PHPで作るPHP
～セルフホストできる言語処理系を作ろう～

nsfisis (いまむら)

PHPerKaigi 2025

いまむら

nsfisis

@デジタルサーカス株式会社

セルフホストできる
言語処理系を作ろう

セルフホストできる
言語処理系を作ろう

言語処理系

ソースコードを実行したり
別の形式に変換したりするもの

php コマンド

PHP の言語処理系
PHP のソースコードを
実行する

Node.js

JavaScript 等の言語処理系
JavaScript 等のソースコードを
実行する

GCC

C 言語等の言語処理系
C 言語等のソースコードを
実行ファイルへ変換する

tsc

TypeScript の言語処理系
TypeScript のソースコードを
JavaScript のソースコードへ
変換する

セルフホストできる
言語処理系を作ろう

セルフホストできる
言語処理系を作ろう

セルフホスト

処理系自身のソースコードを
自分自身が処理できる

tsc

TypeScript で書かれた
TypeScript の処理系
tsc のソースコードを
tsc が JavaScript にできる

GCC

C 言語 等で書かれた
C 言語 等の処理系
GCC のソースコードを
GCC が実行ファイルにできる

Node.js

C++ 等で書かれた
JavaScript 等の処理系
Node.js のソースコードを
Node.js は実行できない

php コマンド

C 言語等で書かれた
PHP の処理系
php コマンドのソースコードを
php コマンドは実行できない

言語
と
処理系

言語

TypeScript	JavaScript	C
tsc	Node.js	GCC

処理系

php
PHP 処理系

phphp
私が作成した/今回の
PHP 処理系

セルフホストできる
言語処理系を作ろう

PHP で書かれた
PHP の処理系

自身のソースコードを
実行できる

言語処理系の中身

1. 字句解析
2. 構文解析
3. なんやかんや
4. 実行

1. 字句解析
2. 構文解析
3. なんやかんや
4. 実行

ソースコード
↓
トークン列
↓
抽象構文木 (AST)
↓
なんらか
↓
実行結果

1. 字句解析
2. 構文解析
3. ~~なんやかんや~~
4. 実行

ソースコード
↓
トークン列
↓
抽象構文木 (AST)
↓
~~なんらか~~
↓
実行結果

```
for ($i = 1; $i <= 100; $i++) {
    if ($i % 15 === 0) {
        echo “FizzBuzz\n”;
    } elseif ($i % 3 === 0) {
        echo “Fizz\n”;
    } elseif ($i % 5 === 0) {
        echo “Buzz\n”;
    } else {
        echo $i, “\n”;
    }
}
```

```
for ($i = 1; $i <= 100; $i++) {
    if ($i % 15 === 0) {
        echo “FizzBuzz\n”;
    } elseif ($i % 3 === 0) {
        echo “Fizz\n”;
    } elseif ($i % 5 === 0) {
        echo “Buzz\n”;
    } else {
        echo $i, “\n”;
    }
}
```

字句解析

意味の単位（トークン）で区切る

```
for ($i = 1; $i <= 100; $i++) {
    if ($i % 15 === 0) {
        echo “FizzBuzz\n”;
    } elseif ($i % 3 === 0) {
        echo “Fizz\n”;
    } elseif ($i % 5 === 0) {
        echo “Buzz\n”;
    } else {
        echo $i, “\n”;
    }
}
```

字句解析

意味の単位（トークン）で区切る

- for
- (
- $i
- =
- 1
- ;
- $i
- <=
- 100
- ;
- …

```
for ($i = 1; $i <= 100; $i++) {
    if ($i % 15 === 0) {
        echo “FizzBuzz\n”;
    } elseif ($i % 3 === 0) {
        echo “Fizz\n”;
    } elseif ($i % 5 === 0) {
        echo “Buzz\n”;
    } else {
        echo $i, “\n”;
    }
}
```

構文解析

意味上のまとまりを認識する

```
for ($i = 1; $i <= 100; $i++) {
    if ($i % 15 === 0) {
        echo “FizzBuzz\n”;
    } elseif ($i % 3 === 0) {
        echo “Fizz\n”;
    } elseif ($i % 5 === 0) {
        echo “Buzz\n”;
    } else {
        echo $i, “\n”;
    }
}
```

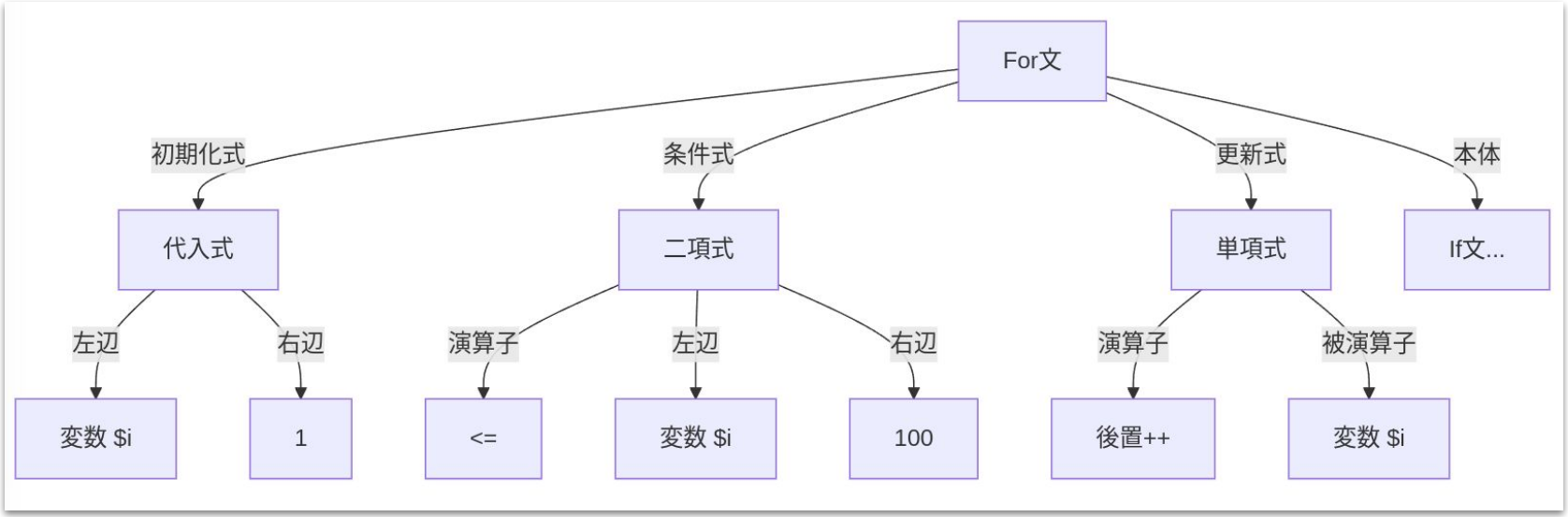
構文解析

意味上のまとまりを認識する
→抽象構文木（AST）

```
for ($i = 1; $i <= 100; $i++) {
    if ($i % 15 === 0) {
        echo “FizzBuzz\n”;
    } elseif ($i % 3 === 0) {
        echo “Fizz\n”;
}
```

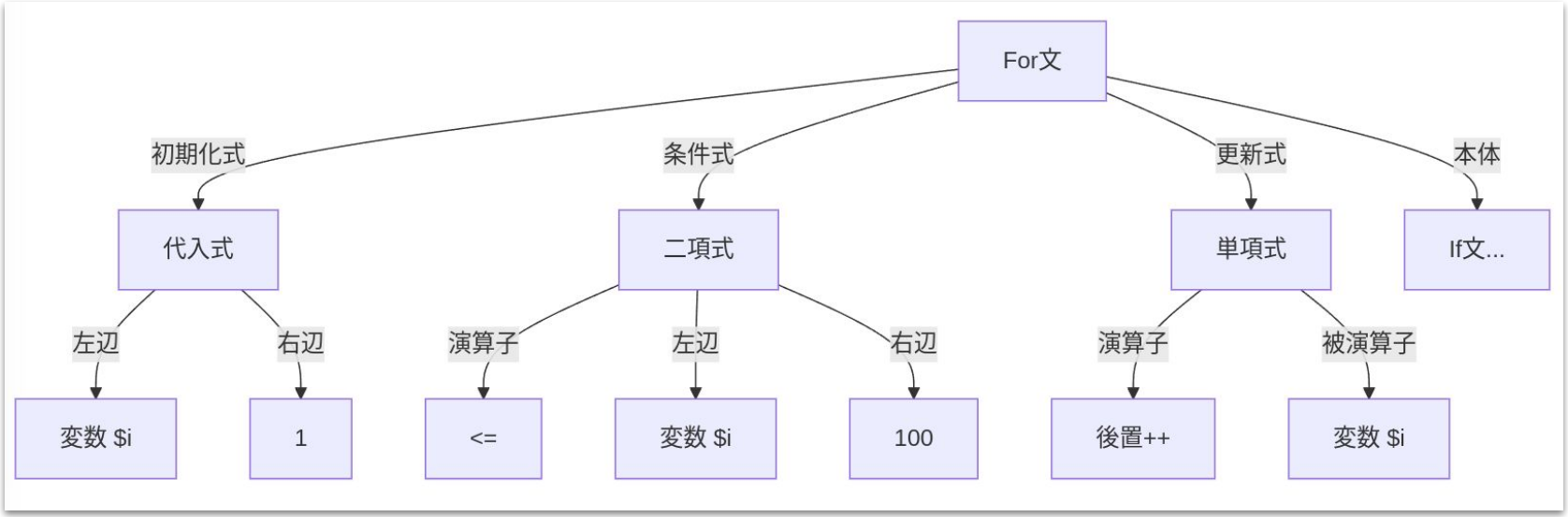
構文解析

意味上のまとまりを認識する
→抽象構文木（AST）



実行

```
for ($i = 1; $i <= 100; $i++) {
    if ($i % 15 === 0) {
        echo “FizzBuzz\n”;
    } elseif ($i % 3 === 0) {
        echo “Fizz\n”;
}
```



1. 字句解析
2. 構文解析
3. 実行

ソースコード
↓
トークン列
↓
抽象構文木 (AST)
↓
実行結果

セルフホスト特有の留意点

セルフホスト特有の留意点

言語機能を使いすぎない
あとから実装する必要がある

字句解析

```
function tokenize(string $source): array
{
    $tokens = [];
    $position = 0;

    while ($position < strlen($source)) {
        ...
    }

    return $tokens;
}
```

```
$c = $source[$position];
++$position;
...
} elseif ($c === '=') {
    if ($source[$position] === '=') {
        ++$position;
        if ($source[$position] === '=') {
            ++$position;
            $tokens[] = '===';
        } else {
            $tokens[] = '==';
        }
    } else {
        $tokens[] = '=';
    }
}
...
```

```
...
} elseif (is_digit($c)) {
    $value = $c;
    while (
        $position < strlen($source) &&
        is_digit($source[$position])
    ) {
        $value .= $source[$position];
        ++$position;
    }
    $tokens[] = ['integer', intval($value)];
}
...
```

構文解析

再帰下降

```
function parse_for_statement(array $tokens, int $position): array
{
    expect_token($tokens, $position, 'for');
    ++$position; // for
    expect_token($tokens, $position, '(');
    ++$position; // (
    [$init, $position] = parse_expression($tokens, $position);
    ++$position; // ;
    [$cond, $position] = parse_expression($tokens, $position);
    ++$position; // ;
    [$update, $position] = parse_expression($tokens, $position);
    ++$position; // )
    [$body, $position] = parse_statement($tokens, $position);
    return [['for', $init, $cond, $update, $body], $position];
}
```

```
function parse_statement(array $tokens, int $position): array
{
    $first = $tokens[$position];
    if ($first[0] === 'for') {
        return parse_for_statement($tokens, $position);
    } elseif ($first[0] === 'if') {
        return parse_if_statement($tokens, $position);
    } elseif ($first[0] === 'return') {
        return parse_return_statement($tokens, $position);
    } elseif ($first[0] === '{') {
        return parse_statements($tokens, $position);
    }
    return parse_expression_statement($tokens, $position);
}
```

実行

```
function evaluate_if_statement(
    array $env,
    array $condition,
    array $then,
    array $else,
): array {
    [$env, $condition_result] = evaluate($env, $condition);
    if ($condition_result) {
        return evaluate($env, $then);
    } else {
        return evaluate($env, $else);
    }
}
```

```
function evaluate_function_declaration(
    array $env,
    string $name,
    array $parameters,
    array $body,
): array {
    $env[‘functions’][$name] = [$parameters, $body];
    return [$env, null];
}
```

```
function evaluate_binary_expr(
    array $env,
    string $operator,
    array $left,
    array $right,
): array {
    [$env, $left_result] = evaluate($env, $left);
    [$env, $right_result] = evaluate($env, $right);
    if ($operator === '%') {
        return [$env, $left_result % $right_result];
    } elseif ($operator === '<') {
        return [$env, $left_result < $right_result];
    }
    ...
}
```

1. 字句解析
2. 構文解析
3. 実行

ソースコード
↓
トークン列
↓
抽象構文木 (AST)
↓
実行結果

セルフホストへの道（の一部）

- function
- while
- for
- if
- return
- break
- continue
- echo

- variable
- constant
- binary expression
- unary expression
- string literal
- number literal
- function call
- type annotation

```
function parse_function_declaration(...): array
{
    $position++; // function
    $name = $tokens[$position][1];
    $position++; // <name>
    $position++; // (
    [$parameters, $position] = parse_parameters($tokens, $position);
    $position++; // )
    if ($tokens[$position][0] === ':') {
        $position++; // :
        $position++; // <type>
    }
    ...
}
```

```
function evaluate_call_expr(
    array $env,
    array $func,
    array $args,
): array {
    $name = ...;
    if (array_key_exists($name, $env['funcs'])) {
        ...
    } else {
        $ret = call_user_func_array($name, $arg);
    }
}
```

まとめ

セルフホストできる処理系を作るのに必要なもの

根気