Теория и практика парсинга формальных языков

- Иван Кочуркин
- Работаю в Positive Technologies над открытым универсальным сигнатурным анализатором кода PT.PM
- Подрабатываю в Swiftify, веб-сервисе для конвертинга кода Objective-C в Swift
- Веду активную деятельность на GitHub
- Пишу статьи на Хабре и GitHub под ником KvanTTT

Почему не Regex?

1. `<table>(.*?)</table>`
2. А если аттрибуты? `<table.*?>(.*?)</table>`
3. А если элементы? `tr`, `td`
4. А если комментарии? `<!-- html comment -->`
5. ...
6. NO NOOOO NO stop the an*gles are not real ZA̡LG̡O IS T̡ON̡y THE PONY HE COMES

Лексемы и токены

- **Лексема** - распознанная последовательность символов
- **Токен** - лексема + тип

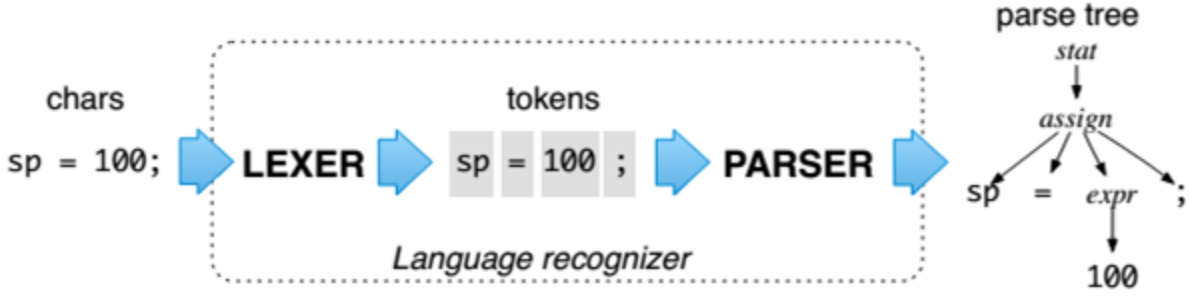
```
MyKeyword: 'var';
Id:        [a-z]+;
Digit:     [0-9]+;
Comment:   '<!--' .*? '-->';
```

- Пример:

  `TagOpen(<) Identifier(html) TagClose(>) Whitespace()`

Дерево разбора и AST

- **Дерево разбора** - древовидная структура, распознанная из потока токенов
- **AST** - дерево разбора без пробелов, точек с запятых и т.д.



Типы парсеров

- Готовые библиотеки парсинга (regex, JSON.NET)
  - API
  - Только самые распространенные языки
- Парсеры, написанные вручную (Roslyn)
  - Большие возможности и гибкость
  - Большой порог вхождения
  - Медленная скорость разработки
- Автоматически сгенерированные парсеры (ANTLR)
  - Порог вхождения
  - Быстрая скорость разработки после освоения
  - Меньшая гибкость по сравнению с ручными парсерами

Грамматика

Формальное описание языка, которое может быть использовано для распознавания его структуры.

Пример грамматики

```
expr
    : expr '*' expr
    | expr '+' expr
    | ID '(' args ')'
    | ID
    ;

ID: [a-zA-Z]+;
```

Пример данных

```
a + b * c
```

Типы языков

Иерархия Хомского

- Регулярные
- Контекстно-свободные
- Контекстно-зависимые
- Тьюринг-полные

Пример КС-КЗ конструкции: `T a = new T()`

Инструменты и библиотеки под C#

- Генераторы
  - Контекстно-свободные (ANTLR, Coco/R, Gardens Point Parser Generator, Grammatica, Hime Parser Generator, LLLPG)
  - Безлексерные PEG (IronMeta, Pagarus)
- Комбинаторы (Parseq, Parsley, LanguageExt.Parsec, Sprache, Superpower)
- Языковые фреймворки (JetBrains MPS, Nitra, Roslyn)

Детальное описание: Parsing In C#: Tools And Libraries.

Парсер-комбинаторы

- Использование внутри языка разработки (C#)
- Использование в IDE

Библиотеки:

- Sprache
- Superpower

Примеры кода парсер-комбинатора

```
// Parse any number of capital 'A's in a row
var parseA = Parse.Char('A').AtLeastOnce();
```

Правило для `id`:

```
Parser<string> identifier =
    from leading in Parse.WhiteSpace.Many()
    from first in Parse.Letter.Once()
    from rest in Parse.LetterOrDigit.Many()
    from trailing in Parse.WhiteSpace.Many()
    select new string(first.Concat(rest).ToArray());

var id = identifier.Parse(" abc123  ");

Assert.AreEqual("abc123", id);
```

Проблемы и задачи парсинга

На основе ANTLR и Roslyn.

- Неоднозначность
- Контекстно-зависимые конструкции
- Регистронезависимость
- Островные языки и конструкции
- Скрытые токены
- Препроцессорные директивы
- Парсинг фрагментов кода
- Обработка и восстановление от ошибок

Неоднозначность

Пример: `var var = 100500;`

Решение с помощью грамматики

```
// Lexer
VAR: 'var';
ID:  [0-9a-zA-Z];

// Parser
varDeclaration
    : VAR identifier ('=' expression)? ';'
    ;

identifier
    : ID
    // Other conflicted tokens
    | VAR;
```

Объектный конструктор в C#

```
class Foo
{
    public string Bar { get; set; }
}
public string Bar { get; set; } = "Bar2";

...

foo = new Foo
{
    Bar = Bar // Umbiguity here
};
```

Какой результат возвращает `nameof`?

```
class Foo
{
    public string Bar { get; set; }
}

static void Main(string[] args)
{
    var foo = new Foo();
    WriteLine(nameof(foo.Bar));
}
```

`nameof` как функция и оператор

```
class Foo
{
    public string Bar { get; set; }
}

static void Main(string[] args)
{
    var foo = new Foo();
    WriteLine(nameof(foo.Bar));
}

static string nameof(string str)
{
    return "42";
}
```

Неоднозначность: решение с использованием вставок кода

- **Действия** - производят вычисления на целевом языке парсера.
- **Семантические предикаты** - возвращают результат.

```
// Lexer
ID:  [0-9a-zA-Z];

// Parser
varDeclaration
    : id {_input.Lt(-1).Text == "var"}? id ('=' expression)? ';
    ;

id
    : ID;
```

Контекстно-зависимые конструкции

Heredoc в PHP или интерполируемые строки в C#

```
<?php
    echo <<< HeredocIdentifier
Line 1.
Line 2.
HeredocIdentifier
;
```

Решение

Использование вставок кода, смотри лексер PHP.

$"Интерполируемые строки в C# {2+2*2}"

```
WriteLine($"{p.Name} is \"{p.Age} year{(p.Age == 1 ? "" : "s")}
WriteLine($"{(p.Age == 2 ? $"{new Person { } }" : "")}");
WriteLine($@"\{p.Name}
            ""\");
WriteLine($"Color[R={func(b: 3):#0.##}, G={g:#0.##}, B={b:#0.##
```

Реализация в лексере C#.

Регистронезависимость

Языки: Delphi, T-SQL, PL/SQL и другие.

Решение с помощью грамматики

Фрагментный токен облегчает запись других токенов.

```
Abstract:           A B S T R A C T;
BoolType:           B O O L E A N | B O O L;
BooleanConstant:    T R U E | F A L S E;

fragment A: [aA];
fragment B: [bB];
```

Без использования фрагметных токенов:

```
Abstract:           [Aa][Bb][Ss][Tt][Rr][Aa][Cc][Tt];
```

Регистронезависимость: решение на уровне рантайма

```
Abstract:           'ABSTRACT';
BoolType:           'BOOLEAN' | 'BOOL';
BooleanConstant:    'TRUE' | 'FALSE';
```

Используется CaseInsensitiveInputStream.

Чувствительные к регистру токены обрабатываются на этапе обхода дерева. Например `$id` и `$ID` в PHP.

Достоинства:

- Код лексера чище и проще
- Производительность выше

Островные языки и конструкции

JavaScript внутри PHP или C# внутри Aspx.

```
<?php
<head>
  <script type="text/javascript">
    document.body.innerHTML="<svg/onload=alert(1)>"
  </script>
</head>
```

Островные языки и конструкции

- Использовать режимы переключения лексем `mode`.
- Сначала парсинг **PHP**. Текст внутри тегов `<script>` - обычная строка.
- Затем парсинг **JavaScript** во время обхода дерева.

PHP: альтернативный синтаксис

Смесь блоков кода на HTML и PHP

```
<?php switch($a): case 1: // without semicolon?>
        <br>
    <?php break ?>
    <?php case 2: ?>
        <br>
    <?php break;?>
    <?php case 3: ?>
        <br>
    <?php break;?>
<?php endswitch; ?>
```

Использование отдельного режима лексем для JavaScript

```
//SCRIPT_BODY: .*? '</script>'; // "Жадная" версия
SCRIPT_BODY:   ~[<]+;
SCRIPT_CLOSE:  '</script>' -> popMode;
SCRIPT_DUMMY:  '<' -> type(SCRIPT_BODY);
```

- `pushMode` - зайти в другой режим распознавания лексем (JavaScript -> PHP)
- `popMode` - выйти из режима (PHP -> JavaScript)
- `type` - изменить тип токена
- `channel` - поместить токен в изолированный канал (пробелы, комментарии)

Обработка `/*комментариев*/` и пр·б·лов

- Включение скрытых токенов в грамматику

```
declaration:
    property COMMENT* COLON COMMENT* expr COMMENT* prio?;
```

- Связывание скрытые токенов с правилами грамматики (ANTLR, Swiftify)
- Связывание скрытых токенов с основными (Roslyn)

Связывание скрытых токенов с узлами дерева (Swiftify)

Предшествующие (Preceding)

```
//First comment
'{' /*Preceding1*/ a = b; /*Preceding2*/ b = c; '}'
```

Последующие (Following)

```
'{' a = b; b = c; /*Following*/ '}' /*Last comment*/
```

Токены-сироты (Orphans)

```
'{' /*Orphan*/ '}'
```

Связывание скрытых токенов со значимыми (Roslyn)

Типы узлов Roslyn

- **Node** - не конечный узел дерева, содержащий детей
- **Token** - конечный узел (keyword, id, литерал, пунктуация)
- **Trivia** - скрытый токен без родителя, связывается с `Token`.
  - Лидирующие (**Leading**)
  - Замыкающие (**Trailing**)

```
// leading 1 (var)
// leading 2 (var)
var foo = 42; /*trailing (;)*/ int bar = 100500; //trailing (;)

// leading (EOF)
EOF
```

Препроцессорные директивы (Roslyn)

```
bool trueFlag =
#if NETCORE
    true
#else
    new Random().Next(100) > 95 ? true : false
#endif
;
```

Лидирующие для `true`

```
#if NETCORE
····
```

Лидирующие для `;`

```
#else
····new Random().Next(100) > 95 ? true : false
#endif
```

Препроцессорные директивы: одноэтапная обработка (Swiftify)

- Одновременный парсинг директив и основного языка.
- Каналы для изоляции токенов препроцессорных директив.

Интерпретация и обработка макросов вместе с функциями:

Objective-C

```
#define DEGREES_TO_RADIANS(degrees) (M_PI * (degrees) / 180)
```

Swift

```
func DEGREES_TO_RADIANS(degrees: Double)
   -> Double { return (.pi * degrees)/180; }
```

Пример на Swiftify.

Препроцессорные директивы: двухэтапная обработка (Codebeat)

```
bool·trueFlag·=
·········
····true
·····
·····································
·····
;
```

1. Токенизация и разбор кода препроцессорных диреткив.
2. Вычисление условных директив `#if` и компилируемых блоков кода.
3. Замена директив из исходника на пробелы.
4. Токенизация и парсинг результирующего текста.

Парсинг фрагментов кода (Swiftify)

Задача: определение корректного правила для фрагмента кода

Применение

- Конвертинг кода в плагине IDE (в Swiftify для плагинов к AppCode, XCode)
- Определение языка программирования (в PT.PM для редактора Approof)

Решение

- Регулярные выражения
- Токенизация и операции с токенами
- Парсинг разными правилами

Примеры: утверждения, декларации методов, свойства.

~~А~~Ошибки парсинга

Лексическая ошибка

```
class # T { }
```

Отдельный канал: `ERROR: . -> channel(ErrorChannel)`

Ошибки парсинга

Отсутствующий и лишний токены

```
class T { // Отсутствующий токен
class T ; { } // Лишний токен
```

Несколько лишних токенов (режим «паники»)

```
class T { a b c }
```

Отсутствующее альтернативное подправило

Ошибки парсинга в Roslyn

```
namespace App
{
    class Program
    {
        static void void Main(string[] args)  // Invalid token
        {
            a                   // ';' expected
            string s = """;  // Newline in constant
            char c = '';     // Empty character literal
        }
    }
}
```

Уязвимость goto fail

```
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
// ...
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;  /* MISTAKE! THIS LINE SHOULD NOT BE HERE */
```

Способы выявления:

- Анализ достоверного дерева разбора (full fidelity)
- Анализ графа потока управления (CFG)

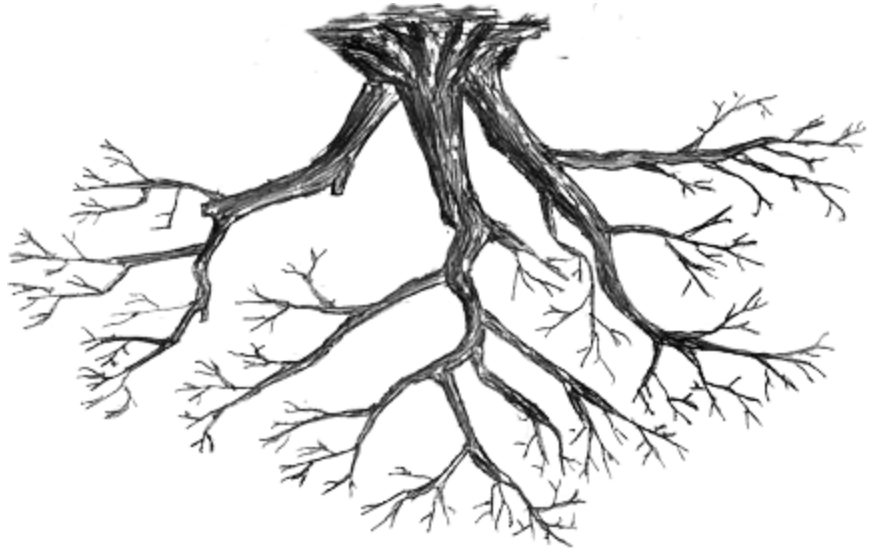
Заключение о парсинге

О чем не будет рассказываться:

- Форматирование кода (антипарсинг)
- Автокомплит
- Производительность

Обработка древовидных структур

- Методы обхода
- Архитектура и реализации Visitor и Listener
- Фичи C# 7 на практике
- Оптимизации



Методы обхода деревьев

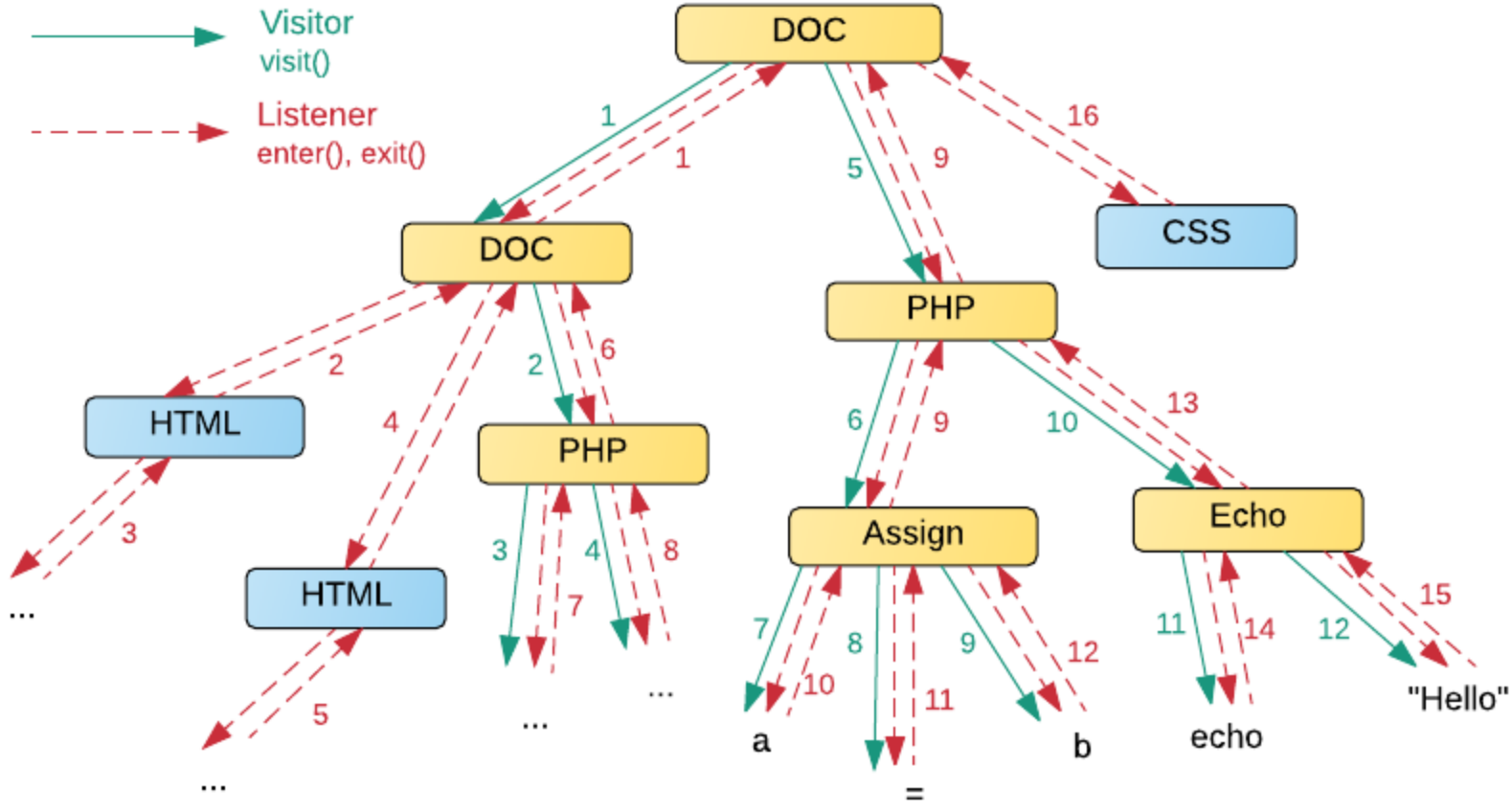
Посетитель (Visitor)

- Тип возвращаемого значения для каждого правила. Например, `string` для конвертера исходных кодов (Swiftify).
- Выборочный обход дочерних узлов.

Слушатель (Listener)

- Посещает все узлы.
- Ограниченный функционал: можно использовать для подсчета простых метрик кода.

Visitor & Listener



Реализации Visitor

Написанные вручную

- Долгая и утомительная разработка

Сгенерированные (ANTLR)

- Избыточность и нарушение Code Style
- Доступны не всегда
- Универсальность (Java, C#, Python2|3, JS, C++, Go, Swift)
- Скорость разработки

Динамические

- Медленная скорость, но хорошо для прототипов

Диспетчеризация с использованием рефлексии

```
// invocation in VisitChildren
Visit((dynamic)customNode);

// visit methods

public virtual T Visit(ExpressionStatement expressionStatement)
{
    return VisitChildren(expressionStatement);
}

public virtual T Visit(StringLiteral stringLiteral)
{
    return VisitChildren(stringLiteral);
}
```

Архитектура Visitor

Архитектура

- Несколько маленьких. Создание визиторов при необходимости.
- Один большой с использованием `partial` классов.

Формализация

Перегрузка всех методов и использование "заглушек"

```
throw new ShouldNotBeVisitedException(context);
```

C# 7: локальные функции

```
public static List<Terminal> GetLeafs(this Rule node)
{
    var result = new List<TerminalNode>();
    GetLeafs(node, result);

    // Local function
    void GetLeafs(Rule localNode, List<Terminal> localResult)
    {
        for (int i = 0; i < localNode.ChildCount; ++i)
        {
            IParseTree child = localNode.GetChild(i);
            // Is expression
            if (child is TerminalNode typedChild)
                localResult.Add(typedChild);
            GetLeafs(child, localResult);
        }
    }
    return result;
}
```

Оптимизации

Мемоизация

- Поиск первого потомка определенного типа `FirstDescendantOfType`.
- Хранение всех потомков для каждого узла дерева вместо их поиска.
- Увеличение производительности в 2-3 раза.
- Увеличение потребления памяти в 3 раза.

Уменьшение аллокаций

- Метод `Visit` - базовый, он часто вызывается.

Ресурсы

- Исходники презентации и примеров.
- Статьи:
  - Теория и практика парсинга исходников с помощью ANTLR и Roslyn
  - Обработка препроцессорных директив в Objective-C
- Движок поиска по шаблонам PT.PM и грамматики grammars-v4 (PL/SQL, T-SQL, PHP, C#, Java8, Objective-C).

Выводы

- Парсинг - это не так просто как кажется
- Существуют языки с различной выразительностью и синтаксисом
- На C# можно пользоваться разными методами и библиотеками парсинга
- ANTLR предоставляет широкие возможности по разработке парсеров
- Roslyn очень продуман в построении дерева, но только C# и VB
- Деревья можно обрабатывать разными способами

Вопросы?

Positive Technologies на GitHub

Swiftify