PHP8から追加されたJITについて学ぼう!

たけてぃ(@takeokunn)

はじめに

2023年現在、PHPは世界中の企業で使われており、最も成功しているプログラミング言語の1つと言っても過言ではないくらい開発者から愛されています。
WikipediaやWordPressもPHP製のプロダクトなので、開発者でなくてもPHPの恩恵を受けている人は多いでしょう。

私も5年以上PHPを用いて開発していますが、日進月歩で進化し続けるPHPにワクワクさせられています。

PHP 8.0が2020年11月にリリースされ、様々な便利な新機能が追加されました。 特に Match式 は複雑な条件分岐を「式」で書くことができ、より柔軟な記述が可能になります。
高く評価されている新機能の1つに JIT (ジャストインタイム) コンパイラ があります。 JITを正しく使うと、条件によってはアプリケーションの1.5倍以上高速化できるということを公式のリリースページに書いてあります。
そもそもJITとは何か、PHP上でどのようにJITが動作しているのかを知れば、アプリケーションの記述を変えずに高速化できるでしょう。

JITについての概要

JIT自体の歴史は長く、様々な言語の処理系で実装されています。 JavaScriptの処理系であるV8 EngineもJITを導入することによって大幅な高速化を実現しました。

Ruby 3.1ではYJITが導入されました。
YJITはShopify社製のJIT Compilerで、同社はRailsアプリケーションが 20%~40% 程度高速化できたという報告をしています。
私の普段使っているEmacsでもJIT Compilerがサポートされて、体感速度として感じられるくらい動作がサクサクになったという実感があります。

PHP、Ruby、JavaScriptのような動的型付き言語は、CやRustのような静的型付き言語よりも実行速度が遅いです。
遅い理由は様ありますが、実行時に字句解析、構文解析、コンパイルをして中間コードを吐き出し、VM上で実行をしているのが原因です。
変数や関数の型情報の多くは実行時に決まり、実行時にVM上で最適化が走るので、どうしても遅くなってしまいます。

高速に実行するアプローチの1つにJIT Compileがあります。
JIT Compileは実行時にマシンコードを吐き出し、キャッシュ上に乗せ、再度実行する時にキャッシュ上のマシンコードを実行するというアプローチです。
マシンコードにすることによって、中間コードにわざわざ変換をしてVM上で変換をする必要がないので高速に動かすことができます。
ちなみに、JITの対比として、C言語のような事前にコンパイルする通常の手法はAOT(Ahead-Of-Time)コンパイルと呼びます。
JIT Compilerの実装は様々ありますが、PHPではDynASMを使っています。 DynASMはLuaでも使われており、既に実績を残しています。

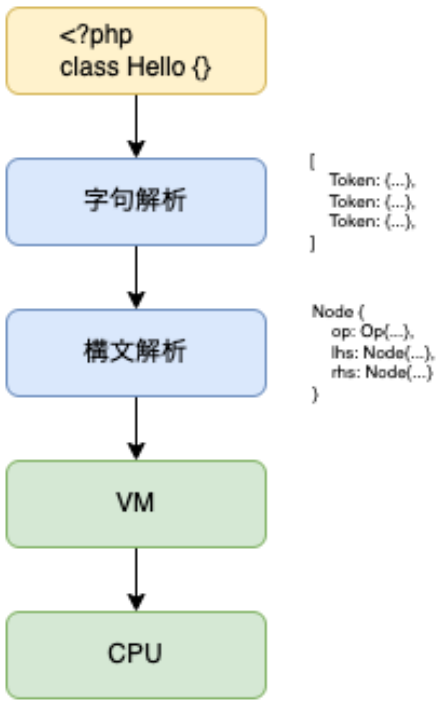
PHPのJITの基本的な仕組み

2023年1月現在、JITはOPCacheのサブセットとして提供されています。
OPCacheとJITの立ち位置について解説していきます。

1. PHPのみの場合

PHPは通常、図のような処理が行われます。
字句解析で文字列をトークン化、構文解析で opcode を含む構文木(AST)を作り、VM上で実行をする、というフローを辿ります。

opcode は中間コードのことであり、VMで解釈できるような命令セットのことです。



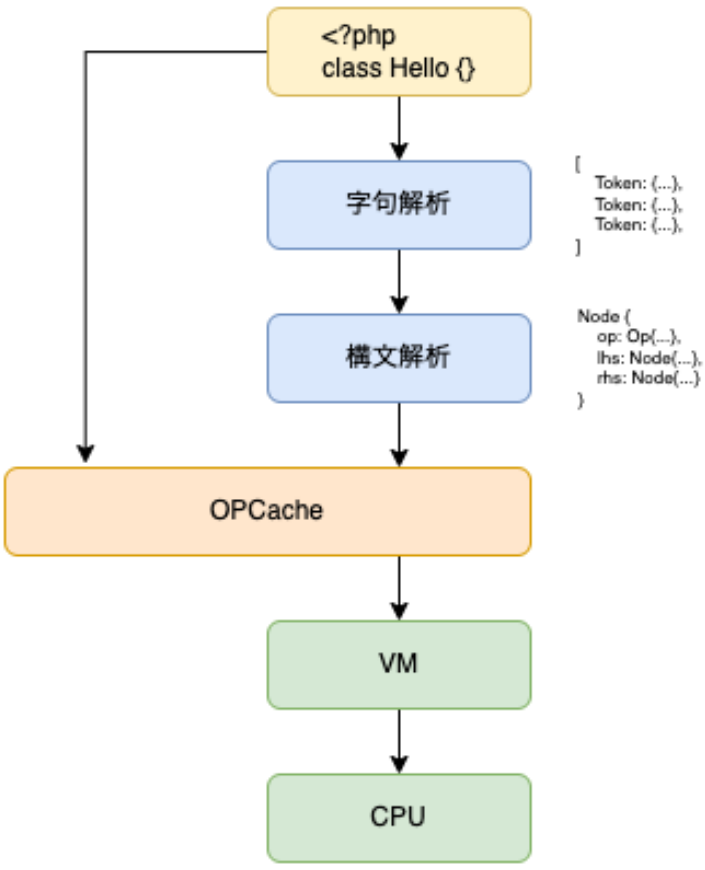
VMの仕事は中間コードを読み込み、最適化した上で処理を実行をすることです。

PHPのVMは ZendEngine と呼ばれており、ZendEngineがCPUアーキテクチャ間の際を吸収しています。
私達が普段 M1 Mac でも Intel x86 でも同じコードから同じ実行結果を得られるのは ZendEngine があるからこそです。

通常PHPは実行時にこれら一連の処理を毎回行っています。
PHPサーバーのプロセスを再起動せずにコードが反映されるというメリットがあります。

しかし、毎回一連の処理をするのは非効率です。
適切にcacheを効かせて実行時の処理を減らす機構こそがOPCacheです。

2. OPCacheの場合



OPCacheを有効にすると、図のような処理が行われます。

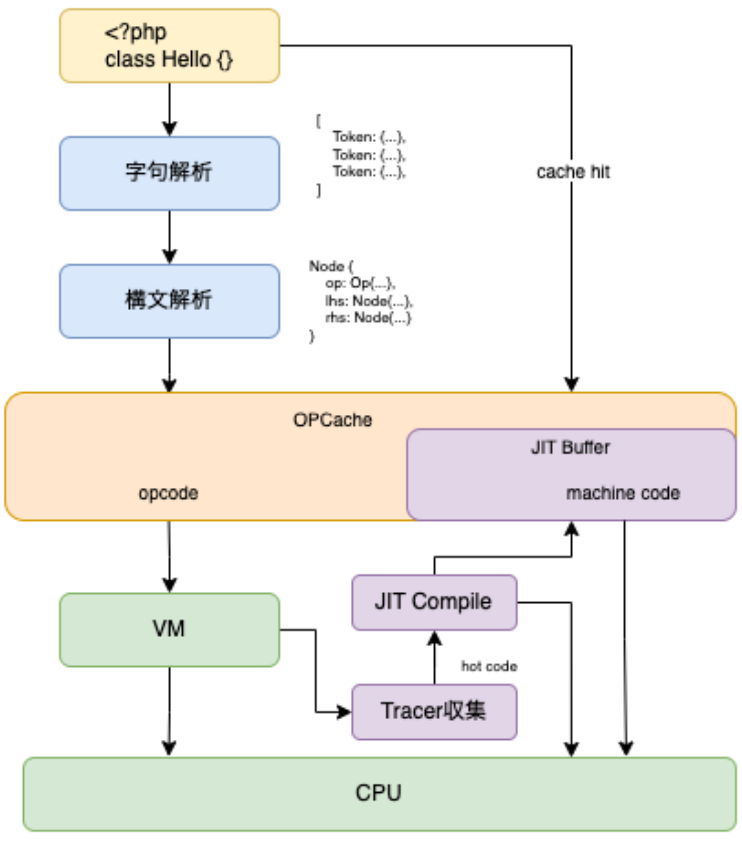
字句解析や構文解析をした結果をOPCacheでキャッシュをすることによって、2回目以降(キャッシュが存在する場合)処理をせずに済みます。

OPCacheでキャッシュされたものは共有メモリに保存されます。 共有メモリに保存されることによって、 apache や php-fpm のような複数プロセスが動かすような環境にも対応しています。

OPCacheはPHP 5.5で標準機能として取り込まれたもので、それ以前から pecl 経由でインストールできました。 歴史と実績のある機能で、ほとんどのPHPを使っている企業ではOPCacheを有効にして運用しているはずです。

ただOPCacheを有効にしたところで、結局は ZendEngine 上で逐次実行をしているのでどうしても遅くなってしまいます。 ZendEngine を経由せずに直接マシンコード生成し、CPU上で実行をする手法がJITというものです。

3. JITの場合



JITを有効にすると、図のような処理が行われます。
Tracerで実行時前後の型情報を集め、JITコンパイルをしてマシンコードを生成、
OPCache内のJIT Bufferにマシンコードをキャッシュをします。
2回目以降(キャッシュが存在する場合)は、マシンコードをそのままCPU上で実行することによってZend Engineすら動かさずにPHPコードを実行できます。

JITを有効にしたら必ずしも高速化するかと言われたらそうではない場合もあります。
最初にJITコンパイルが実行されるタイミングはどうしてもコストがかかってしまいます。
偶にしか実行されないPHPコードも含め全部JITコンパイルをするのは無駄が多く発生してしまいます。
ベンチマークで結果が出ていない場合は大体これが原因だと推測できます。

何度も呼ばれている関数のみを計画的にJITコンパイルする方が筋の良い方法と言えるでしょう。

PHPのJITには tracing という機能があります。
関数がどれくらい呼ばれたか、どのような引数の型で呼ばれたのか等、実行時の情報を収集する機能があります。 関数呼び出し回数の閾値が超えたタイミングでJITコンパイルをします。

インストール方法

JITを利用するにあたって、 php.ini のOPCache周りの設定を有効にする必要があります。

[opcache]
opcache.enable=1
opcache.enable_cli=1

JITを以下のように有効にします。

opcache.jit=tracing ; function
opcache.jit_buffer_size = 128M

opcache.jit のオプションは大きく分けて tracing と function の2つがあります。 tracing や function という文字列はエイリアスであって、4桁の整数値 CRTO を直接指定するという高度な使い方もあります。

CRTOはそれぞれ、以下の略称です。

・C (特定のCPU向けの最適化フラグ)
・R (レジスタの割り付け)
・T (JITを行うトリガ)
・O (最適化レベル)

opcache.jit_buffer_size はコンパイル済みのJITコードを保存する共有メモリの合計サイズです。 少なめに指定してしまうとJITを有効に活用できないので、OPCacheで割り当てた共有メモリの 50% 程度指定すると良いでしょう。

opcache.jit を tracing で有効にした場合、 opcache.jit_hot_loop や opcache.jit_hot_func など細かく指定できます。 基本的にはデフォルトの設定で問題ないですが、チューニングしたい場合はこの辺りの設定を弄ると良いでしょう。

JITに適したコード

以下のような A と B のコードはどちらがJITに適したコードでしょうか？

```
// A
function A($a, $b)
{
   return $a + $b;
}

// B
declare(strict_types=1);

function B(float $a, float $b): float
{
   return $a + $b;
}
```

答えは **B** です。

JIT Compilerは実行時前後に型情報を集め、最適化されたマシンコードを吐き出します。

declare(strict_types=1); は厳格な型検査モードの指定構文です。 暗黙な型変換を抑えることができます。
PHP 7以降型をより厳格に書けるようになり、PHPStanやPsalmのような静的型検査ができるツールも普及してきています。
それらのツールを使い、厳格に記述すればするほどJIT Compilerの恩恵を受けることができるでしょう。

終わりに

冒頭にも書きましたが、PHPという言語は常に正しく、安全に、高速に動かすべく進化をしています。 10年前では考えられないほど堅牢に記述でき、実行時のバグが圧倒的に減ったはずです。

PHPのバージョンを常に上げ続けるのは苦労します。 今まではWarningで済んでたものがErrorになったり、依存するcomposer packageが対応していなかったり、大きいプロダクトであればあるほど大変さが増します。 しかし、バージョンを上げることによって多くのメリットを享受できるので、普段から上げやすい体勢を整えることが求められています。

貴方が関わってるPHPプロジェクトも8.0に上げてJITを有効にし、高速化してみては如何でしょうか。

参考記事

PHP JIT in Depth
https://php.watch/articles/jit-in-depth

JIT のコードを読んでみた
https://www.slideshare.net/y-uti/jit-70023246

PHP8でのJIT導入の背景について調べてみた
https://tech.griphone.co.jp/2021/12/23/php8jit/

JITあれこれ
https://keens.github.io/blog/2018/12/01/jitarekore/