gbroloyalo feat: compiscript grammar 9b0fe4f · 3 months ago

175 lines (136 loc) · 3.89 KB

Code Blame Raw

```
grammar Compiscript;

// ------------------
// Parser Rules
// ------------------

program: statement* EOF;

statement
  : variableDeclaration
  | constantDeclaration
  | assignment
  | functionDeclaration
  | classDeclaration
  | expressionStatement
  | printStatement
  | block
  | ifStatement
  | whileStatement
  | doWhileStatement
  | forStatement
  | foreachStatement
  | tryCatchStatement
  | switchStatement
  | breakStatement
  | continueStatement
  | returnStatement
  ;

block: '{' statement* '}';

variableDeclaration
  : ('let' | 'var') Identifier typeAnnotation? initializer? ';'
  ;

constantDeclaration
  : 'const' Identifier typeAnnotation? '=' expression ';'
  ;

typeAnnotation: ':' type;
initializer: '=' expression;

assignment
  : Identifier '=' expression ';'
```

```
45      | expression '.' Identifier '=' expression ';' // property assignment
46      ;
47
48  expressionStatement: expression ';';
49  printStatement: 'print' '(' expression ')' ';';
50
51  ifStatement: 'if' '(' expression ')' block ('else' block)?;
52  whileStatement: 'while' '(' expression ')' block;
53  doWhileStatement: 'do' block 'while' '(' expression ')' ';';
54  forStatement: 'for' '(' (variableDeclaration | assignment | ';') expression? ';' expression? ')' block;
55  foreachStatement: 'foreach' '(' Identifier 'in' expression ')' block;
56  breakStatement: 'break' ';';
57  continueStatement: 'continue' ';';
58  returnStatement: 'return' expression? ';';
59
60  tryCatchStatement: 'try' block 'catch' '(' Identifier ')' block;
61
62  switchStatement: 'switch' '(' expression ')' '{' switchCase* defaultCase? '}';
63  switchCase: 'case' expression ':' statement*;
64  defaultCase: 'default' ':' statement*;
65
66  functionDeclaration: 'function' Identifier '(' parameters? ')' (':' type)? block;
67  parameters: parameter (',' parameter)*;
68  parameter: Identifier (':' type)?;
69
70  classDeclaration: 'class' Identifier (':' Identifier)? '{' classMember* '}';
71  classMember: functionDeclaration | variableDeclaration | constantDeclaration;
72
73  // ------------------
74  // Expression Rules — Operator Precedence
75  // ------------------
76
77  expression: assignmentExpr;
78
79  assignmentExpr
80    : lhs=leftHandSide '=' assignmentExpr            # AssignExpr
81    | lhs=leftHandSide '.' Identifier '=' assignmentExpr # PropertyAssignExpr
82    | conditionalExpr                                # ExprNoAssign
83    ;
84
85  conditionalExpr
86    : logicalOrExpr ('?' expression ':' expression)? # TernaryExpr
87    ;
88
89  logicalOrExpr
90    : logicalAndExpr ( '||' logicalAndExpr )*
91    ;
92
93  logicalAndExpr
94    : equalityExpr ( '&&' equalityExpr )*
95    ;
96
97  equalityExpr
98    : relationalExpr ( ('==' | '!=') relationalExpr )*
99    ;
100
101 relationalExpr
102   : additiveExpr ( ('<' | '<=' | '>' | '>=') additiveExpr )*
103   ;
104
```

```
105     additiveExpr
106       : multiplicativeExpr ( ('+' | '-') multiplicativeExpr )*
107       ;
108
109     multiplicativeExpr
110       : unaryExpr ( ('*' | '/' | '%') unaryExpr )*
111       ;
112
113     unaryExpr
114       : ('-' | '!') unaryExpr
115       | primaryExpr
116       ;
117
118     primaryExpr
119       : literalExpr
120       | leftHandSide
121       | '(' expression ')'
122       ;
123
124     literalExpr
125       : Literal
126       | arrayLiteral
127       | 'null'
128       | 'true'
129       | 'false'
130       ;
131
132     leftHandSide
133       : primaryAtom (suffixOp)*
134       ;
135
136     primaryAtom
137       : Identifier                                  # IdentifierExpr
138       | 'new' Identifier '(' arguments? ')'         # NewExpr
139       | 'this'                                      # ThisExpr
140       ;
141
142     suffixOp
143       : '(' arguments? ')'                         # CallExpr
144       | '[' expression ']'                         # IndexExpr
145       | '.' Identifier                             # PropertyAccessExpr
146       ;
147
148     arguments: expression (',' expression)*;
149
150     arrayLiteral: '[' (expression (',' expression)*)? ']';
151
152     // ------------------
153     // Types
154     // ------------------
155
156     type: baseType ('[' ']')*;
157     baseType: 'boolean' | 'integer' | 'string' | Identifier;
158
159     // ------------------
160     // Lexer Rules
161     // ------------------
162
163     Literal
164       : IntegerLiteral
```

```
      | StringLiteral
      ;

    IntegerLiteral: [0-9]+;
    StringLiteral: '"' (~["\r\n])* '"';

    Identifier: [a-zA-Z_][a-zA-Z0-9_]*;

    WS: [ \t\r\n]+ -> skip;
    COMMENT: '//' ~[\r\n]* -> skip;
    MULTILINE_COMMENT: '/*' .*? '*/' -> skip;
```