Phpactor から学ぶ Language Server Protocol の仕組み

たけてぃ @takeokunn

1. はじめに

ソフトウェアの世界は言語、フレームワーク、エコシステム、ハードウェア等どれをとっても日進月歩で急速に進歩しています。

筆者の所属しているテキストエディタコミュニティでも直近 10 年大きな進歩発展がいくつかありました。 その代表例として Language Server Protocol（以下 LSP）と Tree-Sitter の普及があげられます。 これにより、新興のエディタでも歴史のあるエディタでも統一された開発環境を提供することが簡単になりました。

本記事ではそんな LSP の歴史と基本機能、PHP の代表的な Language Server である Phpactor について紹介します。

2. LSP について

2.1. 概要

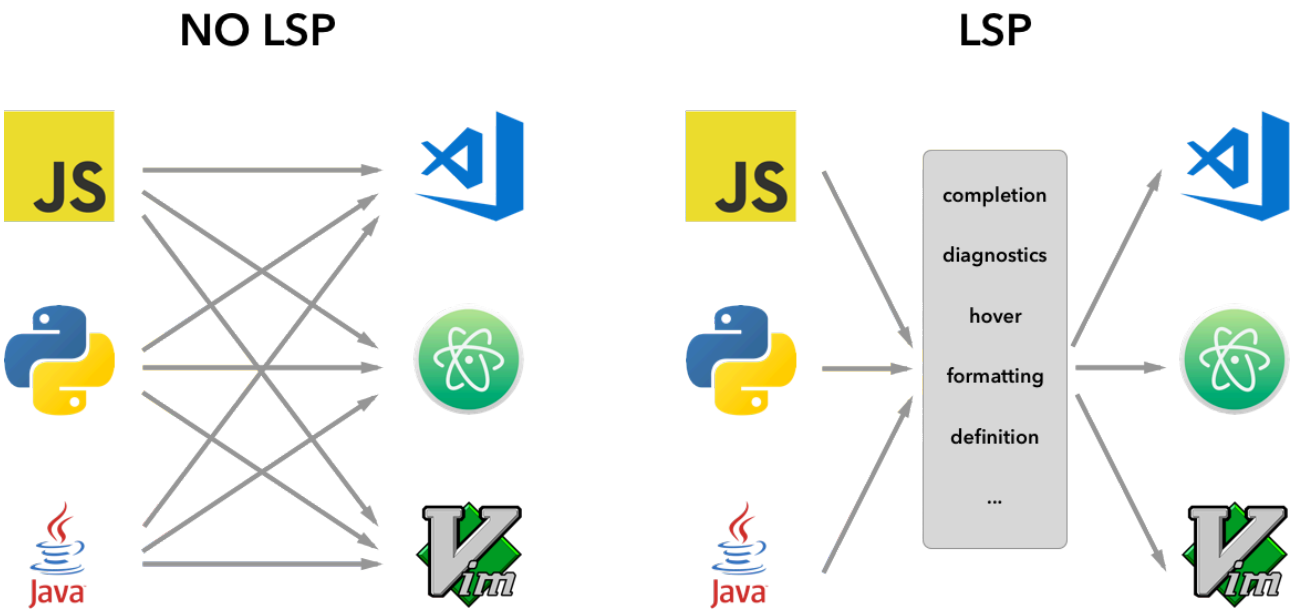
Language Server Protocol （LSP）は、2016 年に Microsoft が発表した JSON-RPC ベースのプロトコルです。[1] 簡単な表現をすると、モダンテキストエディタの機能を提供している定義ジャンプやリネーム、構文エラーを表示する為の標準的な Protocol です。

LSP は LSP Client と Language Server（言語サーバ）の 2 つで構成されています。

LSP Client は Neovim/Emacs/VSCode などテキストエディタのことです。 Language Server は各言語ごとに実装されており、Protocol で定義されたインタフェースに乗っ取って好き好きに実装されています。

ここで重要なのは Protocol で定義されたインタフェースがあるということです。

新しい Language Server を作った時でも既存の LSP Client 実装のまま動かすことが可能になります。 逆もまた然りで、新しい LSP Client を実装しても既存の Language Server を動かすことが可能です。



LSP はさっき定義した関数の補完が効いてほしい、編集時にエラーを出してほしい、といったように LSP はリアルタイムに実行されることを想定したものです。 リアルタイム性を重視しているので実装次第ではパフォーマンスに問題が発生してしまします。

同様の Protocol に Language Server Index Format（LSIF）[2] があります。 事前にインデックスを生成して LSIF ファイルを生成することによって、LSP と同じような機能を実現できます。 これにより、Web 上で定義ジャンプをしたりシンボルの検索も実現可能になります。

2.2. LSP 前史

2.2.1. ctags

普段コードを編集して欠かせない機能のひとつに定義ジャンプがあります。 2025 年現代の定義ジャンプはほぼ LSP で実現されていますが、LSP が普及する前は ctags[3] が使われていました。 ctags はプロジェクト内の関数、変数、クラス、マクロなどのシンボル情報をインデックスファイルに出力プログラムです。 インデックスファイルをテキストエディタ側の実装で読み込むことによって、タグジャンプや補完候補に出すことを実現していました。 universal-ctags[4] は 2025 年 1 月現在でも開発が続いているようです。

[1] Language Server, https://microsoft.github.io/language-server-protocol/overviews/lsp/overview/
[2] Language Server Index Format, https://microsoft.github.io/language-server-protocol/overviews/lsif/overview/

2.2.2. OmniSharp

LSP の前身に C# の OmniSharp[5] があります。

C# を開発するなら Visual Studio が一般的でしたが、OmniSharp は Vim や Emacs でも C# を快適に開発できるようにすることを目的としたものでした。 当時の日本人テキストエディタコミュニティの方々が熱心に OmniSharp にコントリビュートした結果、LSP が生まれたという過去があるようです。

LSP の GitHub Wiki に Protocol-History というページ [6] があり、OmniSharp から着想を得たという記述もあります。

This concept was picked up by OmniSharp.
OmniSharp provides auto complete and other rich editing features for C#.
Initially OmniSharp used the http protocol with a JSON payload.
OmniSharp has been integrated into several editors. One of them was VS Code.

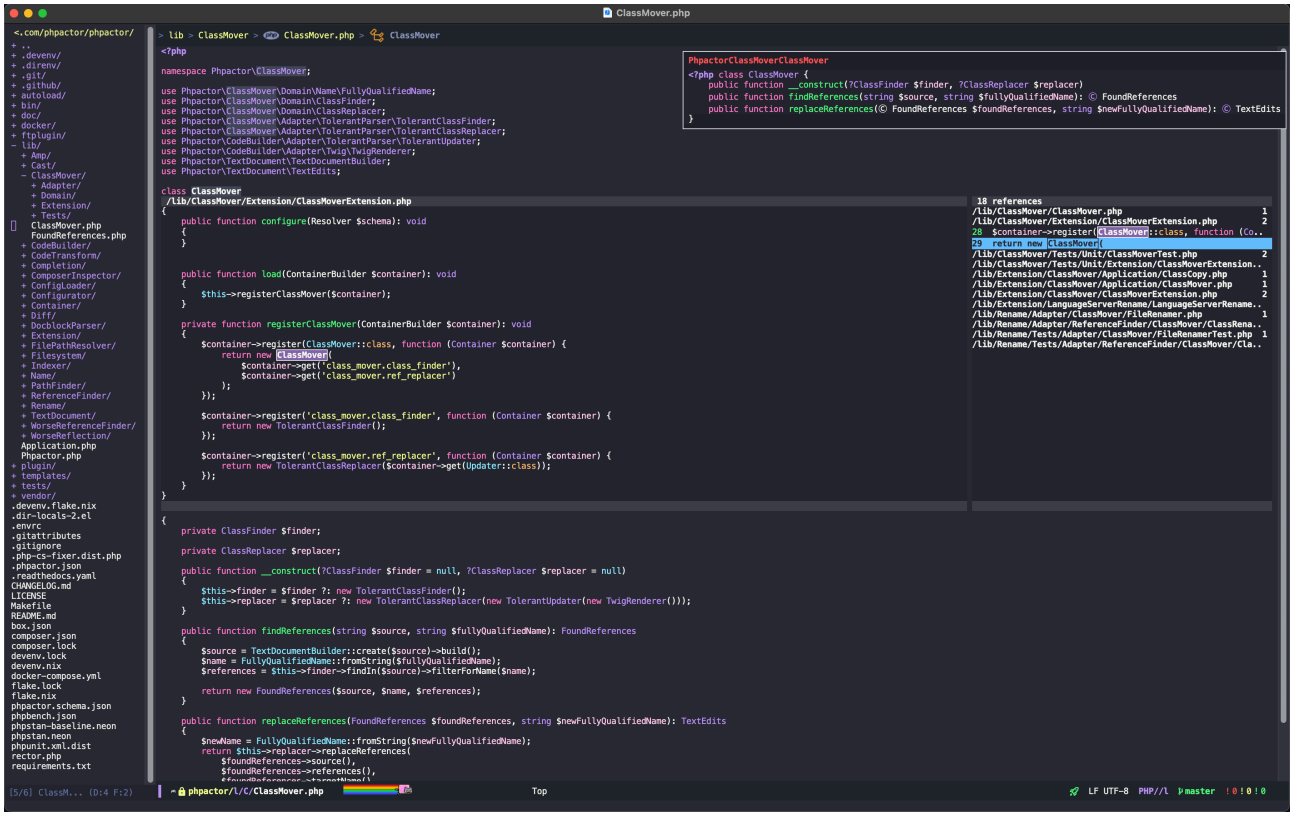
2.3. 基本機能

LSP が提供しているメジャーな機能は以下です。 2025 年現在、普通の IDE なら必ず提供している機能を網羅しています。

名前	機能	機能説明
Diagnostics	検査	コードの解析結果としてエラーや警告をエディタ上に表示する
Completion	補完	入力中のコードから続く候補を補完候補に列挙する
Goto definition	定義へ移動	変数や関数等の定義箇所にジャンプする
Find references	参照を表示/移動	変数の参照箇所をリストアップする
Rename	名前変更	変数の名前を変更する

上記のメジャー機能以外にも、事前に登録したコマンドを実行する機能（Execute Command）や、API に関する情報を表示する機能（Hover）などもあります。

Protocol 自体の解説は「Language Server Protocol の仕様 および実装方法 [7]」が非常にわかりやすく、こちらを参照することをお勧めします。



[3]Universal Ctags, https://ctags.io/
[4]universal-ctags/ctags - GitHub, https://github.com/universal-ctags/ctags
[5]OmniSharp, https://www.omnisharp.net/
[6]Protocol-History - GitHub, https://github.com/microsoft/language-server-protocol/wiki/Protocol-History
[7]Language Server Protocol の仕様 及び実装方法 - Zenn Book, https://zenn.dev/mtshiba/books/language_server_protocol

3. PHP を取り巻く LSP 環境

PHP のメジャーな Language Server は intelephense[8] と Phpactor[9] の 2 種類あります。 他にも Serenata[10] や php-language-server[11] がありますが、筆者の印象では intelephense を使っている人が圧倒的に多いです。

intelephense は OSS ではなく商用ソフトウェアで無料版と有料版があります。 ソースコードは公開されておらず、npm 経由で実行ファイルを入れて利用します。 無料版でも十分使うことは可能ですが、LISENCE を購入することでは Rename や Go to type definition など便利な機能が使えます。

一方 Phpactor や Serenata や php-language-server は OSS の PHP プロジェクトです。 余談ですが、Emacs 標準の LSP Client である eglot では intelephense を採用できなかったので、phpactor と php-language-server をサポートしています。

4. Phpactor について

4.1. 概要

Phpactor は PHP プロジェクトなので Composer で簡単に導入できます。

LSP サポート情報は公式サイトで確認できますが、基本的な機能はサポートされています。[12] 設定は json で記述でき、 `phpactor config:dump` で出力できます。

```JSON
{
    "language_server_phpstan.enabled": false,
    "completion_worse.experimantal": true,
    "language_server_worse_reflection.workspace_index.update_interval": 5000,
    "language_server_php_cs_fixer.enabled": false,
    "php_code_sniffer.enabled": false,
    "prophecy.enabled": false
}
```

Emacs や Vim や VS Code などだいたいのテキストエディタの LSP Client はすでにサポートしているので、手元のテキストエディタで簡単に試すことができます。[13] Phpactor はあくまで Language Server ですので、実際の編集体験や細かい使い勝手は LSP Client 側の対応状況依存になります。

Phpactor で大規模プロジェクトを開くと非常に重いといった問題があります。 公式サイトの Performance 改善方法が記述されているページ通りに設定するとだいぶ緩和されるので対応することをお勧めします。[14]

```Shell
# Large Files
$ phpactor config:set language_server.diagnostics_on_update false

# Indexing
$ phpactor config:set indexer.exclude_patterns '["/vendor/**/Tests/**/*","/vendor/**/tests/**/*","/var/cache/**/*","/vendor/composer/**/*"]'
```

4.2. コマンドライン利用

Phpactor は Language Server としても使えますが、CLI ツールとしても非常に強力です。 CLI ツールとして使えば証跡を残しやすいので、チーム内でコミュニケーションを取る時に非常に便利です。 いくつかの便利なコマンドを紹介します。

4.2.1. Indexer

プロジェクト内の php ファイルをスキャンし、クラスと関数に関するメタ情報を記録するコマンドです。 主に定義ジャンプ時に利用します。 テキストエディタ起動時に実行されますが、事前に実行しておくことによってキャッシュを温めておくことが可能です。

```Shell
# 通常の実行
$ phpactor index:build

# watch
$ phpactor index:build --watch
```

4.2.2. Refactoring

筆者はテキストエディタで編集するのでほぼ使ったことがないですが、リファクタリングを自動でするコマンドも提供しています。

たとえば、 `class:transform` を使えば以下のように自動で修正してくれます。

---

[8] intelephense, https://intelephense.com/
[9] Phpactor, https://phpactor.readthedocs.io/en/master/index.html
[10] Serenata, https://serenata.gitlab.io/
[11] felixfbecker/php-language-server, https://github.com/felixfbecker/php-language-server
[12] LSP Support - Phpactor, https://phpactor.readthedocs.io/en/master/lsp/support.html
[13] Language Server - Phpactor, https://phpactor.readthedocs.io/en/master/usage/language-server.html
[14] Performance - Phpactor, https://phpactor.readthedocs.io/en/master/tips/performance.html

```shell
$ phpactor class:transform path/to/Class.php --transform=add_missing_assignments
```

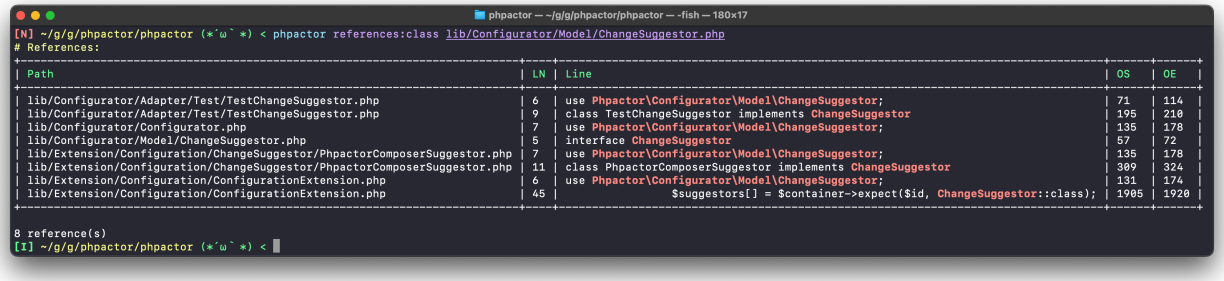
```php
/**
 * before
 */
class AcmeBlogTest extends TestCase
{
    public function setUp()
    {
        $this->blog = new Blog();
    }
}
```

```php
/**
 * after
 */
class AcmeBlogTest extends TestCase
{
    /**
     * @var Blog
     */
    private $blog;

    public function setUp()
    {
        $this->blog = new Blog();
    }
}
```

4.2.3. Navigation

特定のクラスへのすべての参照をリストアップできます。 実際に phpactor のプロジェクトで実行した結果は以下です。（LN: 行番号, OS: offset start, OE: offset end） ドキュメントや影響範囲の調査のときに非常に便利で、筆者はよく仕事で使います。



4.3. 拡張

phpactor はいくつかの PHP 製開発ツールとのインテグレーションを提供しています。[15]

PHPStan のインテグレーションを使えば、Phpactor 上でエラーをテキストエディタに返却することが可能ですので、わざわざ別のツールを導入する必要はあります。

php-cs-fixer のインテグレーションを使えば、 `textDocument/formatting` タイミングで php-cs-fixer を実行する、エラー警告をテキストエディタ上に表示できます。

これらのインテグレーションを有効にすることによって、Phpactor に統合でき、別のツールを導入せずに済みます。

```shell
# phpstan
$ phpactor config:set language_server_phpstan.enabled true

# php-cs-fixer
$ phpactor config:set language_server_php_cs_fixer.enabled true
```

5. 終わりに

普段、何気なく使っているソフトウェアにも歴史や時代背景があり、過去から現在に至るまで、さまざまな機能が追加・削除されてきました。 先人たちに感謝しつつ、それらを最大限活用できるよう勉強を続けていけば、次の一歩が見えてくるかもしれません。

[15] Integration - Phpactor, https://phpactor.readthedocs.io/en/master/integrations.html