Санкт-Петербургский Государственный Университет
Кафедра системного программирования

Разработка решения для поддержки новых парсеров и языков программирования в инструменте Astminer

Автор: И.Н. Уткин, 19.Б10-мм
Научный руководитель: к.т.н., доцент Т.А. Брыксин
Консультант: программист-исследователь JetBrains Е.С. Спирин

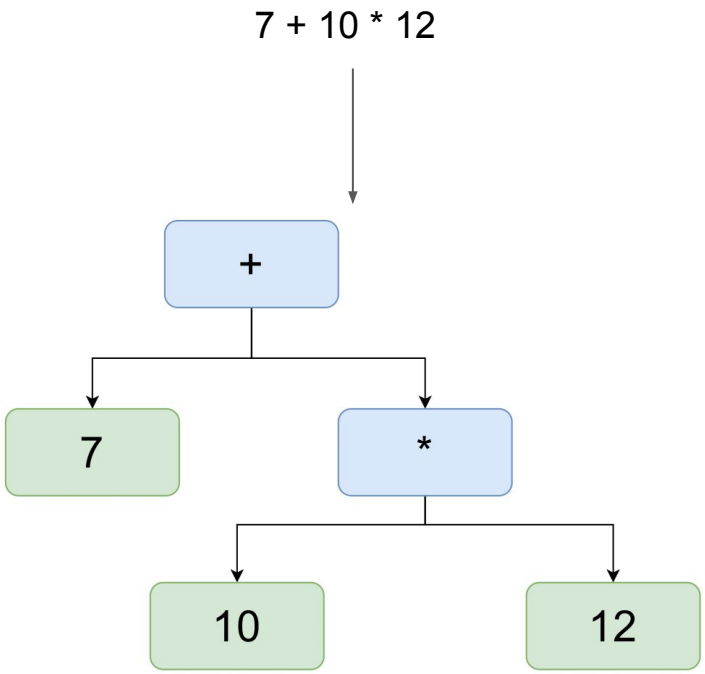
Санкт-Петербург, 2021

Мотивация

- Тяжело сопровождать программные продукты
- Для помощи программистам можно использовать машинное обучение
  - *N. Maneerat and P. Muenchaisri, "Bad-smell prediction from software design model using machine learning techniques,"*
  - *J. Abbineni and O. Thalluri, "Software Defect Detection Using Machine Learning Techniques,"*
- Для обучения модели нужны данные
- Важным параметром является вид представления исходного кода
  - Можно работать с кодом как с естественным языком
    - *Michael D. Ernst “Natural Language is a Programming Language: Applying Natural Language Processing to Software Development”*
  - Сегодня одни из самых популярных видов представления являются абстрактные синтаксические деревья и их производные
    - *Zhensu Sun, Yan Liu, Chen Yang, Yu Qian “PSCS: A Path-based Neural Model for Semantic Code Search”*
    - *Miltiadis Allamanis, Marc Brockschmidt, Mahmoud Khademi “Learning to Represent Programs with Graphs”*

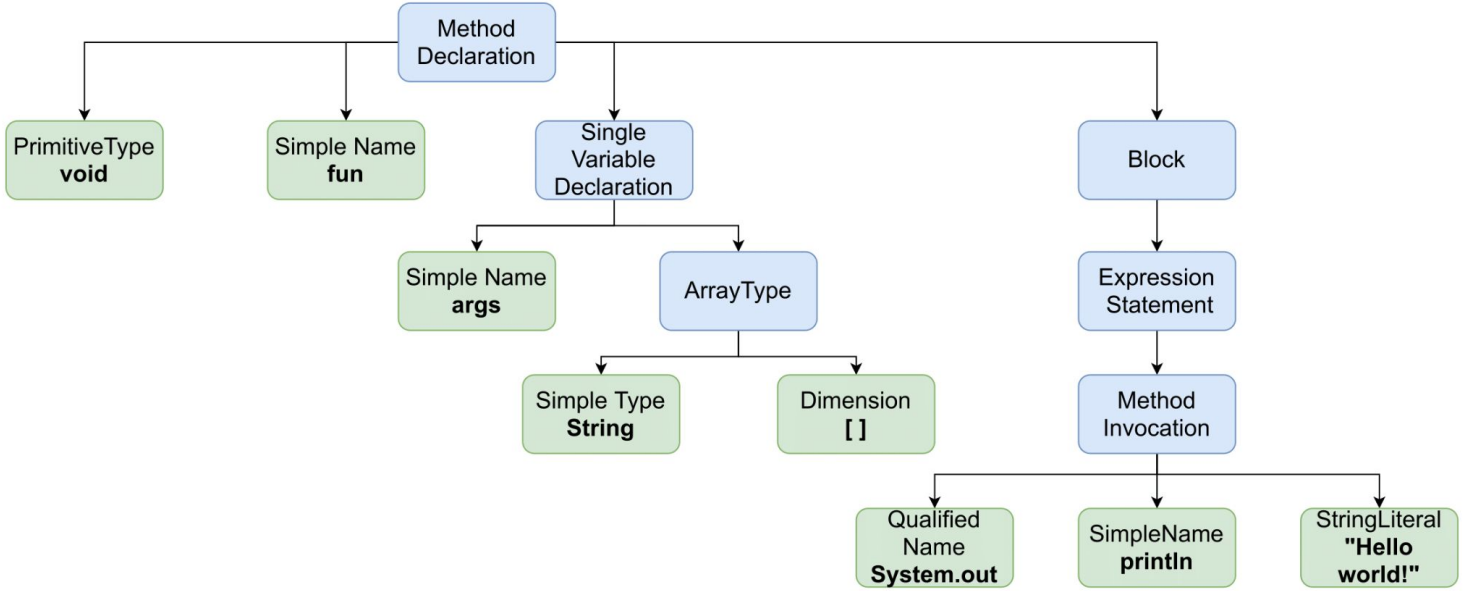
Абстрактное синтаксическое дерево

```
grammar Exp;
eval
    :    additionExp
    ;
additionExp
    :    multiplyExp
         ( '+' multiplyExp
         | '-' multiplyExp
         )*
    ;
multiplyExp
    :    Number
         ( '*' Number
         | '/' Number
         )*
    ;
Number
    :    ('0'..'9')+ ('.' ('0'..'9')+)?
    ;
WS
    :   (' ' | '\t' | '\r'| '\n') {channel=HIDDEN;}
    ;
```



Дерево функции

```
void fun(String[] args) {
    System.out.println("Hello world!");
}
```



Существующие парсеры AST

- ANTLR — инструмент, позволяющий преобразовывать грамматики в парсеры на Java, Python и другие языки

  Официальная страница: antlr.org

- GumTree — фреймворк для работы с исходным кодом в виде деревьев и вычисления различий между ними

  Страница на Github: github.com/GumTreeDiff

- TreeSitter — инструмент генератора парсеров и библиотека инкрементного синтаксического анализа

  Страница на Github: github.com/tree-sitter

Astminer

- Инструмент для конвертации исходного кода в различные представления
- Объединяет несколько парсеров и языков для разбора
- Позволяет фильтровать некоторые вершины
- Может доставать функции из деревьев и собирать о них информацию (название функции, модификаторы, параметры и т.д.)

github.com/JetBrains-Research/astminer

Недостатки:

➔ Сложно масштабировать на новые языки и парсеры

Пример плохо масштабируемого кода

```
override fun toLabeledData(
        parseResult: ParseResult<out Node>
): List<LabeledResult<out Node>> {
    val (root, filePath) = parseResult
    val fileExtension = File(filePath).extension

    val functionInfos = when (fileExtension) {
        "c", "cpp" -> { ... }
        "java" -> { ... }
        "py" -> { ... }
        "js" -> { ... }
        else -> { ... }
    }.filter {...}

    return functionInfos.mapNotNull {...}
}
```

```
fun getParser(
    extension: String,
    javaParser: String
): Parser<out Node> {
    return when (extension) {
        "java" -> { ... }
        "c" -> FuzzyCppParser()
        "cpp" -> FuzzyCppParser()
        "py" -> PythonParser()
        "js" -> JavaScriptParser()
        else -> { ... }
    }
}
```

Плохая масштабируемость фильтров

- Любой фильтр должен уметь разбирать полученное дерево в зависимости от грамматики
- Усложняется добавление новых фильтров и их поддержка

```
class ModifierFilterPredicate(private val excludeModifiers: List<String>)
    : MethodFilterPredicate() {

    // TODO: add other parsers

    private fun gumTreeModifierFilter(root: Node?): Boolean { ... }

    override fun isFiltered(methodInfo: MethodInfo<out Node>): Boolean {
        return gumTreeModifierFilter(methodInfo.method.root)
    }
}
```

Цель и задачи

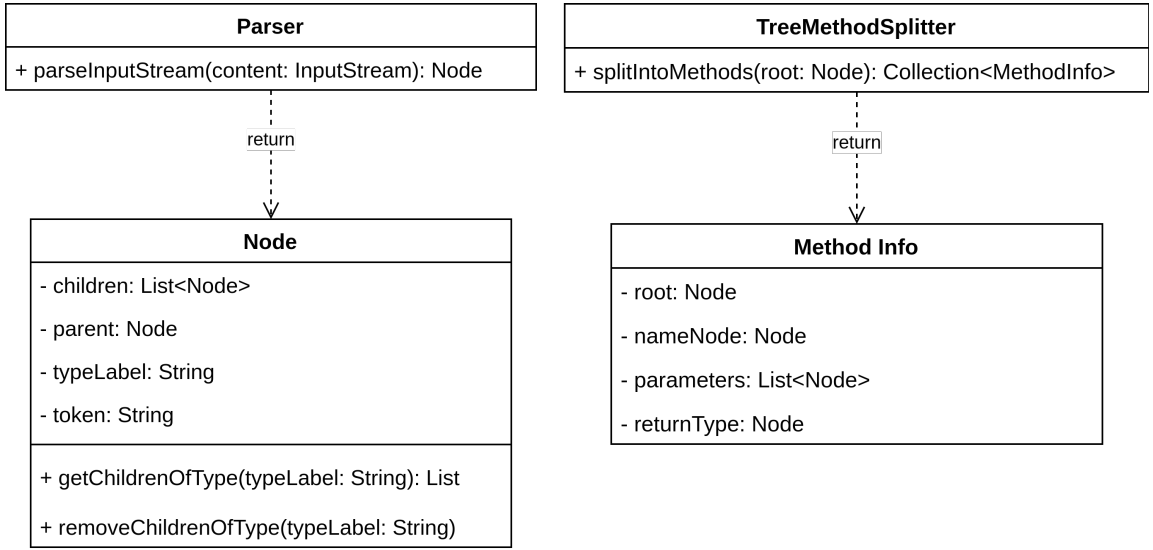
Цель: создать единый API, который позволит объединить специфичные для разбираемых языков компоненты Astminer, для дальнейшего масштабирования на новые языки и парсеры

Задачи:

1. Изучить предметную область и текущую архитектуру Astminer
2. Спроектировать и реализовать решение, позволяющее поддерживать новые парсеры и языки программирования в не зависимости от их числа
3. Интегрировать полученное решение в существующий инструмент
4. Провести апробацию внедренного решения

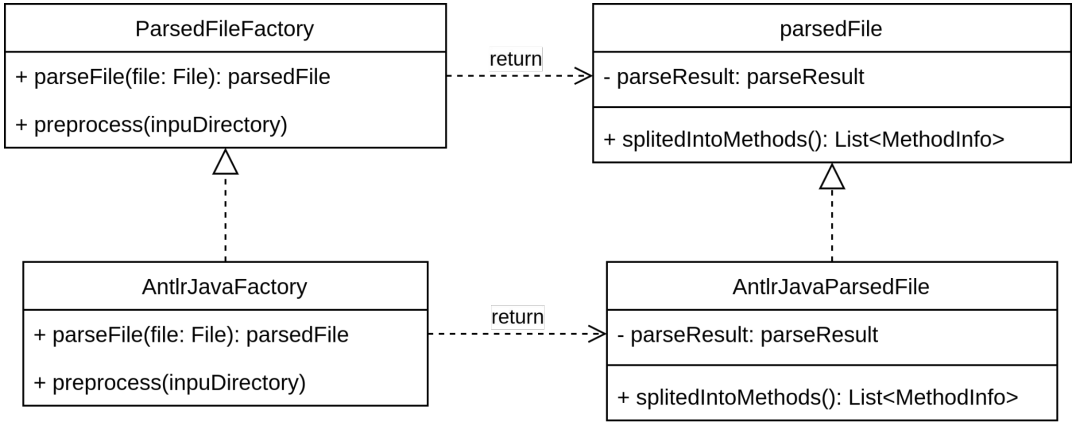
Текущая архитектура разбора

- AST — набор экземпляров Node
- Parser и Splitter уникальны для грамматик
- Не объединены между собой, хотя любой сплиттер работает только с конкретным парсером



Фабрики для работы с языками и парсерами

- Применим паттерн Абстрактная фабрика
- Объединим логику, отвечающую за парсинг и разбиение
- Фабрики работают на уровне всего проекта, создаваемые объекты на уровне файлов



Разбиение на функции

- Заменим Method на Function
- Вместо вершин будем хранить информацию в виде, независящим от грамматики
- Пусть FunctionInfo уже знает к какой грамматике она принадлежит
- По умолчанию любое поле бросит исключение, наследники переопределяют поля
- Для получения большей информации о функции достаточно добавить новое поле и переписать это поле в наследнике

MethodInfo
- root: Node - nameNode: Node - parameters: List<Node> - returnType: Node

FunctionInfo
- root: Node - name: String - modifiers: List<String> - annotations: List<String> - parameters: List<MethodInfoParameter> - returnType: String - enclosingElement: EnclosingElement - isConstructor: Boolean

Апробация

- Для апробации внедренного решение был добавлен язык PHP, так как уже давно существовал запрос на него

```
object AntlrPHPParsedFileFactory: ParsedFileFactory {
    override fun createParsedFile(file: File) = AntlrPHPParsedFile(file)

    class AntlrPHPParsedFile(file: File): ParsedFile<AntlrNode>() {
        override val parseResult = PHPParser().parseFile(file)
        override val splitter = PHPFunctionSplitter()
    }
}
```

- После создания парсера и сплиттера для интеграции нового языка была создана соответствующая фабрика и добавлена всего одна строка кода в существующую функцию

```
private fun getAntlrParsedFileFactory(extension: String): ParsedFileFactory {
    return when (extension) {
        "java" -> AntlrJavaParsedFileFactory
        "javascript" -> AntlrJavascriptParsedFileFactory
        "python" -> AntlrPythonParsedFileFactory
        "php" -> AntlrPHPParsedFileFactory
        else -> { ... }
    }
}
```

Заключение

В ходе данной работы были достигнуты следующие задачи:

1. Были изучена предметная область и архитектура приложения Astminer
2. Спроектирован единый API для добавления новых языков и парсеров
3. Решение было интегрировано и сопровождено тестами и документаций
4. В качестве апробации была добавлена поддержка языка PHP