テキストエディタが
PHPをシンタックスハイライトする仕組みと
モダンテキストエディタ事情について

たけてぃ(@takeokunn)

🍎Introduction

プログラミング言語は日進月歩で進化し続けています。

処理系のパフォーマンス改善、既存のバグ修正、挙動の変更などさまざまな変更が入るが、ユーザーにとって一番影響があるのは「新規の構文追加」ではないでしょうか？
PHPの場合PHP 8.0以降にmatch式やenum構文は直近5年以内に追加されたものです。
PHPに限らずほか言語のRFCでも新規の構文が提案されている例は枚挙に暇がありません。Goのジェネリクスなどもその一例です。

言語レベルで新規の構文が追加された時、テキストエディタでも適切に色付けできることが望ましいです。
それはオープンソースのテキストエディタだろうと商用エディタだろうと区別はありません。
そもそもテキストエディタはどのように構文を解釈してハイライトしてくれているのか、Tree-sitterなどの最近のテキストエディタ事情も踏まえて解説していきます。

なお私は熱狂的なEmacsユーザーでありEmacsのPHP Packageであるphp-modeのオーナー権限もあるので、Emacsひいきな解説になってしまうのはご了承ください。

🍎シンタックスハイライト概要

シンタックスハイライトとは、publicのようなキーワードや関数名や言語特有の構文に色をつけてくれるものです。Wikipediaによると、シンタックスカラリングや構文着色とも言うらしいです。

シンタックスハイライトには、「テキストの可読性を向上させ文脈をより明瞭にする」や「記述ミスや括弧の対応のミスなどを防ぐことができる」等、さまざまなメリットがあります。
明確なデメリットは私は思いつきませんが、「流し読みがしやすくなるのでプログラマーはコード全体を理解しようとはしなくなる」ということを主張する人もいるようです。

配色はカラーテーマごとに違います。世の中には無数のカラーテーマがあり、私はDraculaやSublime Textのdefault themeのmonokaiが好きです。

構文が違うので当然言語の数だけシンタックスハイライトがあります。
それぞれの実装によって方針はまちまちですが、似ている言語から上書きするよう実装することで実装コストを下げる対応をしています。PHPはCやJavaと記述が似ているので、CやJavaの実装を上書きし、部分的にPHPの構文を追加して対応できます。実際EmacsのPHP Packageではこのような対応をしています。

🍎シンタックスハイライトの大まかなしくみ

世の中にはテキストエディタの実装は無数にあります。
プログラミング言語に対応したテキストエディタの実装はおおよそ2つに大別できます。

- 正規表現ベース
- ASTベース

正規表現ベースのシンタックスハイライトはVimやEmacsのような古くからあるテキストエディタでよく使われています。当然それぞれのテキストエディタごとに実装は違うので移植性はありません。

ASTベースのシンタックスハイライト実装は2024年1月現在Tree-sitterが一強だと言っても過言ではありません。Tree-sitterはC/Rust製のツールで、特定のテキストエディタに依存しない形で実装された高速で動作するパーサジェネレータツールです。Tree-sitterはもともとと Atomで使用するためにGitHubによって開発され、2018年にリリースされました。

ソースコードをパースして構文木をS式を出力することにより、各テキストエディタはS式を解釈する実装をすれば各々が構文を解釈する必要がない、という作りになっています。

テキストエディタにおけるシンタックスハイライトの難しいところは以下が挙げられます。

- 常にユーザーが入力し続けるので構文が確定しない
- 1入力ごとにハイライトする必要がある為高速で挙動させる必要がある

また、1言語で複数言語を表現する場合難易度が上がります。たとえばVue.jsはHTML/CSS/JavaScriptを1ファイル内で記述できるし、PHPももともとHyperText PreprocessorなのでHTMLを記述できます。

🍎正規表現によるシンタックスハイライト

正規表現によるシンタックスハイライトを採用しているVimやEmacsでは以下のような実装がされています。

- キーワードは直接色を付ける
- 正規表現によって構文を定義する
- **$** の後は確実に変数
- **function**の後は確実に関数名になり、その後の括弧は関数の引数になる
- // の直後はすべてコメントになる

Emacsではシンタックステーブルというものがデフォルトで用意されており、独自の記法で記述する必要があります。以下は実際にphp-modeで実装されているコードを抜粋したものです。Emacs Lispの正規表現がそもそも難しいのもあり、複雑怪奇で特殊な訓練しないと読めないことが分かるでしょう。

Emacsではシンタックステーブルというものがデフォルトで用意されており、独自の記法で記述する必要があります。右図は実際にphp-modeで実装されているコードを抜粋したものです。Emacs Lispの正規表現がそもそも難しいのもあり、複雑怪奇で特殊な訓練しないと読めないことが分かるでしょう。

```
;; Highlight function/method names
("\\<function\\s-+&?\\(\\(?:\\sw\\|\\s_\\)+\\)\\s-*(" 1 'php-function-name)

; Support the ::class constant in PHP5.6
("\\sw+\\(::\\)\\(class\\)\\b" (1 'php-paamayim-nekudotayim) (2 'php-magical-constant))

;; Class declaration keywords (class, trait, interface)
("\\_<\\(class\\|trait\\|interface\\)\\_>" . 'php-class-declaration)
```

言語内に複数言語あるVue.jsやPHPのような言語では、Emacsの場合カーソル位置によって対象の言語に切り替える処理をしています。

正規表現ベースのシンタックスハイライトには以下のようなメリットとデメリットがあります。

- メリット
  - 低メモリで高速で動く
  - 構文を確定しなくてもハイライトできる
- デメリット
  - 正規表現の難易度が高い
  - 正規表現エンジンの実装依存になる
  - 複雑な構文を持っている言語だと実装難易度が高い
  - 各テキストエディタごとに実装する必要がある

西暦2000年以前からある機能ですので、現在のコンピュータで動かすと当然パフォーマンスが非常に良く、マシンスペックの低いコンピュータでも問題なく動くようになっています。

一方デメリットに正規表現特有の問題が挙げられます。
ひとつは正規表現エンジンはテキストエディタに内蔵されているエンジン依存になってしまうことです。
ベーシックな正規表現の記法はだいたいの実装でサポートしてくれていますが、先読み後読みなどは実装によってまちまちです。Emacs組込みの正規表現エンジンは先読み後読みのサポートをしていない為、カーソルを擬似的に動かすことによってむりやり先読みを実現する、といったテクニックが必要になってきます。
正規表現エンジンを取り替えることは基本的にはできないのでそれぞれのエディタに従うほかありません。

また、複雑な構文を持っている言語だと実装難易度が高いという点もあります。PHPのような割と簡単な単純な言語だとまだマシですが、C++のような複雑怪奇な構文をもつ言語だと正規表現で表現するのは至難の業です。

Emacsにはcc-engineというCに似た言語をまるっとシンタックスハイライトしてくれるコードを提供してくれているのですが、実装は天才が成した仕事なので我々凡人には理解するのは難しいものとなっています。

正規表現エンジンもレンダリングのしくみも違うので当然エディタごとに実装する必要があります。
世の中にプログラミング言語も機能も増えている昨今、Emacsのようなユーザー数が減っているエディタがすべての言語のバージョンアップに対応するのは厳しいという現状があります。
PHPに関しては私やtadsanが対応していくので、我々の目が黒いうちは最新の構文を使えるはずです。

🍎ASTベースによるシンタックスハイライト

ASTベースのシンタックスハイライトのしくみは2024年1月現在Tree-sitterが一強ですので、Tree-sitterを元に解説しますのでご了承ください。Tree-sitterはRust/Cで書かれていて特定のエディタに依存しない構文解析ツールです。
特定のテキストエディタに依存しないという思想はLSPと似ているので、LSPのような立ち位置のツールだと思っていただいてかまいません。

tree-sitter本体とtree-sitter-{language}のような言語ごとのgrammarを提供しています。各テキストエディタはTree-sitterのC言語部分をwrapしたうえで各エディタでシンタックスハイライトできるように実装しています。

tree-sitter-phpのgrammarを一部抜粋すると右図です。
yaccを見たことある人は馴染があるような文法で記述されています。

```
// return <expression>;
return_statement: $ => seq(
    keyword('return'), optional($._expression), $._semicolon,
),

// ++$<_variable>, <_variable>--
update_expression: $ => prec.left(PREC.INC, choice(
    seq($._variable, '++'),
    seq($._variable, '--'),
    seq('++', $._variable),
    seq('--', $._variable),
)),
```

実際に右図のPHPをtree-sitter parseした結果は下図です。
S式で表現されていてtoken情報と座標を返します。

```
<?php

final class HelloCommand extends Command
{
    public function __construct() {}
}
```

```
(program [0, 0] - [4, 36]
  (php_tag [0, 0] - [0, 5])
  (class_declaration [2, 0] - [4, 36]
    modifier: (final_modifier [2, 0] - [2, 5])
    name: (name [2, 12] - [2, 24])
    (base_clause [2, 25] - [2, 40]
      (name [2, 33] - [2, 40]))
    body: (declaration_list [3, 0] - [4, 36]
      (method_declaration [4, 4] - [4, 36]
        (visibility_modifier [4, 4] - [4, 10])
        name: (name [4, 20] - [4, 31])
        parameters: (formal_parameters [4, 31] - [4, 33])
        body: (compound_statement [4, 34] - [4, 36])))))
/var/folders/cb/3r410lh103x9hthl1pmy3jqw0000gp/T/babel-3wPZaM/tree-sitterNg42xu.php    0 ms    (MISSING "}" [4, 36] - [4, 36])
```

また、Tree-sitterは非常に賢いので構文エラーが出た場合エラーの箇所まで表示してくれます。


```
(program [0, 0] - [4, 36]
  (php_tag [0, 0] - [0, 5])
  (class_declaration [2, 0] - [4, 36]
    modifier: (final_modifier [2, 0] - [2, 5])
    name: (name [2, 12] - [2, 24])
    (base_clause [2, 25] - [2, 40]
      (name [2, 33] - [2, 40]))
    body: (declaration_list [3, 0] - [4, 36]
      (method_declaration [4, 4] - [4, 36]
        (visibility_modifier [4, 4] - [4, 10])
        name: (name [4, 20] - [4, 31])
        parameters: (formal_parameters [4, 31] - [4, 33])
        body: (compound_statement [4, 34] - [4, 36])))))
/var/folders/cb/3r410lh103x9hthl1pmy3jqw0000gp/T/babel-3wPZaM/tree-sitterNg42xu.php    0 ms    (MISSING "}" [4, 36] - [4, 36])
```


1言語内に複数言語の場合、特定のtoken内は別のgrammarを適用するという処理を書けるというのもTree-sitterの特徴です。

Tree-sitterによるシンタックスハイライトには以下のようなメリットとデメリットがあります。

- メリット
  - メジャーな言語はだいたいサポートされている
  - エディタごとの実装をする必要ないのでメンテナンスされる可能性が高い
- デメリット
  - 構文が確定するまで色がつかない
  - 毎回ASTを作る必要があるので正規表現と比べて低速
  - テキストエディタ本体はTree-sitterのサポートをし続けないといけない

メリットとしてサポートしている言語もテキストエディタも多いことが挙げられます。
2024年現在使われているプログラミング言語のだいたいのgrammarは公式から提供されています。
Neovimも標準でサポートしており、Emacsでも29からサポートされました。

デメリットとしてはASTとして解釈することに由来するものが挙げられます。
テキストエディタでコードを編集している間構文が確定しない、構文エラーの時間が時間が必ず発生します。
Tree-sitterは構文エラーを最小限にするようなアルゴリズムが採用されていますが、正規表現と比べてどうしても色が付かない時間が発生してしまいます。
また、テキストを編集する毎にASTを作る必要があるので、正規表現で色付けするよりも当然計算コストがかかります。

Tree-sitterを使うとなるとCレイヤを触る必要があります。
基本的にCレイヤをテキストエディタ側は変更することは意図していないので、通常のPackageと違って何か問題が起きた時に修正しつらいという問題もあります。

🍎終わりに

プログラマーが快適にプログラムを編集するには、プログラミング言語の進化にエディタも追従する必要があります。
過去の資産と向き合いながら、新しい技術と上手に付き合っていくことが求められています。

ぜひ普段使ったことのないテキストエディタを使ったり、新しいプラグインにチャレンジしてみてはいかがでしょうか。

🍎終わりに

シンタックスハイライト - Wikipedia
https://ja.wikipedia.org/wiki/%E3%82%B7%E3%83%B3%E3%82%BF%E3%83%83%E3%82%AF%E3%82%B9%E3%83%8F%E3%82%A4%E3%83%A9%E3%82%A4%E3%83%88

Tree-Sitter公式サイト
https://tree-sitter.github.io/

tree-sitter/tree-sitter-php - GitHub
https://github.com/tree-sitter/tree-sitter-php

emacs-php/php-mode - GitHub
https://github.com/emacs-php/php-mode

emacsemacs-mirror/emacs - GitHub
https://github.com/emacs-mirror/emacs