SmartHR

PHPで“本気で”🔥 WebAssemblyを動かす方法

PHPカンファレンス福岡2025 @ 2025.11.8

株式会社 SmartHR
Uchio Kondo

自己紹介

- Uchio Kondo
- 普段はRubyを書いています
- 好きなカレー: Tiki
- 所属: SmartHR
  (人事労務/タレントマネジメントSaaS)

最初にちょっとだけ宣伝します！
すいません！！！１

来年2月にも、
博多で熱いイベントがあるのを
ご存知ですか…？

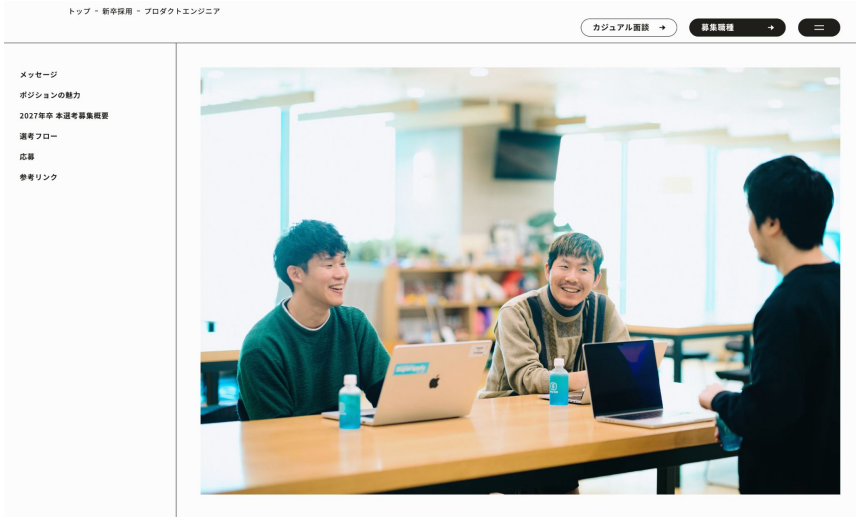
https://regional.rubykaigi.org/fukuoka05/

宣伝まとめ

- 福岡Rubyist会議05 (2026.2.28 @ 博多)
- CfP is open now!!!!!!!!! (~ 11.18)
- 来年も福岡でワイワイやりましょう！
- **Rubyistでなくても**、""""やってる人"""" はしゃべってほしい！

宣伝 #2

- SmartHRでは新卒エンジニアを採用中！
- 福岡からもフルリモートOK！
- 選考もリモート可能！



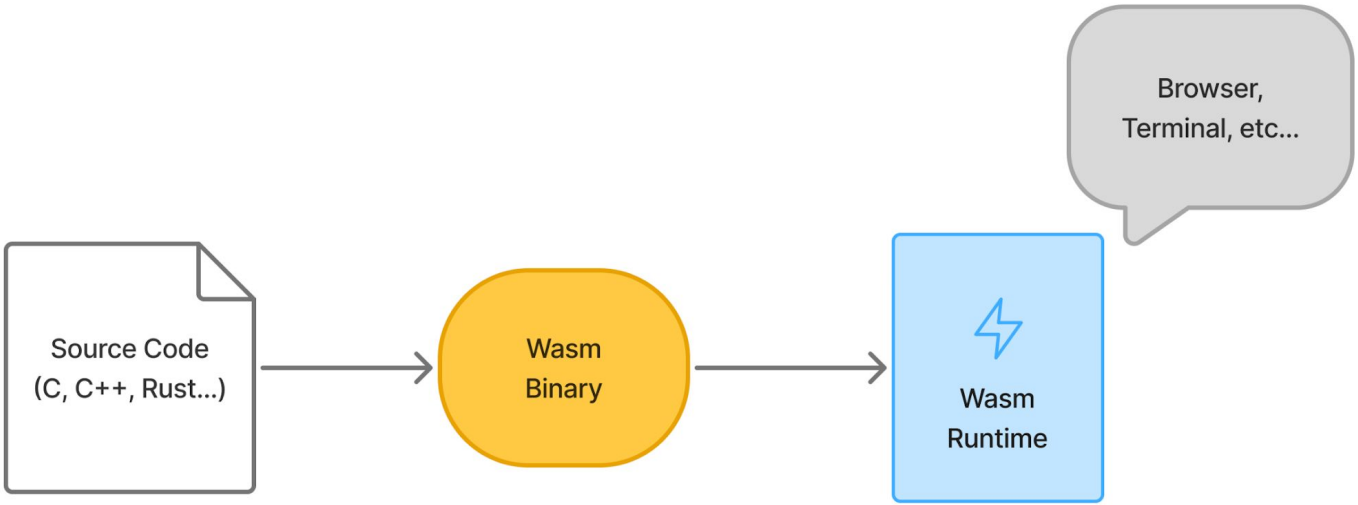
https://recruit.smarthr.co.jp/newgrads/product-engineer/

今日のテーマは WebAssembly

WebAssembly (Wasm) とは？

- **様々な言語をブラウザ上で動かすための技術**
  - C/C++、Rust、Go、Python、Ruby などをサポート
- **仕組み:**
  - 各言語のコードをWasm形式のバイナリーにコンパイル
  - そのバイナリーをブラウザ内で実行

WebAssembly Execution Flow



https://udzura.jp/slides/2025/rubykaigi/#16

Why WebAssembly?

- **メリット:**
  - JavaScript以外の言語をブラウザで利用可能
  - コンパイルによる**高速化や最適化**が期待できる

Wasmの実行環境はブラウザだけではない

- Wasmバイナリーの形式にすれば、どこでも動かせる:
  - ブラウザ内
  - ターミナル (CLI)
  - 組み込み環境、ミドルウェア内 (例: Envoy)...

「どんな言語でも書けるし、どんな場所でも動かせる」

Wasmの様々な実行環境の実装例

- Wasmは**他の言語に組み込んで動かす**使い方ができる
- **一般的な実装:**
  - CやRustで書かれたランタイムを使う
  - ネイティブライブラリやFFI (外部関数インターフェース) を使って組み込むパターンが多い
  - Wasmで他言語のプラグインが書ける

純粋に特定言語で書かれたランタイムの例

- **Go言語 (Wazero):**
  - 純粋なGoだけで組まれたWasmランタイム
  - Cコードを使わず、Goから動的にWasmプラグインを読み込ませることが可能
- **Ruby (Wardite; 拙作):**
  - WarditeはRubyで書かれたWasmランタイム

PHPでWasmを動かすには？

- **方法1: C拡張もしくはFFIを使う**
  - Cで書かれたWasmランタイムをPHPから呼び出す
    - ネイティブ拡張が必須になる
  - 技術的には可能だが、運用や実装が面倒になることが多い

PHPでWasmを動かすには？

- **方法2: PHPで純粋なVMを実装する**
  - PHPだけでWasmバイナリーをパースし、実行するVM（仮想マシン）を実装する...

え!!　PHPだけでWasm VMを!?

できらぁ！（画像略）

本発表のゴール:
PHPでWasm VMを実装し、
WasmをPHP上で動かす！

Wasm VM実装に必要な知識

VM（仮想マシン）の基本的な概念

- **VMとは？**
  - 言語処理における**抽象レイヤー（インターフェース）**
  - 様々な環境でコードを動かすために導入される (例: Java、Ruby、Python、PHPのZend VM)

YARV Maniacs 【第 2 回】 VM ってなんだろう

初稿：2005-06-19

書いた人: ささだ

はじめに

YARV Maniacs の第 2 回です。前回 (YARV Maniacs 【第 1 回】 『Ruby ソースコード完全解説』不完全解説) は RHG の紹介という ~~手抜き~~ 工夫をしたおかげで結構簡単に書けたのですが、早速 2 回目からネタに詰まりました。連載開始前はソースコードの細かいところを逐一解説して、「だから高速化されてるんですよ」ということを示そうと思っていたのですが、それだとあまりにマニアックだし、あまりに興味を持つ人は少ないだろうし、あまりにそもそも理解してくれる人が少なそうなので、どうしようかなぁ、と。

というわけで、もうちょっと簡単なところから解説していこうかと思っています。という建前で、本当のところは簡単なことじゃないと説明できそうにないからなんですが。

で、なんで仮想化するかということですが、まぁ、ぶっちゃけ便利になるからですね。

... 具体的な何かに依存するよりは、中間層を設けることによって別々のものを扱いやすくしましょう、というのが仮想化です。中間層により、上層で利用することのできるインターフェースを共通化することで利用しやすくしましょうね、ということです。

『VM ってなんだろう』 より引用

VM（仮想マシン）の基本的な概念

- **VMの構成要素:**
  - **命令のセット:** VMが解釈できる命令 (例: `i32.add`, `i32.const`)
  - **実行機構（エバリュエーター/評価器）:** 命令を解釈し、処理を行う

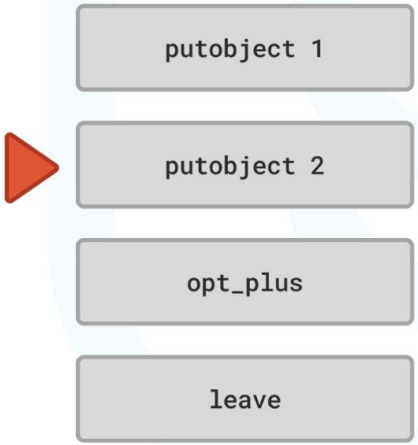
Wasm VMは「スタックマシン」

- WasmのVMは**スタックマシン**を採用している
- 多くのVM (PHP Zend VM, Java, Ruby...) もスタックマシンを採用

スタックマシンとは？

- 演算の対象値を **スタック（LIFO構造）** で管理するVM
- **命令の実行プロセス例:**
  - `Const 10` → スタックに10を積む
  - `Const 20` → スタックに20を積む
  - `Add` → スタックから20と10を取り出す (Pop)
  - 計算 (`20 + 10 = 30`)
  - 結果30をスタックに戻す (Push)

VMのスタックの状態を観測



※ Rubyのコード経由で確認することはできないので、あくまで模式図

https://docs.google.com/presentation/d/1W5H9NkaF6rDWWF387XXHMMMTs9GU1dDcCCCC1n5aaDs/edit?usp=sharing

PHPでWasm風命令を動かす概念プログラム

- **命令をPHPの文字列の配列で表現:**
- **スタックもPHPの配列で表現:**
  - `i32.const 10` 実行後: `[10]`
  - `i32.const 20` 実行後: `[10, 20]`
  - `i32.add` 実行後: `10, 20` を取り出し計算、
    `30` をスタックに戻す: `[30]`
- 最終的なスタックのトップ（30）が返り値

実装コード: スタック等の定義

```php
class VM {
    private $stack = [];
    private $instructions = [];
    private $pc = 0; // program counter

    public function load($instructions): void {
        $this->instructions = $instructions;
    }

    public function run(): void {
        // instruction loop:
        while ($this->pc < count($this->instructions)) {
            $this->evaluate($this->instructions[$this->pc]);
            $this->pc++;
        }
        print("Final Stack: [" . implode(', ', $this->stack) . "]\n");
    }
    // ...
}
```

実装コード: 命令を評価する本体

```php
class VM {
    private function evaluate($instruction): void {
        list($op, $arg) = explode(' ', $instruction . ' ', 2);
        switch ($op) {
            case 'i32.const':
                $this->stack[] = (int)$arg;
                break;
            case 'i32.add':
                $b = array_pop($this->stack);
                $a = array_pop($this->stack);
                $this->stack[] = $a + $b;
                break;
            default:
                throw new Exception("Unimplemented or unsupported instruction: $op");
        }
    }
}
```

このVMを動かしてみる

```php
$vm = new VM();
$vm->load([
    'i32.const 10',
    'i32.const 20',
    'i32.add',
]);
$vm->run();
```

結果

```
$ php ./samplevm/main.php
Final Stack: [30]
```

Wasm VMを学ぶ

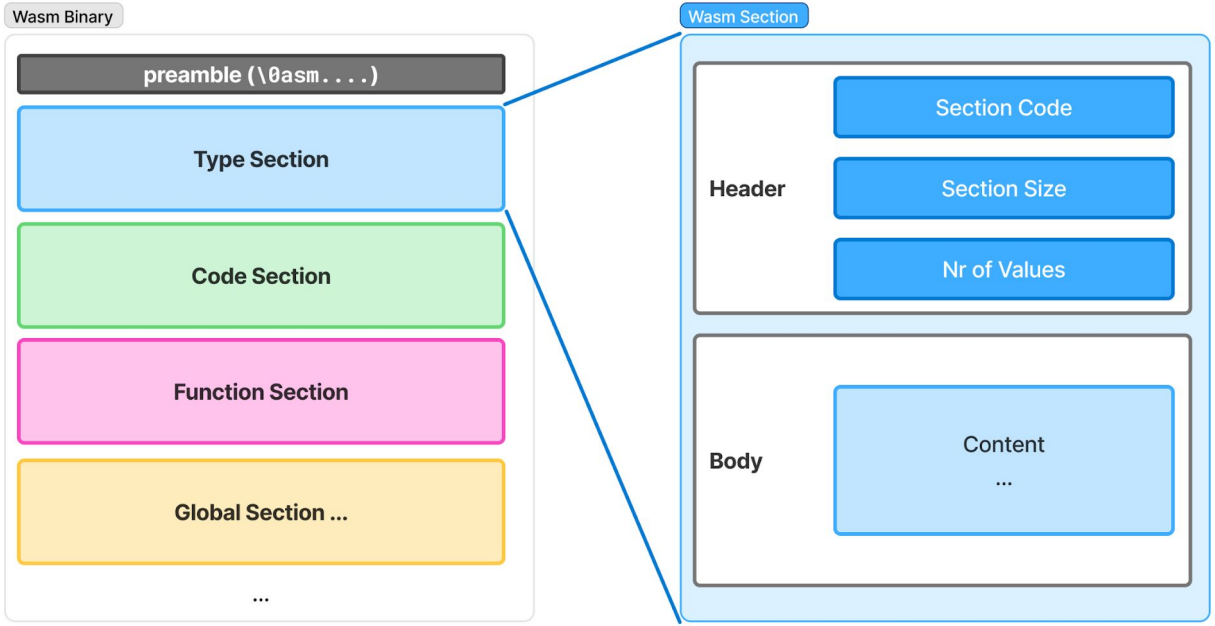
Wasm VMを実装するための学習リソース

- Wasmの完全な実装は複雑でボリュームがある
- 「RustでWasm Runtimeを実装する」(ゴリラさんによる本) がある
  - RustでWasmの最小実装を解説している
  - Wasmの仕様に忠実な設計を採用

Wasm実装のポイント (1): バイナリーフォーマット

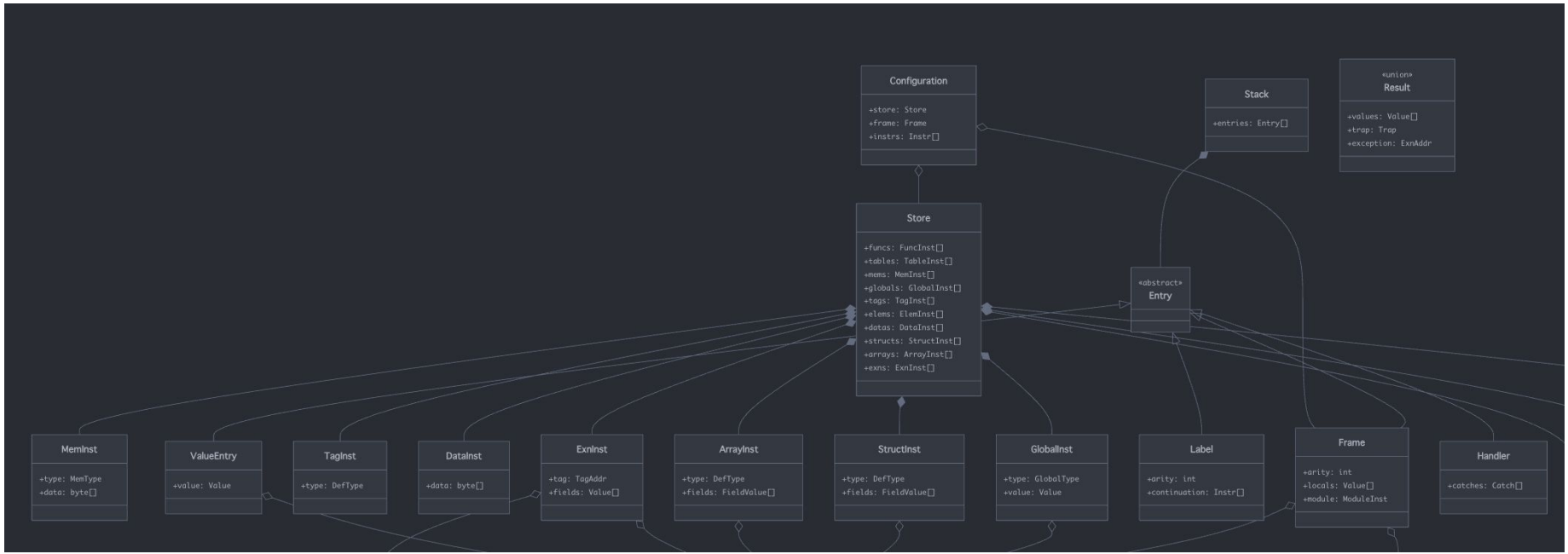
- Wasmファイルは**バイナリーフォーマット**を持っている
- **構造:** 複数の**セクション**が並んで構成されている

Binary Format Overview



Wasm実装のポイント (2): 内部設計と命令

- Wasmの仕様にはVMの**内部アーキテクチャ**に関する記述がある
- **命令セット:** Wasmには約200種類の命令がある
- まずはゴリラ本で紹介されている**最低限の命令セット**から実装


Configuration
+store: Store
+frame: Frame
+instrs: Instr[]
Stack
+entries: Entry[]
«union»
Result
+values: Value[]
+trap: Trap
+exception: ExnAddr
Store
+funcs: FuncInst[]
+tables: TableInst[]
+mems: MemInst[]
+globals: GlobalInst[]
+tags: TagInst[]
+elems: ElemInst[]
+datas: DataInst[]
+structs: StructInst[]
+arrays: ArrayInst[]
+exns: ExnInst[]
«abstract»
Entry
MemInst
+type: MemType
+data: byte[]
ValueEntry
+value: Value
TagInst
+type: DefType
DataInst
+data: byte[]
ExnInst
+tag: TagAddr
+fields: Value[]
ArrayInst
+type: DefType
+fields: FieldValue[]
StructInst
+type: DefType
+fields: FieldValue[]
GlobalInst
+type: GlobalType
+value: Value
Label
+arity: int
+continuation: Instr[]
Frame
+arity: int
+locals: Value[]
+module: ModuleInst
Handler
+catches: Catch[]

WebAssembly Opcodes

by Pengo Wray

WebAssembly is an open, industry-wide effort to bring a safe, efficient assembly language to the web. WebAssembly technology is developed collaboratively by major browser vendors including Mozilla, Google, Microsoft, and Apple. WebAssembly modules can be downloaded and executed by the majority of browsers in use today.

Toggle Dark Mode

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	unreachable	nop	block	loop	if	else	try	catch	throw	rethrow	throw_ref	end	br	br_if	br_table	return
1_	call	call_indirect	return_call	return_call_indirect	call_ref	return_call_ref			delegate	catch_all	drop	select	select t			try_table
2_	local.get	local.set	local.tee	global.get	global.set	table.get	table.set		i32.load	i64.load	f32.load	f64.load	i32.load8_s	i32.load8_u	i32.load16_s	i32.load16_u
3_	i64.load8_s	i64.load8_u	i64.load16_s	i64.load16_u	i64.load32_s	i64.load32_u	i32.store	i64.store	f32.store	f64.store	i32.store8	i32.store16	i64.store8	i64.store16	i64.store32	memory.size
4_	memory.grow	i32.const	i64.const	f32.const	f64.const	i32.eqz	i32.eq	i32.ne	i32.lt_s	i32.lt_u	i32.gt_s	i32.gt_u	i32.le_s	i32.le_u	i32.ge_s	i32.ge_u
5_	i64.eqz	i64.eq	i64.ne	i64.lt_s	i64.lt_u	i64.gt_s	i64.gt_u	i64.le_s	i64.le_u	i64.ge_s	i64.ge_u	f32.eq	f32.ne	f32.lt	f32.gt	f32.le
6_	f32.ge	f64.eq	f64.ne	f64.lt	f64.gt	f64.le	f64.ge	i32.clz	i32.ctz	i32.popcnt	i32.add	i32.sub	i32.mul	i32.div_s	i32.div_u	i32.rem_s
7_	i32.rem_u	i32.and	i32.or	i32.xor	i32.shl	i32.shr_s	i32.shr_u	i32.rotl	i32.rotr	i64.clz	i64.ctz	i64.popcnt	i64.add	i64.sub	i64.mul	i64.div_s
8_	i64.div_u	i64.rem_s	i64.rem_u	i64.and	i64.or	i64.xor	i64.shl	i64.shr_s	i64.shr_u	i64.rotl	i64.rotr	f32.abs	f32.neg	f32.ceil	f32.floor	f32.trunc
9_	f32.nearest	f32.sqrt	f32.add	f32.sub	f32.mul	f32.div	f32.min	f32.max	f32.copysign	f64.abs	f64.neg	f64.ceil	f64.floor	f64.trunc	f64.nearest	f64.sqrt
A_	f64.add	f64.sub	f64.mul	f64.div	f64.min	f64.max	f64.copysign	i32.wrap_i64	i32.trunc_f32_s	i32.trunc_f32_u	i32.trunc_f64_s	i32.trunc_f64_u	i64.extend_i32_s	i64.extend_i32_u	i64.trunc_f32_s	i64.trunc_f32_u
B_	i64.trunc_f64_s	i64.trunc_f64_u	f32.convert_i32_s	f32.convert_i32_u	f32.convert_i64_s	f32.convert_i64_u	f32.demote_f64	f64.convert_i32_s	f64.convert_i32_u	f64.convert_i64_s	f64.convert_i64_u	f64.promote_f32	i32.reinterpret_f32	i64.reinterpret_f64	f32.reinterpret_i32	f64.reinterpret_i64
C_	i32.extend8_s	i32.extend16_s	i64.extend8_s	i64.extend16_s	i64.extend32_s											
D_	ref.null	ref.is_null	ref.func	ref.as_non_null	br_on_null	ref.eq	br_on_non_null									
E_																
F_												GC Str	FC	SIMD	Threads	

SmartHR

https://pengowray.github.io/wasm-ops/

WebAssembly Opcodes

by Pengo Wray

WebAssembly is an open, industry-wide effort to bring a safe, efficient assembly language to the web. WebAssembly technology is developed collaboratively by major browser vendors including Mozilla, Google, Microsoft, and Apple. WebAssembly modules can be downloaded and executed by the majority of browsers in use today.

Toggle Dark Mode

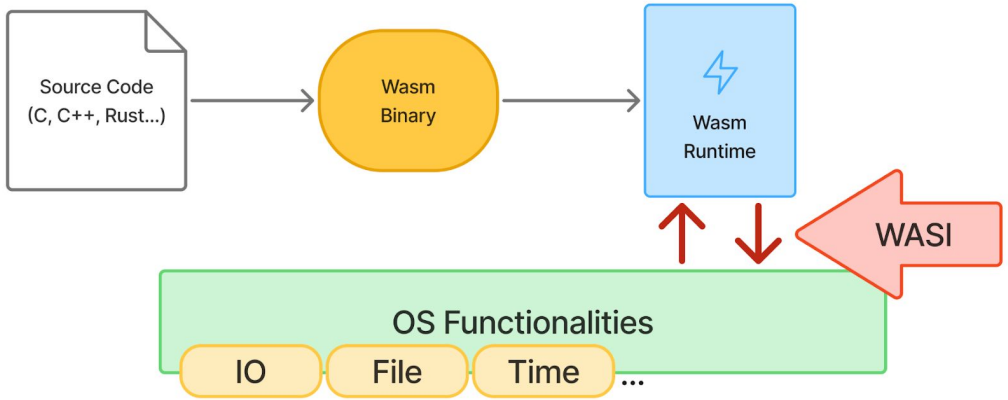
	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	unreachable	nop	block	loop	if	else	try	catch	throw	rethrow	throw_ref	end	br	br_if	br_table	return
1_	call	call_indirect	return_call	return_call_indirect	call_ref	return_call_ref			delegate	catch_all	drop	select	select t			try_table
2_	local.get	local.set	local.tee	global.get	global.set	table.get	table.set		i32.load	i64.load	f32.load	f64.load	i32.load8_s	i32.load8_u	i32.load16_s	i32.load16_u
3_	i64.load8_s	i64.load8_u	i64.load16_s	i64.load16_u	i64.load32_s	i64.load32_u	i32.store	i64.store	f32.store	f64.store	i32.store8	i32.store16	i64.store8	i64.store16	i64.store32	memory.size
4_	memory.grow	i32.const	i64.const	f32.const	f64.const	i32.eqz	i32.eq	i32.ne	i32.lt_s	i32.lt_u	i32.gt_s	i32.gt_u	i32.le_s	i32.le_u	i32.ge_s	i32.ge_u
5_	i64.eqz	i64.eq	i64.ne	i64.lt_s	i64.lt_u	i64.gt_s	i64.gt_u	i64.le_s	i64.le_u	i64.ge_s	i64.ge_u	f32.eq	f32.ne	f32.lt	f32.gt	f32.le
6_	f32.ge	f64.eq	f64.ne	f64.lt	f64.gt	f64.le	f64.ge	i32.clz	i32.ctz	i32.popcnt	i32.add	i32.sub	i32.mul	i32.div_s	i32.div_u	i32.rem_s
7_	i32.rem_u	i32.and	i32.or	i32.xor	i32.shl	i32.shr_s	i32.shr_u	i32.rotl	i32.rotr	i64.clz	i64.ctz	i64.popcnt	i64.add	i64.sub	i64.mul	i64.div_s
8_	i64.div_u	i64.rem_s	i64.rem_u	i64.and	i64.or	i64.xor	i64.shl	i64.shr_s	i64.shr_u	i64.rotl	i64.rotr	f32.abs	f32.neg	f32.ceil	f32.floor	f32.trunc
9_	f32.nearest	f32.sqrt	f32.add	f32.sub	f32.mul	f32.div	f32.min	f32.max	f32.copysign	f64.abs	f64.neg	f64.ceil	f64.floor	f64.trunc	f64.nearest	f64.sqrt
A_	f64.add	f64.sub	f64.mul	f64.div	f64.min	f64.max	f64.copysign	i32.wrap_i64	i32.trunc_f32_s	i32.trunc_f32_u	i32.trunc_f64_s	i32.trunc_f64_u	i64.extend_i32_s	i64.extend_i32_u	i64.trunc_f32_s	i64.trunc_f32_u
B_	i64.trunc_f64_s	i64.trunc_f64_u	f32.convert_i32_s	f32.convert_i32_u	f32.convert_i64_s	f32.convert_i64_u	f32.demote_f64	f64.convert_i32_s	f64.convert_i32_u	f64.convert_i64_s	f64.convert_i64_u	f64.promote_f32	i32.reinterpret_f32	i64.reinterpret_f64	f32.reinterpret_i32	f64.reinterpret_i64
C_	i32.extend8_s	i32.extend16_s	i64.extend8_s	i64.extend16_s	i64.extend32_s											
D_	ref.null	ref.is_null	ref.func	ref.as_non_null	br_on_null	ref.eq	br_on_non_null									
E_																
F_												GC Str	FC	SIMD	Threads	

Wasm実装のポイント (3): WASI (System Interface)

- **WASI (WebAssembly System Interface) とは？**
  - Wasm VMで「Hello World」のようなOS機能（標準出力など）を使うための**抽象化されたインターフェース**
- Wasmバイナリー内でWASI仕様を満たす関数を呼び出すことで、システム機能を利用できる
- **例: 標準出力**に書き出す命令の実装が必要

WASI picture

WASI defines APIs to interact with the OS



https://udzura.jp/slides/2025/rubykaigi/#23

ということで
実際に作ってみた

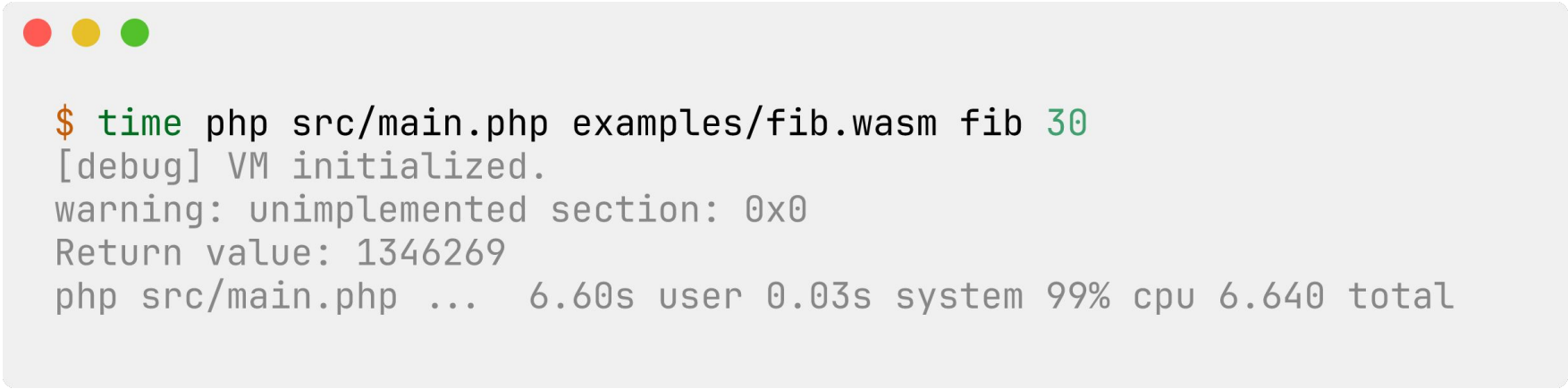
リポジトリは以下です

- GitHub - udzura/phpconvm

PHPによるWasm VMの実装 (デモ)

- **実装:** Gorilla本の設計をベースに、過去にRubyで実装した経験を活かし、**PHPに移植**した
- **デモ内容:** 自作PHP VM/Ruby製VMでWasmバイナリー (Hello World, fib) を実行

```
$ php src/main.php examples/helloworld.wasm
[debug] VM initialized.
warning: unimplemented section: 0x0
Hello, World!
```


$ time php src/main.php examples/fib.wasm fib 30
[debug] VM initialized.
warning: unimplemented section: 0x0
Return value: 1346269
php src/main.php ... 6.60s user 0.03s system 99% cpu 6.640 total

```
## JIT enabled:
$ time php src/main.php examples/fib.wasm fib 30
[debug] VM initialized.
warning: unimplemented section: 0x0
Return value: 1346269
php src/main.php ...  4.29s user 0.04s system 99% cpu 4.334 total
```

ちなみに参考元のRuby実装の速度

```
$ time bundle exec ruby exe/wardite ../../examples/fib.wasm fib 30
warning: unimplemented section: 0x00
return value: 1346269
bundle exec ruby ...  7.90s user 0.07s system 99% cpu 8.019 total
```

```
$ time bundle exec ruby --yjit exe/wardite ../../examples/fib.wasm fib 30
warning: unimplemented section: 0x00
return value: 1346269
bundle exec ruby --yjit ...  3.44s user 0.06s system 98% cpu 3.549 total
```

結果を比較してみた

- fib(30)を計算した場合の実行時間比較

実装	実行時間(通常)	実行時間(JIT有効)
PHP実装	6.64秒	4.33秒
Ruby実装	8.02秒	3.55秒

PHPによるWasm VMの実装の感想

- **学び:**
  - PHP、豊富な文字列操作関数があるので結構サクサク作れた...
  - オンメモリバッファの作り方知らなかった。

    `fopen('php://memory', 'r+b')`
- あと正直な話
  - AIありがとう...
  - 今日の画像もAI使ったよ

まとめと今後

- **本日の発表:**
  - PHPによるWasm VMの最小限の実装を紹介
- **今後の展望:**
  - 命令をさらに追加することで、本格的に動作するVMになるはず
  - 興味がある方は、ぜひプルリクエストをお待ちしています

Respects...

https://fortee.jp/phperkaigi-2024/proposal/ac59d0dd-795a-47cb-ba59-c0b1772d00cc

ご清聴ありがとうございました！！

SmartHR