がファーストクラス
- ▶ 唯一のデータ構造 table

*1 <https://github.com/nymphium/llix>
Lua とは.Lua とは?!

- ▶ 軽量

アイマス 2 から NetBSD、Wireshark や [このスライド](#) (Lua \LaTeX) にまで組み込める

- ▶ 関数がファーストクラス

- ▶ 唯一のデータ構造 table

*1 <https://github.com/nymphium/llix>
Lua とは.Lua とは?!

- ▶ 軽量

アイマス 2 から NetBSD、Wireshark やこのスライド (Lua \LaTeX) にまで組み込める

レジスタベース VM 上で動く (PUC-Lua)

- ▶ 関数がファーストクラス

- ▶ 唯一のデータ構造 table

*1 <https://github.com/nymphium/llix>
Lua とは.Lua とは?!

- ▶ 軽量

アイマス 2 から NetBSD、Wireshark やこのスライド (Lua_{La}T_EX) にまで組み込める

レジスタベース VM 上で動く (PUC-Lua)

- ▶ 関数がファーストクラス

高階関数を扱うことができ、CPS を用いた例外処理機構のある Lua インタプリタ^{*1} も実装できる

- ▶ 唯一のデータ構造 table

^{*1} <https://github.com/nymphium/llix>

▶ 軽量

アイマス 2 から NetBSD、Wireshark やこのスライド (Lua_{La}T_EX) にまで組み込める

レジスタベース VM 上で動く (PUC-Lua)

▶ 関数がファーストクラス

高階関数を扱うことができ、CPS を用いた例外処理機構のある Lua インタプリタ^{*1} も実装できる

▶ 唯一のデータ構造 table

- ▶ 簡単に言うと連想配列、数字もキー
- ▶ 関数、数値、文字列、table 自身も、オブジェクトは全部詰め込める
- ▶ メタテーブルという機能を用いることにより演算子オーバーロードやプロトタイプベースのオブジェクト指向プログラミングができる

^{*1} <https://github.com/nymphium/llix>

▶ 軽量

アイマス 2 から NetBSD、Wireshark やこのスライド (LuaL^AT_EX) にまで組み込める

レジスタベース VM 上で動く (PUC-Lua)

▶ 関数がファーストクラス

高階関数を扱うことができ、CPS を用いた例外処理機構のある Lua インタプリタ^{*1} も実装できる

▶ 唯一のデータ構造 table

- ▶ 簡単に言うと連想配列 ← Perl では ?
- ▶ 関数、数値、文字列、table 自身も、オブジェクトは全部詰め込める
- ▶ メタテーブルという機能を用いることにより演算子オーバーロードやプロトタイプベースのオブジェクト指向プログラミングができる

^{*1} <https://github.com/nymphium/llix>

Perlでは.....

Perlでは..... ない

Lua の VM

自己紹介

Lua とは

Lua/MoonScript とは

Lua とは?!

Lua VM の調査

Lua VM について

バイトコード

読み方

header block

function block

VM instructions

example

解析

まとめ

参考文献

Lua の VM

自己紹介

Lua とは

Lua/MoonScript とは

Lua とは?!

Lua VM の調査

Lua VM について

バイトコード

読み方

header block

function block

VM instructions

example

解析

まとめ

参考文献

- ▶ リオデジャネイロ・カトリカ大学の開発した Lua 処理系 (つまり本家, PUC-Lua) の VM

ソース → バイトコード → VM で実行

- ▶ リオデジャネイロ・カトリカ大学の開発した Lua 処理系 (つまり本家, PUC-Lua) の VM

ソース → バイトコード → VM で実行
の調査と解析

- ▶ リオデジャネイロ・カトリカ大学の開発した Lua 処理系 (つまり本家, PUC-Lua) の VM

ソース → バイトコード → VM で実行

の調査と解析

Lua5.2 を見る (Lua \LaTeX が v5.2 ベースなので)

1 環境 (メインチャンク、関数クロージャ) ごとにレジスタ (slots, constants, upvalues, locals, functions) を用意

```
function hello()  
  print("hello")  
end
```

slots:	2 個	(<code>print</code> と <code>"hello"</code> をセットするレジスタ)
constants:	2 個	(<code>"print"</code> 、 <code>"hello"</code>)
upvalues:	1 個	(<code>_ENV</code> ^{*2})
functions:	0 個	
locals:	0 個	

^{*2} グローバル変数は `_ENV` table から持ってくるため
Lua VM の調査: Lua VM について

Lua の VM

自己紹介

Lua とは

Lua/MoonScript とは

Lua とは?!

Lua VM の調査

Lua VM について

バイトコード

読み方

header block

function block

VM instructions

example

解析

まとめ

参考文献

● バイトコード

- ▶ `luac` コマンドでバイトコードを生成
- ▶ `luac -l bytecode.out` でいい感じに出力
- ▶ `luac -l -l bytecode.out` で情報が増える

● バイトコード

ひとまず標準入力から渡してみる

```
cat <<LUACODE | luac -l -  
print("hello, world")  
LUACODE
```

ひとまず標準入力から渡してみる

```
cat <<LUACODE | luac -l -  
print("hello, world")  
LUACODE
```

↓

```
main <stdin:0,0> (4 instructions at 0x2268790)  
0+ params, 2 slots, 1 upvalue, 0 locals, 2 constants, 0 functions  
  1      [1]   GETTABUP      0 0 -1 ; _ENV "print"  
  2      [1]   LOADK         1 -2 ; "hello, world"  
  3      [1]   CALL          0 2 1  
  4      [1]   RETURN        0 1
```

LuaL^AT_EX にダイレクトに書いてみる

Code 1: codes/texutils/dumpdemo.lua

```
1 tex.print([==[\begin{lstlisting}[numbers=none]]==])
2 local function foo() end -- この関数内の環境を書き出す
3 local LEN = 31
4 local acc, cnt = "", 0
5 for c in string.dump(foo):gmatch(".") do
6     acc = acc .. ("%02X"):format(c:byte())
7     cnt = cnt + 1
8     if cnt % LEN < 1 then
9         tex.print(acc)
10        acc = ""
11    else acc = acc .. " "
12    end
13 end
14 if #acc > 0 then tex.print(acc) end
15 tex.print([[\end{lstlisting}]])
```

Lua_{La}T_EX にダイレクトに書いてみる

Code 1: codes/texutils/dumpdemo.lua

```
1 tex.print([==[\begin{lstlisting}[numbers=none]]==])
2 local function foo() end -- この関数内の環境を書き出す
3 local LEN = 31
4 local acc, cnt = "", 0
5 for c in string.dump(foo):gmatch(".") do
6     acc = acc .. ("%02X"):format(c:byte())
7     cnt = cnt + 1
8     if cnt % LEN < 1 then
9         tex.print(acc)
10        acc = ""
11    else acc = acc .. " "
12    end
13 end
14 if #acc > 0 then tex.print(acc) end
15 tex.print([[\end{lstlisting}]])
```

```
1B 4C 75 61 52 00 01 04 08 04 08 00 19 93 0D 0A 1A 0A 02 00 00 00 02 00 00 00 00 02 01 00
00 00 1F 00 80 00 00 00 00 00 00 00 00 00 00 00 00 00 1D 00 00 00 00 00 00 00 40 63 6F 64 65
73 2F 74 65 78 75 74 69 6C 73 2F 64 75 6D 70 64 65 6D 6F 2E 6C 75 61 00 01 00 00 00 02 00 00
00 00 00 00 00 00 00 00 00
```

Lua の VM

自己紹介

Lua とは

Lua/MoonScript とは

Lua とは?!

Lua VM の調査

Lua VM について

バイトコード

読み方

header block

function block

VM instructions

example

解析

まとめ

参考文献

```
1B 4C 75 61 52 00 01 04 08 04 08 00 19 93 0D 0A 1A 0A 02 00 00 00 02 00 00 00 00 00 02 01 00
00 00 1F 00 80 00 00 00 00 00 00 00 00 00 00 00 00 00 1D 00 00 00 00 00 00 00 40 63 6F 64 65
73 2F 74 65 78 75 74 69 6C 73 2F 64 75 6D 70 64 65 6D 6F 2E 6C 75 61 00 01 00 00 00 02 00 00
00 00 00 00 00 00 00 00
```

まずこれを読む。

●header block

1B 4C 75 61 52 00 01 04 08 04 08 00 19 93 0D 0A 1A 0A

header block

●header block

1B 4C 75 61 52 00 01 04 08 04 08 00 19 93 0D 0A 1A 0A

header block

▶ 4bytes

header signature (ESC "Lua")

●header block

1B 4C 75 61 52 00 01 04 08 04 08 00 19 93 0D 0A 1A 0A

header block

- ▶ 4bytes
header signature (ESC "Lua")
- ▶ 1byte
Lua Version
(例では 52、つまり Lua 5.2)

*3 32bit integer

●header block

1B 4C 75 61 52 00 01 04 08 04 08 00 19 93 0D 0A 1A 0A

header block

- ▶ 4bytes
header signature (ESC "Lua")
- ▶ 1byte
Lua Version
(例では 52、つまり Lua 5.2)
- ▶ 1byte
Format Version
(0 = official(default))

●header block

1B 4C 75 61 52 00 01 04 08 04 08 00 19 93 0D 0A 1A 0A

header block

- ▶ 4bytes
header signature (ESC "Lua")
- ▶ 1byte
Lua Version
(例では 52、つまり Lua 5.2)
- ▶ 1byte
Format Version
(0 = official(default))
- ▶ 1byte
Endianness flag
(1 = little endian(default), 0 = big endian)

*3 32bit integer

●header block

1B 4C 75 61 52 00 01 04 08 04 08 00 19 93 0D 0A 1A 0A

header block

- ▶ 4bytes
header signature (ESC "Lua")
- ▶ 1byte
Lua Version
(例では 52、つまり Lua 5.2)
- ▶ 1byte
Format Version
(0 = official(default))
- ▶ 1byte
Endianness flag
(1 = little endian(default), 0 = big endian)
- ▶ 1byte
size of `int` (C lang)

*3 32bit integer

●header block

1B 4C 75 61 52 00 01 04 08 04 08 00 19 93 0D 0A 1A 0A

header block

- ▶ 4bytes
header signature (ESC "Lua")
- ▶ 1byte
Lua Version
(例では 52、つまり Lua 5.2)
- ▶ 1byte
Format Version
(0 = official(default))
- ▶ 1byte
Endianness flag
(1 = little endian(default), 0 = big endian)
- ▶ 1byte
size of `int` (C lang)
- ▶ 1byte
size of `size_t` (C lang)

*3 32bit integer

1B 4C 75 61 52 00 01 04 08 04 08 00 19 93 0D 0A 1A 0A

header block

- ▶ 4bytes
header signature (ESC "Lua")
- ▶ 1byte
Lua Version
(例では 52、つまり Lua 5.2)
- ▶ 1byte
Format Version
(0 = official(default))
- ▶ 1byte
Endianness flag
(1 = little endian(default), 0 = big endian)
- ▶ 1byte
size of `int` (C lang)
- ▶ 1byte
size of `size_t` (C lang)
- ▶ 1byte
size of instruction

*3 32bit integer

1B 4C 75 61 52 00 01 04 08 04 08 00 19 93 0D 0A 1A 0A

header block

- ▶ 4bytes
header signature (ESC "Lua")
- ▶ 1byte
Lua Version
(例では 52、つまり Lua 5.2)
- ▶ 1byte
Format Version
(0 = official(default))
- ▶ 1byte
Endianness flag
(1 = little endian(default), 0 = big endian)
- ▶ 1byte
size of `int` (C lang)
- ▶ 1byte
size of `size_t` (C lang)
- ▶ 1byte
size of instruction
- ▶ 1byte
size of `lua_Number`^{*3}(C lang)

*3 32bit integer

1B 4C 75 61 52 00 01 04 08 04 08 00 19 93 0D 0A 1A 0A

header block

- ▶ 4bytes
header signature (ESC "Lua")
- ▶ 1byte
Lua Version
(例では 52、つまり Lua 5.2)
- ▶ 1byte
Format Version
(0 = official(default))
- ▶ 1byte
Endianness flag
(1 = little endian(default), 0 = big endian)
- ▶ 1byte
size of `int` (C lang)
- ▶ 1byte
size of `size_t` (C lang)
- ▶ 1byte
size of instruction
- ▶ 1byte
size of `lua_Number`^{*3}(C lang)
- ▶ 1byte
Integral flag
(0 = floating point(default), 1 = integral number type)

*3 32bit integer

1B 4C 75 61 52 00 01 04 08 04 08 00 19 93 0D 0A 1A 0A

▶ 6bytes

LUAC_DATA, “data to catch conversion errors”(ソースのコメントより)

▶ 19 93

Lua1.0 のリリース年

▶ 0D 0A

DOS の改行 (CR LF)、DOS→UNIX での改行のデータ変換の検知

▶ 1A

SUB

▶ 0A

UNIX の改行 (LF)、UNIX→DOS での改行のデータ変換の検知

ちなみに
Lua5.2

1B 4C 75 61 52 00 01 04 08 04 08 00 19 93 0D 0A 1A 0A



Lua5.3

header signature		version	format	version	LUAC_DATA				objects sizes								
1B	4C	75	61	53	00	19	93	0D	0A	1A	0A	04	08	04	08	08	
				LUAC_INT* ⁴					LUAC_NUM* ⁵								
78	56	00	00	00	00	00	00	00	00	00	00	00	00	00	28	77	40

*4 Endianness の検査、0x5678 を dump しているのでこの場合リトルエンディアン

*5 IEEE754 float format の検査

ちなみに
Lua5.2

1B 4C 75 61 52 00 01 04 08 04 08 00 19 93 0D 0A 1A 0A



Lua5.3

header signature	version	format	version	LUAC_DATA	objects sizes
1B 4C 75 61	53	00	19 93 0D 0A	1A 0A	04 08 04 08 08
LUAC_INT* ⁴			LUAC_NUM* ⁵		
78 56 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	28 77 40

長い

*4 Endianness の検査、0x5678 を dump しているのでこの場合リトルエンディアン

*5 IEEE754 float format の検査

02 00 00 00 02 00 00 00 00 00 02 01 00.....
function block

- ▶ `int`
line defined
- ▶ `int`
last line defined
- ▶ 1byte
number of parameters
- ▶ 1byte
`is_vararg`
- ▶ 1byte
number of registers used by the function
- ▶ List
number/list of instructions
- ▶ List
number/list of constants
- ▶ List
number/list of upvalues
- ▶ List
debug info

02 00 00 00 02 00 00 00 00 00 02 01 00.....
function block

- ▶ `int`
line defined
- ▶ `int`
last line defined
- ▶ 1byte
number of parameters
- ▶ 1byte
`is_vararg`
- ▶ 1byte
number of registers used by the function
- ▶ List
number/list of instructions
- ▶ List
number/list of constants
- ▶ List
number/list of upvalues
- ▶ List
debug info

header block

1B	4C	75	61	52	00	01	04	08	04	08	00	19	93	0D	0A	1A	0A	02	00	00	00	02	00	00	00	00	00	02	01	00
00	00	1F	00	80	00	00	00	00	00	00	00	00	00	00	00	00	00	1D	00	00	00	00	00	00	00	40	63	6F	64	65
73	2F	74	65	78	75	74	69	6C	73	2F	64	75	6D	70	64	65	6D	6F	2E	6C	75	61	00	01	00	00	00	02	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

function block

Lua の VM

自己紹介

Lua とは

Lua/MoonScript とは

Lua とは?!

Lua VM の調査

Lua VM について

バイトコード

読み方

header block

function block

VM instructions

example

解析

まとめ

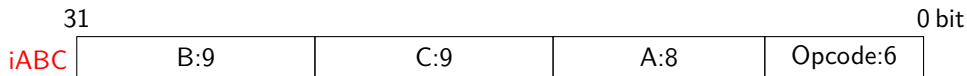
参考文献

32bit、3つの形式

32bit、3つの形式

▶ iABC

opcode R(A) R(B) R(C)



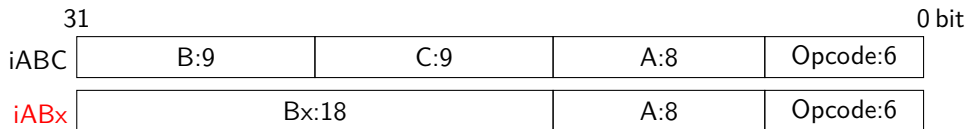
32bit、3つの形式

▶ iABC

opcode R(A) R(B) R(C)

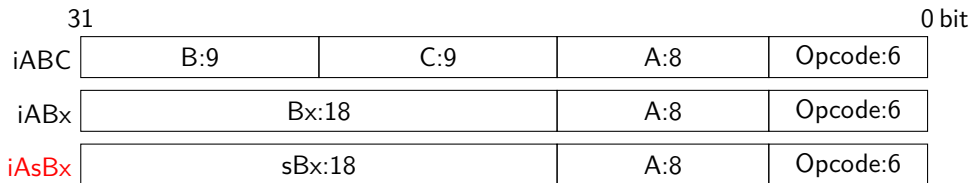
▶ iABx

opcode R(A) (unsigned integer)Bx



32bit、3つの形式

- ▶ iABC
opcode R(A) R(B) R(C)
- ▶ iABx
opcode R(A) (unsigned integer)Bx
- ▶ iAsBx
opcode R(A) (signed integer)Bx



Q. この命令は?

1F 00 80 00 (funciton foo())より)

Q. この命令は?

1F 00 80 00 (funciton foo())より)

1. endian を考慮しながら bit にする

Q. この命令は?

1F 00 80 00 (funciton foo()より)

1. endian を考慮しながら bit にする
ヘッダの情報よりリトルエンディアン

1F 00 80 00 → 00 80 00 1F
↓

Q. この命令は?

1F 00 80 00 (funciton foo()より)

1. endian を考慮しながら bit にする
ヘッダの情報よりリトルエンディアン

1F 00 80 00 → 00 80 00 1F

↓

00000000 10000000 00000000 00 011111

Q. この命令は?

1F 00 80 00 (funciton foo())より)

1. endian を考慮しながら bit にする
ヘッダの情報よりリトルエンディアン

1F 00 80 00 → 00 80 00 1F

↓

0000000 10000000 00000000 00 011111

Q. この命令は?

1F 00 80 00 (funciton foo()より)

1. endian を考慮しながら bit にする
ヘッダの情報よりリトルエンディアン

1F 00 80 00 → 00 80 00 1F

↓

 0000000 10000000 00000000 00011111 ^{RETURN}

2. Opcode 0b11111 は RETURN

Q. この命令は?

1F 00 80 00 (funciton foo()より)

1. endian を考慮しながら bit にする
ヘッダの情報よりリトルエンディアン

1F 00 80 00 → 00 80 00 1F

↓

0000^B000
1000^C000
0000^A000
00
11111^{RETURN}

2. Opcode 0b11111 は RETURN
3. RETURN は iABC 形式 (ただし R(C) は not used)

Q. この命令は?

1F 00 80 00 (funciton foo()より)

1. endian を考慮しながら bit にする
ヘッダの情報よりリトルエンディアン

1F 00 80 00 → 00 80 00 1F

↓

B
 00000000 C 10000000 A 00000000 00 RETURN 011111

2. Opcode 0b111111 は RETURN
3. RETURN は iABC 形式 (ただし R(C) は not used)

A. RETURN 0 1 (0)

チャンク終わりは必ずコレなので読むときに便利!!

Q. この命令をバイトコードに直すと? (little endian)

CALL 0 2 1

Q. この命令をバイトコードに直すと? (little endian)

CALL 0 2 1

1. CALL は Opcode 0b11101

Q. この命令をバイトコードに直すと? (little endian)

CALL 0 2 1

1. CALL は Opcode 0b11101
2. iABC 形式なので、32bit で以下のようなになる

0000^B001 00000^C00 010000^A00 000^{CALL}11101

Q. この命令をバイトコードに直すと? (little endian)

CALL 0 2 1

1. CALL は Opcode 0b11101
2. iABC 形式なので、32bit で以下のようなになる

0000^B001 00000^C00 010000^A00 000^{CALL}11101

3. little endian で 16 進数にしておわり

Q. この命令をバイトコードに直すと? (little endian)

CALL 0 2 1

1. CALL は Opcode 0b11101
2. iABC 形式なので、32bit で以下のようなになる

0000^B001 0000^C000 01000^A00 00^{CALL}11101

3. little endian で 16 進数にしておわり

A. 1d 40 00 01

Lua の VM

自己紹介

Lua とは

Lua/MoonScript とは

Lua とは?!

Lua VM の調査

Lua VM について

バイトコード

読み方

header block

function block

VM instructions

example

解析

まとめ

参考文献

かいた

<https://gist.github.com/Nymphium/d47385929fd0d23e98207670f9c588c3>

- ▶ Lua 5.2 バイトコードをターゲット
- ▶ header/function block の constants まで解析
- ▶ MoonScript で実装

バグを一つみつけてしまった

- ▶ 5.1 のバイトコードの資料見ながら実装していったら 5.2 とちょこちょこ変わっててホンマキレタ
- ▶ 5.2 と 5.3 もちょいちょい違っててキレタ (2)

デモ

1. header block から情報を読み取る
2. 読み取った情報に基づき functoin block を読み取る
だけ

できそうなこと

- ▶ 最適化
最適化っぽいことはほとんどしない (TAICALL、number の即値演算程度) ので
いっばいできそう
- ▶ 型チェック
constants pool から逆に命令を見ていけばいけるか
- ▶ 可視化 (フローグラフなど)

Lua の VM

自己紹介

Lua とは

Lua/MoonScript とは

Lua とは?!

Lua VM の調査

Lua VM について

バイトコード

読み方

header block

function block

VM instructions

example

解析

まとめ

参考文献

- ▶ Lua MV のバイトコードが読める
- ▶ Lua はバイナリ解析だってできる
- ▶ Lua の気持ちが理解できる
- ▶ わりとバージョンの互換がない
- ▶ Lua はまだまだ高速化できる部分がある

- ▶ Lua 5.2 Bytecode and Virtual Machine (Web site)

<http://files.catwell.info/misc/mirror/lua-5.2-bytecode-vm-dirk-laurie/lua52vm.html>

- ▶ A No-Frills Introduction to Lua 5.1 VM Instructions (pdf)

<http://luaforge.net/docman/83/98/ANoFrillsIntroToLua51VMInstructions.pdf>

- ▶ Source code

- ▶ on web

<http://www.lua.org/source/5.2/>

- ▶ tar

<https://www.lua.org/ftp/lua-5.2.4.tar.gz>

<https://www.lua.org/ftp/lua-5.3.2.tar.gz> (stable latest)

おわり