Lab1 Week4

王宁森 周子轩
22307130058 22307130401

截图与输出

```
eunice@eunice-VMware:~/Desktop/compiler/week4$ tree-sitter parse week4case.ts
(program [0, 0] - [3, 1]
  (statement [0, 0] - [0, 12]
    (declaration [0, 0] - [0, 12]
      (variable_declaration [0, 0] - [0, 12]
        name: (identifier [0, 4] - [0, 7])
        value: (expression [0, 10] - [0, 11]
          (number [0, 10] - [0, 11])))))
  (statement [1, 0] - [3, 1]
    (for_statement [1, 0] - [3, 1]
      init: (for_var_declaration [1, 5] - [1, 22]
        name: (identifier [1, 9] - [1, 10])
        type: (type_annotation [1, 10] - [1, 18]
          (primitive_type [1, 12] - [1, 18]))
        value: (expression [1, 21] - [1, 22]
          (number [1, 21] - [1, 22])))
      condition: (expression [1, 24] - [1, 30]
        (binary_expression [1, 24] - [1, 30]
          left: (expression [1, 24] - [1, 25]
            (identifier [1, 24] - [1, 25]))
          right: (expression [1, 29] - [1, 30]
            (number [1, 29] - [1, 30]))))
      update: (expression [1, 32] - [1, 35]
        (update_expression [1, 32] - [1, 35]
          argument: (expression [1, 32] - [1, 33]
            (identifier [1, 32] - [1, 33]))))
      body: (statement [1, 37] - [3, 1]
        (statement_block [1, 37] - [3, 1]
          (statement [2, 4] - [2, 18]
            (expression_statement [2, 4] - [2, 18]
              (expression [2, 4] - [2, 17]
                (assignment_expression [2, 4] - [2, 17]
                  left: (identifier [2, 4] - [2, 7])
                  right: (expression [2, 10] - [2, 17]
                    (binary_expression [2, 10] - [2, 17]
                      left: (expression [2, 10] - [2, 13]
                        (identifier [2, 10] - [2, 13]))
                      right: (expression [2, 16] - [2, 17]
                        (identifier [2, 16] - [2, 17])))))))))))
```

for循环解析

：for (init; condition; update) { body } • 可选的初始化语句，可选的循环条件，可选的更新语句， • 循环体可以是单条语句或代码块

```
    for_statement: $ => seq(
      // week4 for语句
      'for',
      '(',
      field('init', optional(choice(
        seq(
          $.expression,
          repeat(seq(',', $.expression))
        ),
        $.variable_declaration),
      )),
      ';',
      field('condition', optional($.expression)),
      ';',
      field('update', optional(
        seq(
          $.expression,
          repeat(seq(',', $.expression)))
        )
      ),
      ')',
      field('body', $.statement)
    ),
```

for循环分为四个field，依次为括号内部的三个：init, condition, update，中间分号分隔；后面循环体部分statement。

- init部分为可选的两种可能：如果不需要定义新变量，则可以是一个或多个表达式expression，例如a = 1或a = 1, b = 1或a, b = 1；如果需要定义新变量，则直接使用后面不带分号的variable_declaration即可。关于variable_declaration和新的带分号的variable_declaration_statement，我们在后面遇到的问题中进行了详细的讨论。
- condition部分较为简单，就是一个可选的expression；
- update中允许对多个变量进行更新，因此会允许在第一个expression后面用逗号隔开并跟上更多expression。当然，这部分也是可选的，可以是空的。
- body部分就是必选的一个statement，至少是一个分号。

遇到的问题

```
variable_declaration: $ => choice(
    // const声明必须初始化
    seq(
        field('kind', 'const'),
        field('name', $.identifier), // 变量名
        optional(field('type', $.type_annotation)), // 可选的类型注解
        seq('=', field('value', $.expression)), // 初始化表达式是必须的
        repeat(seq(
          ',',
          field('name', $.identifier), // 变量名
          optional(field('type', $.type_annotation)), // 可选的类型注解
```

```
            seq('=', field('value', $.expression)),
            )// 初始化表达式是必须的))
          )
      ),
      // let声明初始化表达式是可选的
      seq(
          field('kind', 'let'),
          field('name', $.identifier),
          optional(field('type', $.type_annotation)),
          optional(seq('=', field('value', $.expression))), // 初始化表达式是可选的
          repeat(seq(
            ',',
            field('name', $.identifier), // 变量名
            optional(field('type', $.type_annotation)), // 可选的类型注解
            optional(seq('=', field('value', $.expression))),
          )// 初始化表达式是必须的))
        )
      )
  ),


  variable_declaration_statement: $ => seq(
    $.variable_declaration,
    optional($._semicolon),
  ),
```

此前variable_declaration中就进行了分号的处理，这导致在for的init部分遇到了问题，init后必须跟一个分号来分隔init和condition部分，而原有的variable_declaration后面就有一个可选的分号，于是可能会与必须存在的分号发生重复从而产生错误。于是我们采用了上面的处理方式，variable_declaration_statement中才在最后加入可选分号，而for的init部分直接使用不加上分号的variable_declaration。

而variable_declaration也在上次的基础上作出了修正。我们意识到上次的写法中，不能够识别利用逗号,分隔开的连续多个变量声明，例如let a = 1, b = 2。我们现在加入了这种识别。做法也很简单，在之前的const（必须带初始化）和let（可以带初始化）后面加上repeat的前序序列即可。

解析二元运算

```
    binary_expression: $ => choice(
      ...[
        ['+', 'binary_plus'],
        ['-', 'binary_plus'],
        ['*', 'binary_times'],
        ['/', 'binary_times'],
        ['%', 'binary_times'],
        ['<', 'binary_relation'],
        ['<=', 'binary_relation'],
        ['==', 'binary_equality'],
        ['===', 'binary_equality'],
        ['!=', 'binary_equality'],
```

```
        ['!==', 'binary_equality'],
        ['>=', 'binary_relation'],
        ['>', 'binary_relation'],
      ].map(([operator, precedence, associativity]) =>
        (associativity === 'right' ? prec.right : prec.left)(precedence, seq(
          // week4 识别二元操作binary_expression
          field('left', $.expression),
          field('operator', operator),
          field('right', $.expression)
        )),
      ),
    ),
```

`binary_expression`语法规则规定了二元运算表达式，定义了十三种二元运算规则及其优先级，通过`map`函数动态生成多个子规则。
二元运算的结构由三部分构成：左操作数、运算符和右操作数。分别对应`field('left', $.expression),`、`field('operator', operator),`和`field('right', $.expression)`。