ChatGPT

Парсинг PHP з Tree-sitter у Python

Щоб аналізувати PHP-код, використовують Tree-sitter із відповідною граматикою. Наприклад, можна встановити пакет tree-sitter-languages і далі у Python так:

```
from tree_sitter_languages import get_language, get_parser
language = get_language('php')      # отримати мову PHP
parser = get_parser('php')          # отримати парсер для PHP
tree = parser.parse(bytes(php_code, "utf8"))
```

У цьому прикладі `parser` автоматично налаштовує мову, тому можна одразу викликати `parser.parse(...)` [1]. Після виклику отримуємо об'єкт `tree`, а кореневий вузол – `tree.root_node`. Усі вузли мають методи типу `child_by_field_name()`, `children`, `named_children` тощо [2].

Пошук викликів `$this->...Response()`

Щоб знайти виклики методів типу `$this->listResponse($arg)`, підійде запит (Query) по шаблону Tree-sitter. У граматиці PHP такий виклик відображається як `method_call_expression`, де поле `object` – `$this`, а поле `name` – ім'я методу. Наприклад, запит може бути таким (з Syntax S-expression):

```
(method_call_expression
    object: (variable_name) @object
    name:   (name)          @method
)
```

Через Python це реалізують аналогічно. Наприклад, за допомогою API Tree-sitter:

```
from tree_sitter import Query, QueryCursor

query = Query(language, """
  (method_call_expression
      object: (variable_name) @object
      name:   (name)          @method
  )
""")
cursor = QueryCursor()
for match in cursor.matches(query, tree.root_node, bytes(php_code, "utf8")):
```

```
    obj_node = None
    method_node = None
    for capture in match.captures:
        if capture.index == 0:
            obj_node = capture.node
        elif capture.index == 1:
            method_node = capture.node
    if obj_node and method_node:
        obj = obj_node.text.decode()     # наприклад "$this"
        method = method_node.text.decode()  # ім'я методу, наприклад
"listResponse"
        if obj == '$this' and method.endswith('Response'):
            args = capture.node.parent.child_by_field_name('arguments')
            print(f"Виклик методу: {obj}->{method}, аргументи:
{args.text.decode()}")
```

Цей приклад ілюструє створення запиту, що ловить усі виклики типу `X->method(...)`. Далі у циклі фільтрується `obj == '$this'` та перевіряється, що `method` закінчується на `"Response"`. Цей підхід подібний до прикладу в Rust: там показано використання `Query::new` з аналогічним шаблоном (наприклад `(method_call_expression object: (variable_name) @object name: (name) @method)`), де далі обходять матчинги і читають зловлені вузли [3] [4].

Проходження AST і визначення імені методу

В Python-API після парсингу можна ітерувати AST або використовувати Query для конкретних шаблонів. Наприклад, альтернативно до Query можна рекурсивно обійти дерево:

```
def traverse(node):
    yield node
    for child in node.children:
        yield from traverse(child)

for node in traverse(tree.root_node):
    if node.type == 'method_call_expression':
        object_node = node.child_by_field_name('object')
        name_node = node.child_by_field_name('name')
        if object_node and name_node:
            obj = object_node.text.decode()
            meth = name_node.text.decode()
            if obj == '$this' and meth.endswith('Response'):
                print("Знайдено:", obj, "->", meth)
```

Тут `child_by_field_name('name')` дає вузол з ім'ям методу. У схожому прикладі на Rust для глобальних функцій використовували `child_by_field_name("name")` та `"parameters"`,

`"return_type"` тощо [5] [6]. Аналогічно, у PHP-методу ім'я знаходиться в полі `name`, параметри – в `parameters`.

Знаходження визначення методу і аналіз повернення

Після того, як ми знаємо ім'я викликаного методу (наприклад `"listResponse"`), треба знайти його визначення. У PHP-gramatici методи в класах відповідають вузлам `method_definition`, а ім'я методу – вузлу типу `property_identifier`. Отже, можна використати запит типу:

```
query_def = Query(language, """
  (method_definition
      name: (property_identifier) @method_name
  )
""")
cursor = QueryCursor()
method_defs = {}
for match in cursor.matches(query_def, tree.root_node, bytes(php_code, "utf8")):
    name_node = match.captures[0].node
    method_name = name_node.text.decode()
    method_defs[method_name] = name_node.parent  # вузол method_definition
```

Таким чином збираються усі методи класів. Після цього можемо знайти той, чиїй назві відповідає наше викликане ім'я. (Зауважимо, що в одному файлі може бути декілька класів, тому треба відповідно структурувати пошук. У фреймворку Laravel метод контролера зазвичай визначений у тому самому файлі.)

На StackOverflow наведено приклад подібного запиту – для поєднання полів виведення полів і методів класу використовували `(method_definition name: (property_identifier) @method-name)` [7]. За аналогією ми застосуємо такий паттерн для PHP.

Витягнення типу або структури повернення

Після того, як знайдено вузол `method_definition` для конкретного методу, можна дослідити його тіло. По-перше, якщо у PHP-метода в оголошенні є вказаний тип повернення (PHP 7+), то він доступний через поле `return_type`. У Tree-sitter це робиться так:

```
return_type_node = method_node.child_by_field_name('return_type')
if return_type_node:
    return_type = return_type_node.text.decode()
    print("Декларований тип повернення:", return_type)
```

Також можна перевірити всі оператори `return` усередині тіла методу. Наприклад, обійти дітей `method_node` у пошуку вузлів `return_statement`:

```
for subnode in traverse(method_node):
    if subnode.type == 'return_statement':
        expr = subnode.child_by_field_name('argument')
        print("Знайдено return:", expr.type, "-> текст:", expr.text.decode())
```

Наприклад, якщо метод повертає `new JsonResource(...)`, то дерево покаже вузол типу `object_creation_expression`, а якщо повертає масив – вузол `array_creation_expression`. Таким чином можна зрозуміти, що метод формує відповідь у вигляді JSON-релікворсу або масиву. Зверніть увагу, що стандартних полів `return_type` у старому PHP-коді може не бути, тоді аналіз слід вести по синтаксису `return`.

У підсумку: поєднуючи запити Tree-sitter та обхід AST, можна автоматично знаходити у Laravel-коді виклики `$this->SomethingResponse(...)`, визначати ім'я цього методу та досліджувати, що він повертає – через явний тип у сигнатурі або перевірку `return`-виразів.

Джерела: Використання py-tree-sitter із завантаженням мов описано у документації та прикладах (наприклад, py-tree-sitter-languages [1] або TIL-блог Simon Willison [8]). Приклади запитів `method_call_expression` та `method_definition` показані у прикладах (див. [21†L568-L578] та [47†L154-L158]). Інтерфейс Python для `child_by_field_name` описано у офіційному API (див. [29†L82-L87]).

---

[1] GitHub - grantjenks/py-tree-sitter-languages: Binary Python wheels for all tree sitter languages.
https://github.com/grantjenks/py-tree-sitter-languages

[2] [8] Using tree-sitter with Python | Simon Willison's TILs
https://til.simonwillison.net/python/tree-sitter

[3] [4] [5] [6] Rust语法分析工具tree-sitter-php的使用，PHP语言解析库tree-sitter-php实现高效代码解析与语法高亮
https://bbs.itying.com/topic/689c55982cb460013cc11736

[7] treesitter - How to use `or` logic in tree-sitter query syntax? - Stack Overflow
https://stackoverflow.com/questions/78910790/how-to-use-or-logic-in-tree-sitter-query-syntax