仓库介绍

本仓库为LiteJS词法分析工作方向的代码仓库，由黄毓玮创建、杨哲宇和黎可为参与维护。

仓库结构

ECMA/ -> ECMA标准文档

flex_and_bison/ -> 使用flex和bison进行词法分析的可行性研究

quickjs_test_result/ -> 对quickjs执行test262测试套件（使用默认设置和打开所有测试用例两种情况）的结果，及其失败用例的分类

test262/ -> Test262: ECMAScript Test Suite (ECMA TR/104)

tokenizer/ -> LiteJS词法分析器源代码

词法分析器

背景：

工作目标：使用无依赖类库的纯C编写一个符合ECMA最新标准的词法分析器。编写过程中适当参照quickjs的代码实现。

标准文档：ECMA-262 11 th Edition / June 2020 -> Chapter 11 ECMAScript Language: Lexical Grammar

实现情况：

- white space
- line terminator
- comment
- identifier
- keyword
- punctuator
- null literals
- boolean literals
- string literals
- numeric literals
- template literals
- regexp literals

测试情况

> gcc版本：gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0

在Linux64位（WSL 2）环境下，用./tokenizer/source.txt进行了测试并与quickjs对比输出。除此之外未进行任何其他测试

词法分析器组成介绍

词法分析器采用递归下降法设计，使用C语言通过状态机实现（见next_token函数。原理：从INITIAL状态开始，根据下一个字符的情况设置状态status，并在下轮循环进入对应状态的switch-case，如此往复直到读取到该Token的末字符；此时调用finish函数组装返回值给next_token的调用方）。各部分功能如下：

`template.h`：定义状态常量、Token类型常量、字符和字符串常量，并定义了Token数据结构和next_token函数的返回结构

`template.c`：词法分析器主程序。其中

`main函数`：读取source.txt并不断循环调用next_token子程序，输出结果

`next_token函数`：从position位置开始读取下一个Token并返回。通过状态机原理实现

`*_finish函数`：由next_token函数在解析Token完成时调用，负责设置返回值并进行相应的词义解释（如将非十进制值转换为整数值、解析正则表达式的body和flag等）。注意：部分词义解析在处理状态的同时就完成了（如字符串字面量的转义字符处理），不再需要在此处解释

`compare.py`：Python实现的对拍程序。同时打印出quickjs和LiteJS的词法分析结果。使用前需要注意修改源代码中的命令行部分。同时需要编译quickjs使其打开dump_token开关（quickjs源代码->quickjs.c->next_token(JSParseState *)->末尾处解注释）

- String 字符串特殊说明：

字符串字面量的存储形式与其他token的存储形式不太一样，因为需要实现Unicode支持，字符串使用了`char_16t`数组作为内部存储形式。使用的是token结构体中的`content_16`数组。在存储时，所有的字符均按16位存储，如果是ASCII字符则高八位均为0。

- String 字符串实现：

字符串的解析最重要的工作是对于转义字符的处理，在我们的实现中，我们编写了一个转义字符处理函数`parse_escape`，当状态机读取到`\`时，就会进入转义处理部分，除了单双引号和换行连接的情况，其他所有情况均有上述函数处理。

- String 字符串已知缺陷：
  - 用于将token输出的调试功能还无法输出除ASCII以外的字符，具体的Unicode解析还需要由虚拟机来完成。
  - 目前还不支持源码级的UTF-8输入，字符串中使用Unicode仅能通过转义字符输入。
  - 字符串变量的值单独开了一个数组，增大的内存占用，最好在之后将所有token值统一为`char_16t`存储。
  - 没有实现JS严格模式

其他说明

作者在编写词法编辑器对拍程序的过程中，曾经试图编制自动化对拍程序来比对Token结果，但由于quickjs的dump_token输出与预期不符合而未能实现，具体原因如下：

- quickjs切分 `var character = ["0", "1", "2"];` 之类的语句，输出为：

```
ident: 'var'
ident: 'character'
token: '='
token: '['
string: '0'
token: ','
string: '1'
token: ','
string: '2'
token: ']'
token: ';'
token: '['
string: '0'
token: ','
string: '1'
token: ','
string: '2'
token: ']'
token: ';'
```

  可以看到数组与分号（异常地）打印了两次。

- quickjs存储标点符号时采用的 `token→val` 为 `int` 类型，例如 `≤` 为 `-101` 、 `++` 为 `-106` 。打印时quickjs作者采用 `%c` 格式输出，必定是一个乱码字符。而且， `token` 与 `int` 的对应关系我们没有找到
- ECMA中的 `big number` 类型，quickjs的 `dump_token` 打印为 `nan` 。这样就不具备验证我们程序正确性的效果了

下附完整的LiteJS与quickjs的切词对比情况（针对./tokenizer/source.txt）：

```
====LITEJS====              ├====QUICKJS====
[  5] CLASS: 'class'         |ident: 'class'
[ 10] IDENT: 'test'          |ident: 'test'
[ 11]LBRACE: '{'             |token: '{'
[ 33] IDENT: 'constructor'   |ident: 'constructor'
[ 34]LPAREN: '('             |token: '('
[ 35]RPAREN: ')'             |token: ')'
[ 36]LBRACE: '{'             |token: '{'
[ 37]RBRACE: '}'             |token: '('
[ 39]RBRACE: '}'             |token: ')'
[119]   VAR: 'var'           |token: '{'
[127] IDENT: 'unicode'       |token: '}'
[129]    EQ: '='             |token: '}'
[131]LBRACK: '['             |ident: 'var'
[139]STRING: '0'             |ident: 'unicode'
```

```
[140] COMMA: ','              |token: '='
[149]STRING: '1'              |token: '['
[150] COMMA: ','              |string: '0'
[159]STRING: '2'              |token: ','
[160]RBRACK: ']'              |string: '1'
[161]  SEMI: ';'              |token: ','
[165]   VAR: 'var'            |string: '2'
[175] IDENT: 'character'      |token: ']'
[177]    EQ: '='              |token: ';'
[179]LBRACK: '['              |token: '['
[182]STRING: '0'              |string: '0'
[183] COMMA: ','              |token: ','
[187]STRING: '1'              |string: '1'
[188] COMMA: ','              |token: ','
[192]STRING: '2'              |string: '2'
[193]RBRACK: ']'              |token: ']'
[194]  SEMI: ';'              |token: ';'
[198]   FOR: 'for'            |ident: 'var'
[200]LPAREN: '('              |ident: 'character'
[203]   VAR: 'var'            |token: '='
[209] IDENT: 'index'          |token: '['
[211]    EQ: '='              |string: '0'
————NV(long long): 0          |token: ','
[214]NUMBER: '0'              |string: '1'
[220] IDENT: 'index'          |token: ','
[223]  LTEQ: '≤'              |string: '2'
————NV(long long): 9          |token: ']'
[225]NUMBER: '9'              |token: ';'
[226]  SEMI: ';'              |token: '['
[232] IDENT: 'index'          |string: '0'
[234] PLUS2: '++'             |token: ','
[235]RPAREN: ')'              |string: '1'
[237]LBRACE: '{'              |token: ','
[242]    IF: 'if'             |string: '2'
[244]LPAREN: '('              |token: ']'
[251] IDENT: 'unicode'        |token: ';'
[252]LBRACK: '['              |ident: 'for'
[257] IDENT: 'index'          |token: '('
[258]RBRACK: ']'              |ident: 'var'
[262]NOTEQ2: '≠='             |ident: 'index'
[272] IDENT: 'character'      |token: '='
[273]LBRACK: '['              |number: 0
[278] IDENT: 'index'          |token: ';'
[279]RBRACK: ']'              |ident: 'index'
[280]RPAREN: ')'              |token: (-101)
[282]LBRACE: '{'              |number: 9
[293] IDENT: '$ERROR'         |token: ';'
[294]LPAREN: '('              |ident: 'index'
[297]STRING: '#'              |token: (-106)
[299]  PLUS: '+'              |token: ')'
[309] IDENT: 'character'      |token: '{'
[310]LBRACK: '['              |token: '('
[315] IDENT: 'index'          |ident: 'var'
[316]RBRACK: ']'              |ident: 'index'
[318]  PLUS: '+'              |token: '='
[322]STRING: ' '              |number: 0
```

```
[323]RPAREN: ')'              |token: ';'
[324]  SEMI: ';'              |ident: 'index'
[328]RBRACE: '}'              |token: (-101)
[330]RBRACE: '}'              |number: 9
[335]   VAR: 'var'            |token: ';'
[343] IDENT: 'big_num'        |ident: 'index'
[345]    EQ: '='              |token: (-106)
————NV(big_int): 9999999999999n|token: ')'
[360]NUMBER: '9999999999999n' |token: '{'
[365]   VAR: 'var'            |ident: 'if'
[375] IDENT: 'float_num'      |token: '('
[377]    EQ: '='              |ident: 'unicode'
————NV(double): 0.0000110000 |token: '['
[384]NUMBER: '1.1e-5'         |ident: 'index'
[386]  PLUS: '+'              |token: ']'
————NV(double): 1.0000000100 |token: (-95)
[397]NUMBER: '1.00000001'     |ident: 'character'
[401] IDENT: 'let'            |token: '['
[410] IDENT: 'more_num'       |ident: 'index'
[412]    EQ: '='              |token: ']'
————NV(double): 1.0000000000 |token: ')'
[415]NUMBER: '1.'             |token: '{'
[417]  PLUS: '+'              |ident: '$ERROR'
————NV(double): 0.0200000000 |token: '('
[421]NUMBER: '.02'            |string: '#'
[423]  PLUS: '+'              |token: '+'
————NV(double): 3000000.0000000000|ident: 'character'
[428]NUMBER: '.03e8'          |token: '['
[430]  PLUS: '+'              |ident: 'index'
————NV(double): 0.0400000000 |token: ']'
[435]NUMBER: '4.e-2'          |token: '+'
[439] IDENT: 'let'            |string: ' '
[455] IDENT: 'non_decimal_num'|token: ')'
[457]    EQ: '='              |token: ';'
————NV(long long): 42        |token: '}'
[468]NUMBER: '0b00101010'     |token: '}'
[469]  PLUS: '+'              |ident: 'var'
————NV(long long): 511       |ident: 'big_num'
[474]NUMBER: '0O777'          |token: '='
[475] MINUS: '-'              |number: nan
————NV(long long): 43694     |ident: 'var'
[481]NUMBER: '0xAAAE'         |ident: 'float_num'
[482]  PLUS: '+'              |token: '='
————NV(long long): 100       |number: 1.1e-05
[485]NUMBER: '100'            |token: '+'
[486] MINUS: '-'              |number: 1.00000001
————NV(long long): 1         |ident: 'let'
[489]NUMBER: '0X1'            |ident: 'more_num'
[490]  PLUS: '+'              |ident: 'let'
————NV(long long): 0         |ident: 'more_num'
[493]NUMBER: '0B0'            |token: '='
[494] MINUS: '-'              |number: 1
————NV(long long): 0         |token: '+'
[497]NUMBER: '0o0'            |number: 0.02
[501] IDENT: 'let'            |token: '+'
[513] IDENT: 'decimal_num'    |number: 3000000
```

```
[515]    EQ: '='              |token: '+'
————NV(long long): 1234567890|number: 0.04
[526]NUMBER: '1234567890'     |ident: 'let'
[531]   VAR: 'var'            |ident: 'non_decimal_num'
[542] IDENT: 'regexp_str'     |ident: 'let'
[544]    EQ: '='              |ident: 'non_decimal_num'
————REGEXP(body=w+/*, flag=g)|token: '='
[554]REGEXP: 'w+/*/g'         |number: 42
[559] IDENT: 'let'            |token: '+'
[569] IDENT: 'empty_str'      |number: 511
[571]    EQ: '='              |token: '-'
[574]STRING: ''               |number: 43694
[578] IDENT: 'let'            |token: '+'
[587] IDENT: 'str_test'       |number: 100
[589]    EQ: '='              |token: '-'
[607]STRING: '\ \       123\'       |number: 1
[611] IDENT: 'let'            |token: '+'
[621] IDENT: 'str_test2'      |number: 0
[623]    EQ: '='              |token: '-'
[631]STRING: 'asdfg'          |number: 0
                              |ident: 'let'
                              |ident: 'decimal_num'
                              |ident: 'let'
                              |ident: 'decimal_num'
                              |token: '='
                              |number: 1234567890
                              |ident: 'var'
                              |ident: 'regexp_str'
                              |token: '='
                              |token: '/'
                              |ident: 'let'
                              |ident: 'empty_str'
                              |ident: 'let'
                              |ident: 'empty_str'
                              |token: '='
                              |string: ''
                              |ident: 'let'
                              |ident: 'str_test'
                              |ident: 'let'
                              |ident: 'str_test'
                              |token: '='
                              |string: '\       \       123\'
                              |ident: 'let'
                              |ident: 'str_test2'
                              |ident: 'let'
                              |ident: 'str_test2'
                              |token: '='
                              |string: 'asdfg'
                              |eof
                              |
```