Semestrální práce X36PJP

Zvýrazňování a kontrola syntaxe jazyka PHP5

Martin Řezník - martin@yosarin.net, reznim3@fel.cvut.cz

Obsah

Rozbor

Program (aplikace napsaná v PHP) na vstupu obdrží skript napsaný v PHP5, provede jeho lexikální rozbor a syntaktickou analýzu. Výstupem je html se zvýrazněnou syntaxí jazyka a seznam všech definovaných funkcí, metod a tříd, včetně odkazů na ně.

Gramatika

Gramatiku jsem se rozhodl vytvořit co nejkompletnější – netroufám si tvrdit zda pokryla celý jazyk, ale je poměrně obsáhlá. Je možné, že gramatika se bude průběžně ještě měnit (pokud narazím na nějakou chybu), aktuální verze bude dostupná na stránce http://pjp.yosarin.net/gramatika.txt. O rozsáhlosti gramatiky svědčí už to, že má celkem přes 460 pravidel. Pokrývá jak obyčejně psané (procedurální) kódy, tak objektový model (měl by odpovídat PHP5 i PHP4).

Symboly psané v gramatice velkými písmeny (a zeleně) jsou terminálními symboly jazyka, symboly začínající dvěmi podtržítky (v červené barvě) jsou výstupními symboly. Zbytek tvoří neterminály.

Gramatika je tolik rozsáhlá především proto, že v PHP se liší bloky uvnitř tříd a mimo ně pouze několika maličkostmi – například že uvnitř třídy není možné definovat další třídu.

```
1.  file -> HTML phpCode

2.  phpCode -> PHP_OPEN_TAG nextBlock
3.  phpCode -> EOI

4.  nextBlock -> codeBlock nextBlock
5.  nextBlock -> phpEnd

6.  phpEnd -> PHP_CLOSE_TAG HTML phpCode
7.  phpEnd -> EOI
8.  phpEnd ->

9.  codeBlock -> OPEN_BLOCK nextCodeInsideBlock CLOSE_BLOCK
10. codeBlock -> expression

11. nextCodeInsideBlock -> codeInsideBlock nextCodeInsideBlock
12. nextCodeInsideBlock ->

13. codeInsideBlock -> OPEN_BLOCK nextCodeInsideBlock CLOSE_BLOCK
14. codeInsideBlock -> expression
15. codeInsideBlock -> PHP_CLOSE_TAG HTML PHP_OPEN_TAG

16. nextElseCodeInsideBlock -> ifExpression nextElseCodeInsideBlock
17. nextElseCodeInsideBlock -> elseExpression elseCodeBlockEnd
18. nextElseCodeInsideBlock -> PHP_CLOSE_TAG HTML PHP_OPEN_TAG elseCodeBlockEnd

19. elseCodeBlockEnd -> nextElseCodeInsideBlock
20. elseCodeBlockEnd ->

21. unsafeExpression -> unfinishedExpression
22. unsafeExpression -> lastExpression

23. expression -> elseExpression
24. expression -> ifExpression

25. ifExpression -> KW_IF OPEN_BRACKET fillLine CLOSE_BRACKET ifBlock

26. elseExpression -> unsafeExpression afterUnsafe
27. elseExpression -> safeExpression

28. afterUnsafe -> expressionEnd

29. safeExpression -> functionDeclaration
30. safeExpression -> classDeclaration
31. safeExpression -> interfaceDeclaration
32. safeExpression -> SEMICOLON
33. safeExpression -> KW_TRY OPEN_BLOCK nextCodeInsideBlock CLOSE_BLOCK KW_CATCH OPEN_BRACKET IDENT VAR CLOSE_BRACKET OPEN_BLOCK
    nextCodeInsideBlock CLOSE_BLOCK nextCatch
34. safeExpression -> KW_FOR OPEN_BRACKET forList CLOSE_BRACKET forBlock
35. safeExpression -> KW_FOREACH OPEN_BRACKET foreachList CLOSE_BRACKET foreachBlock
36. safeExpression -> KW_SWITCH OPEN_BRACKET fillLine CLOSE_BRACKET switchBlock
37. safeExpression -> KW_WHILE OPEN_BRACKET fillLine CLOSE_BRACKET whileBlock
38. safeExpression -> KW_DECLARE OPEN_BRACKET IDENT ASSIGN constValue nextDeclare CLOSE_BRACKET declareBlock

39. unfinishedExpression -> KW_DO doBlock KW_WHILE OPEN_BRACKET fillLine CLOSE_BRACKET
40. unfinishedExpression -> KW_ECHO fillLine nextEcho
41. unfinishedExpression -> staticAssign

42. nextEcho -> COMMA fillLine nextEcho
43. nextEcho ->

44. nextCatch -> KW_CATCH OPEN_BRACKET IDENT VAR CLOSE_BRACKET OPEN_BLOCK nextCodeInsideBlock CLOSE_BLOCK nextCatch
45. nextCatch ->

46. nextDeclare -> COMMA IDENT ASSIGN constValue nextDeclare
```

```
47. nextDeclare ->

48. declareBlock -> expression
49. declareBlock -> OPEN_BLOCK nextCodeInsideBlock CLOSE_BLOCK
50. declareBlock -> COLON nextCodeInsideBlock KW_ENDDECLARE expressionEnd

51. forList -> forParams SEMICOLON line SEMICOLON forParams

52. forParams -> fillLine lineNext
53. forParams ->

54. lineNext -> COMMA fillLine lineNext
55. lineNext ->

56. foreachList -> fillLine KW_AS variable assignKeys

57. forBlock -> expression
58. forBlock -> OPEN_BLOCK nextCodeInsideBlock CLOSE_BLOCK
59. forBlock -> COLON nextCodeInsideBlock KW_ENDFOR

60. foreachBlock -> expression
61. foreachBlock -> OPEN_BLOCK nextCodeInsideBlock CLOSE_BLOCK
62. foreachBlock -> COLON nextCodeInsideBlock KW_ENDFOREACH

63. ifBlock -> elseExpression nextIf
64. ifBlock -> ifExpression
65. ifBlock -> OPEN_BLOCK nextCodeInsideBlock CLOSE_BLOCK nextIf
66. ifBlock -> ifColonBlock
67. ifBlock -> PHP_CLOSE_TAG HTML PHP_OPEN_TAG

68. ifColonBlock -> COLON nextElseCodeInsideBlock nextColonIf

69. elseBlock -> OPEN_BLOCK nextCodeInsideBlock CLOSE_BLOCK
70. elseBlock -> elseExpression

71. nextColonIf -> KW_ELSE COLON nextCodeInsideBlock KW_ENDIF expressionEnd
72. nextColonIf -> KW_ELSEIF OPEN_BRACKET fillLine CLOSE_BRACKET ifColonBlock
73. nextColonIf -> KW_ENDIF expressionEnd

74. nextIf -> KW_ELSE elseIf
75. nextIf -> KW_ELSEIF OPEN_BRACKET fillLine CLOSE_BRACKET ifBlock
76. nextIf ->

77. elseIf -> KW_IF OPEN_BRACKET fillLine CLOSE_BRACKET ifBlock
78. elseIf -> elseBlock

79. switchBlock -> OPEN_BLOCK case CLOSE_BLOCK
80. switchBlock -> COLON case KW_ENDSWITCH

81. case -> KW_CASE fillLine COLON nextCodeInsideBlock case
82. case -> KW_DEFAULT COLON nextCodeInsideBlock case
83. case ->

84. whileBlock -> expression
85. whileBlock -> OPEN_BLOCK nextCodeInsideBlock CLOSE_BLOCK
86. whileBlock -> COLON nextCodeInsideBlock KW_ENDWHILE

87. doBlock -> expression
88. doBlock -> OPEN_BLOCK nextCodeInsideBlock CLOSE_BLOCK

89. lastExpression -> fillLine
90. lastExpression -> KW_BREAK line
91. lastExpression -> KW_CONTINUE line
92. lastExpression -> KW_RETURN line
93. lastExpression -> KW_GLOBAL VAR nextGlobal
94. lastExpression -> KW_THROW fillLine
95. lastExpression -> includeFile fillLine

96. nextListParam -> COMMA maybeVariable nextListParam
97. nextListParam ->

98. maybeVariable -> variable
99. maybeVariable ->

100.dieParams -> OPEN_BRACKET line CLOSE_BRACKET
101.dieParams ->

102.includeFile -> KW_INCLUDE
103.includeFile -> KW_INCLUDE_ONCE
104.includeFile -> KW_REQUIRE
105.includeFile -> KW_REQUIRE_ONCE

106.nextGlobal -> COMMA VAR nextGlobal
107.nextGlobal ->

108.staticAssign -> KW_STATIC __STATIC_VAR_DECLARATION VAR ASSIGN __STATIC_VAR_VALUE classDeclarationValue nextStatic

109.nextStatic -> COMMA __STATIC_VAR_DECLARATION VAR ASSIGN __STATIC_VAR_VALUE classDeclarationValue nextStatic
110.nextStatic ->

111.fillLine -> logic chainNext

112.chainNext -> chainOp logic chainNext
113.chainNext -> QUOTATION logic chainNext COLON logic chainNext
114.chainNext ->

115.logic -> compare logicNext
116.logicNext -> logicOp compare logicNext
117.logicNext ->

118.compare -> bitwise compareNext
119.compareNext -> compareOp bitwise compareNext
```

```
120.compareNext ->

121.bitwise -> add bitwiseNext
122.bitwiseNext -> bitwiseOp add bitwiseNext
123.bitwiseNext ->

124.add -> mult addNext
125.addNext -> addOp mult addNext
126.addNext ->

127.mult -> last multNext
128.multNext -> multOp last multNext
129.multNext ->

130.last -> unaryOp last
131.last -> value
132.last -> KW_EXIT dieParams
133.last -> KW_DIE dieParams
134.last -> KW_LIST OPEN_BRACKET line nextListParam CLOSE_BRACKET ASSIGN last
135.last -> KW_EMPTY OPEN_BRACKET variable CLOSE_BRACKET
136.last -> __SET_TYPE TYPE varOrIdent
137.last -> variable variableNext
138.last -> IDENT identFirstNext
139.last -> OPEN_BRACKET fillLine CLOSE_BRACKET

140.varOrIdent -> variable variableNext
141.varOrIdent -> IDENT identFirstNext

142.identFirstNext -> __P_FUNCTION_CALL functionCall maybeObject
143.identFirstNext -> __P_CLASS_CALL CLASS_PROPERTY classPropertyNext
144.identFirstNext -> __P_CONST_CALL KW_INSTANCEOF instanceofNext
145.identFirstNext -> __P_CONST_CALL

146.identFirstNextCutted -> __P_CLASS_CALL CLASS_PROPERTY classPropertyNextCutted
147.identFirstNextCutted -> __P_CONST_CALL KW_INSTANCEOF instanceofNext
148.identFirstNextCutted -> __P_CONST_CALL

149.maybeObject -> INSTANCE_PROPERTY instancePropertyNext
150.maybeObject ->

151.maybeObjectCutted -> INSTANCE_PROPERTY instancePropertyNextCutted
152.maybeObjectCutted ->

153.instancePropertyNextCutted -> variable variableNextCutted
154.instancePropertyNextCutted -> IDENT maybeObjectCutted

155.instancePropertyNext -> variable variableNext
156.instancePropertyNext -> IDENT maybeObject

157.staticNext -> functionCall maybeObject
158.staticNext -> KW_INSTANCEOF instanceofNext
159.staticNext ->

160.classPropertyNext -> __CLASS_PROPERTY_CALL IDENT staticNext
161.classPropertyNext -> __CLASS_PROPERTY_CALL variable variableNext

162.classPropertyNextCutted -> __CLASS_PROPERTY_CALL variable variableNextCutted

163.variableNext -> __P_FUNCTION_POINTER_OR_METHOD functionCall maybeObject
164.variableNext -> __VARIABLE_ASSIGN assignOp assignNext
165.variableNext -> incrementOp
166.variableNext -> variableNextCutted

167.variableNextCutted -> KW_INSTANCEOF instanceofNext
168.variableNextCutted ->

169.instanceofNext -> variable variableNextCutted
170.instanceofNext -> IDENT identFirstNextCutted

171.assignNext -> AMP variable
172.assignNext -> last

173.line -> fillLine
174.line ->

175.logicOp -> AND
176.logicOp -> OR
177.logicOp -> XOR
178.logicOp -> KW_T_LOGICAL_AND
179.logicOp -> KW_T_LOGICAL_OR
180.logicOp -> KW_T_LOGICAL_XOR

181.compareOp -> MORE
182.compareOp -> LESS
183.compareOp -> MORE_OR_EQUAL
184.compareOp -> LESS_OR_EQUAL
185.compareOp -> EQUAL
186.compareOp -> SAFE_EQUAL
187.compareOp -> NOT_EQUAL
188.compareOp -> SAFE_NOT_EQUAL

189.multOp -> DIVIDE
190.multOp -> MULTIPLE
191.multOp -> MODULO

192.addOp -> ADD
193.addOp -> SUBS

194.bitwiseOp -> PIPE
195.bitwiseOp -> AMP
196.bitwiseOp -> SHIFT_LEFT
197.bitwiseOp -> SHIFT_RIGHT
```

```
198.incrementOp -> INCREMENT
199.incrementOp -> DECREMENT

200.unaryOp -> COMPLEMENT
201.unaryOp -> NON
202.unaryOp -> MUTE
203.unaryOp -> KW_CLONE
204.unaryOp -> constUnaryOp

205.constUnaryOp -> ADD
206.constUnaryOp -> SUBS


207.reference -> AMP __VAR_PASSED_BY_REFERENCE
208.reference ->

209.chainOp -> DOT

210.assignOp -> ASSIGN
211.assignOp -> ASSIGN_INCREMENT
212.assignOp -> ASSIGN_DECREMENT
213.assignOp -> ASSIGN_MULTIPLY
214.assignOp -> ASSIGN_DIVIDE
215.assignOp -> ASSIGN_BINARY_OR
216.assignOp -> ASSIGN_BINARY_XOR
217.assignOp -> ASSIGN_BINARY_AND
218.assignOp -> ASSIGN_DOT
219.assignOp -> ASSIGN_MODULO
220.assignOp -> ASSIGN_BINARY_LEFT
221.assignOp -> ASSIGN_BINARY_RIGHT

222.setType -> TYPE
223.setType ->

224.type -> typeInBracket
225.type -> __TYPE_ARRAY KW_ARRAY

226.typeInBracket -> __TYPE_INT KW_INT
227.typeInBracket -> __TYPE_FLOAT KW_FLOAT
228.typeInBracket -> __TYPE_STRING KW_STRING
229.typeInBracket -> __TYPE_BOOL KW_BOOL
230.typeInBracket -> __TYPE_OBJECT KW_OBJECT

231.constValue -> __TYPE_STRING STRING
232.constValue -> __TYPE_FLOAT FLOAT
233.constValue -> __TYPE_HEXA HEXA
234.constValue -> __TYPE_INT INT
235.constValue -> __TYPE_BOOL KW_TRUE
236.constValue -> __TYPE_BOOL KW_FALSE
237.constValue -> __TYPE_NULL KW_NULL

238.array -> __TYPE_ARRAY KW_ARRAY OPEN_BRACKET arrayParams CLOSE_BRACKET

239.arrayParams -> fillLine assignKeys nextArrayParam
240.arrayParams ->

241.assignKeys -> AS fillLine
242.assignKeys ->

243.nextArrayParam -> COMMA arrayParams
244.nextArrayParam ->

245.classDeclarationValue -> nextConstUnaryOp constValue
246.classDeclarationValue -> array

247.nextConstUnaryOp -> constUnaryOp nextConstUnaryOp
248.nextConstUnaryOp ->

249.value -> constValue
250.value -> array
251.value -> KW_NEW __NEW_OBJECT_CREATE IDENT maybeFunction
252.value -> incrementOp variable

253.maybeFunction -> functionCall
254.maybeFunction ->

255.functionCall -> OPEN_BRACKET paramList CLOSE_BRACKET

256.paramList -> __L_FUNCTION_CALL_PARAM fillLine __P_FUNCTION_CALL_PARAM nextParam
257.paramList ->

258.nextParam -> COMMA paramList
259.nextParam ->

260.access -> KW_PRIVATE
261.access -> KW_PUBLIC
262.access -> KW_PROTECTED

263.maybeAccess -> access
264.maybeAccess ->

265.static -> KW_STATIC

266.maybeStatic -> static
267.maybeStatic ->

268.final -> KW_FINAL
269.final -> KW_ABSTRACT

270.maybeFinal -> final
271.maybeFinal ->
```

```
272.functionDeclaration -> KW_FUNCTION __FUNCTION_DECLARE IDENT OPEN_BRACKET paramDefinition CLOSE_BRACKET OPEN_BLOCK
    nextCodeInsideBlock CLOSE_BLOCK

273.paramDefinition -> ident reference __FUNCTION_PARAM VAR paramDeclare nextParamDefinition
274.paramDefinition ->

275.ident -> __FUNCTION_PARAM_TYPE IDENT
276.ident ->

277.paramDeclare -> ASSIGN __L_PARAM_DEFAULT_VALUE classDeclarationValue __P_PARAM_DEFAULT_VALUE
278.paramDeclare ->

279.nextParamDefinition -> COMMA ident reference __FUNCTION_PARAM VAR paramDeclare nextParamDefinition
280.nextParamDefinition ->

281.classDeclaration -> maybeFinal KW_CLASS __CLASS_DECLARE IDENT extend implement OPEN_BLOCK classBlock CLOSE_BLOCK

282.extend -> KW_EXTEND __CLASS_EXTENDS IDENT
283.extend ->

284.implement -> KW_IMPLEMENT __CLASS_IMPLEMENTS IDENT nextImplement
285.implement ->

286.nextImplement -> COMMA __CLASS_IMPLEMENTS IDENT nextImplement
287.nextImplement ->

288.classBlock -> classBodyExpression classBlock
289.classBlock ->

290.classFunctionDefine -> KW_FUNCTION __METHOD_DEFINE IDENT OPEN_BRACKET paramDefinition CLOSE_BRACKET OPEN_BLOCK
    classNextFunctionBlock CLOSE_BLOCK
291.classFunctionDeclare -> KW_FUNCTION __METHOD_DECLARE IDENT OPEN_BRACKET paramDefinition CLOSE_BRACKET expressionEnd

292.classNextFunctionBlock -> classFunctionBlock classNextFunctionBlock
293.classNextFunctionBlock ->

294.classNextElseFunctionBlock -> classIfExpression classNextElseFunctionBlock
295.classNextElseFunctionBlock -> classElseExpression classElseCodeBlockEnd

296.classElseCodeBlockEnd -> classNextElseFunctionBlock
297.classElseCodeBlockEnd ->

298.classFunctionBlock -> OPEN_BLOCK classNextFunctionBlock CLOSE_BLOCK
299.classFunctionBlock -> KW_FUNCTION __IN_CLASS_FUNCTION_DEFINE IDENT OPEN_BRACKET paramDefinition CLOSE_BRACKET OPEN_BLOCK
    classNextFunctionBlock CLOSE_BLOCK
300.classFunctionBlock -> staticAssign
301.classFunctionBlock -> classExpression

302.classExpression -> classElseExpression
303.classExpression -> classIfExpression

304.classIfExpression -> KW_IF OPEN_BRACKET fillLine CLOSE_BRACKET classIfBlock

305.classElseExpression -> expressionEnd
306.classElseExpression -> lastExpression expressionEnd
307.classElseExpression -> KW_TRY OPEN_BLOCK classNextFunctionBlock CLOSE_BLOCK KW_CATCH OPEN_BRACKET IDENT VAR CLOSE_BRACKET
    OPEN_BLOCK classNextFunctionBlock CLOSE_BLOCK classNextCatch
308.classElseExpression -> KW_FOR OPEN_BRACKET forList CLOSE_BRACKET classForBlock
309.classElseExpression -> KW_FOREACH OPEN_BRACKET foreachList CLOSE_BRACKET classForeachBlock
310.classElseExpression -> KW_SWITCH OPEN_BRACKET fillLine CLOSE_BRACKET classCaseBlock
311.classElseExpression -> KW_WHILE OPEN_BRACKET fillLine CLOSE_BRACKET classWhileBlock
312.classElseExpression -> KW_DO classDoBlock KW_WHILE OPEN_BRACKET fillLine CLOSE_BRACKET expressionEnd
313.classElseExpression -> KW_ECHO fillLine nextEcho expressionEnd
314.classElseExpression -> KW_DECLARE OPEN_BRACKET IDENT ASSIGN constValue nextDeclare CLOSE_BRACKET classDeclareBlock

315.classBodyExpression -> KW_CONST __CLASS_CONST_DECLARE IDENT ASSIGN __L_CLASS_CONST_VALUE constValue __P_CLASS_CONST_VALUE
    nextConst
316.classBodyExpression -> static classStaticFirst
317.classBodyExpression -> access classAccessFirst
318.classBodyExpression -> KW_FINAL classFinalFirst
319.classBodyExpression -> KW_ABSTRACT classAbstractFirst
320.classBodyExpression -> KW_VAR classVarsDeclare
321.classBodyExpression -> classFunctionDefine

322.classStaticFirst -> access finalLast
323.classStaticFirst -> KW_FINAL maybeAccess classFunctionDefine
324.classStaticFirst -> KW_ABSTRACT maybeAccess classFunctionDeclare
325.classStaticFirst -> classDeclare

326.classAccessFirst -> static finalLast
327.classAccessFirst -> KW_FINAL maybeAccess classFunctionDefine
328.classAccessFirst -> KW_ABSTRACT maybeAccess classFunctionDeclare
329.classAccessFirst -> classDeclare

330.classFinalFirst -> static maybeAccess classFunctionDefine
331.classFinalFirst -> access maybeStatic classFunctionDefine
332.classFinalFirst -> classFunctionDefine

333.classAbstractFirst -> static maybeAccess classFunctionDeclare
334.classAbstractFirst -> access maybeStatic classFunctionDeclare
335.classAbstractFirst -> classFunctionDeclare

336.finalLast -> KW_FINAL classFunctionDefine
337.finalLast -> KW_ABSTRACT classFunctionDeclare
338.finalLast -> classDeclare

339.classDeclare -> classFunctionDefine
340.classDeclare -> classVarsDeclare

341.classVarsDeclare -> __CLASS_PROPERTY_DECLARE VAR classAsign nextClassAsign SEMICOLON
```

```
342.classNextCatch -> KW_CATCH OPEN_BRACKET IDENT VAR CLOSE_BRACKET OPEN_BLOCK classNextFunctionBlock CLOSE_BLOCK classNextCatch
343.classNextCatch ->

344.classDeclareBlock -> classExpression
345.classDeclareBlock -> OPEN_BLOCK classNextFunctionBlock CLOSE_BLOCK
346.classDeclareBlock -> COLON classNextFunctionBlock KW_ENDDECLARE

347.classForBlock -> classExpression
348.classForBlock -> OPEN_BLOCK classNextFunctionBlock CLOSE_BLOCK
349.classForBlock -> COLON classNextFunctionBlock KW_ENDFOR expressionEnd

350.classForeachBlock -> classExpression
351.classForeachBlock -> OPEN_BLOCK classNextFunctionBlock CLOSE_BLOCK
352.classForeachBlock -> COLON classNextFunctionBlock KW_ENDFOREACH expressionEnd

353.classIfBlock -> classElseExpression classNextIf
354.classIfBlock -> classIfExpression
355.classIfBlock -> OPEN_BLOCK classNextFunctionBlock CLOSE_BLOCK classNextIf
356.classIfBlock -> classIfColonBlock

357.classIfColonBlock -> COLON classNextElseFunctionBlock classNextColonIf

358.classElseBlock -> OPEN_BLOCK classNextFunctionBlock CLOSE_BLOCK
359.classElseBlock -> classElseExpression

360.classNextColonIf -> KW_ELSE COLON classNextFunctionBlock KW_ENDIF expressionEnd
361.classNextColonIf -> KW_ELSEIF OPEN_BRACKET fillLine CLOSE_BRACKET classIfColonBlock
362.classNextColonIf -> KW_ENDIF expressionEnd

363.classNextIf -> KW_ELSE classElseIf
364.classNextIf -> KW_ELSEIF OPEN_BRACKET fillLine CLOSE_BRACKET classIfBlock
365.classNextIf ->

366.classElseIf -> KW_IF OPEN_BRACKET fillLine CLOSE_BRACKET classIfBlock
367.classElseIf -> classElseBlock

368.classCaseBlock -> OPEN_BLOCK classCase CLOSE_BLOCK
369.classCaseBlock -> COLON classCase KW_ENDSWITCH

370.classCase -> KW_CASE fillLine COLON classNextFunctionBlock classCase
371.classCase -> KW_DEFAULT COLON classNextFunctionBlock classCase
372.classCase ->

373.classWhileBlock -> classExpression
374.classWhileBlock -> OPEN_BLOCK classNextFunctionBlock CLOSE_BLOCK
375.classWhileBlock -> COLON classNextFunctionBlock KW_ENDWHILE expressionEnd

376.classDoBlock -> classExpression
377.classDoBlock -> OPEN_BLOCK classNextFunctionBlock CLOSE_BLOCK

378.nextConst -> COMMA IDENT ASSIGN constValue nextConst
379.nextConst -> SEMICOLON

380.nextClassAsign -> COMMA __CLASS_PROPERTY_DECLARE VAR classAsign nextClassAsign
381.nextClassAsign ->

382.classAsign -> ASSIGN __L_CLASS_VARIABLE_ASSIGN classDeclarationValue __P_CLASS_VARIABLE_ASSIGN
383.classAsign ->

384.interfaceDeclaration -> KW_INTERFACE __INTERFACE_DECLARATION IDENT interfaceExtendsList OPEN_BLOCK interfaceBlock CLOSE_BLOCK

385.interfaceExtendsList -> KW_EXTEND __INTERFACE_EXTENDS IDENT nextExtend
386.interfaceExtendsList ->

387.nextExtend -> COMMA IDENT nextExtend
388.nextExtend ->

389.interfaceBlock -> KW_FUNCTION __INTERFACE_METHOD_DECLARATION IDENT OPEN_BRACKET paramDefinition CLOSE_BRACKET expressionEnd
    interfaceBlock
390.interfaceBlock -> KW_CONST __INTERFACE_CONST_DECLARE IDENT ASSIGN __L_INTERFACE_CONST_VALUE constValue __P_INTERFACE_CONST_VALUE
    nextConst
391.interfaceBlock ->

392.variable -> __VARIABLE_CALL VAR object
393.variable -> DOLLAR OPEN_BLOCK fillLine CLOSE_BLOCK object

394.object -> INSTANCE_PROPERTY identAndVar object
395.object -> __P_INSTANCE_PROPERTY_CALL OPEN_LIST line CLOSE_LIST object
396.object -> __P_INSTANCE_PROPERTY_CALL __STRING_PART OPEN_BLOCK fillLine CLOSE_BLOCK object
397.object -> __P_INSTANCE_PROPERTY_CALL

398.identAndVar -> VAR
399.identAndVar -> IDENT
400.identAndVar -> DOLLAR OPEN_BLOCK fillLine CLOSE_BLOCK
401.identAndVar -> keyWords

402.keyWords -> KW_IF
403.keyWords -> KW_TRY
404.keyWords -> KW_CATCH
405.keyWords -> KW_FOR
406.keyWords -> KW_FOREACH
407.keyWords -> KW_SWITCH
408.keyWords -> KW_WHILE
409.keyWords -> KW_DECLARE
410.keyWords -> KW_DO
411.keyWords -> KW_ECHO
412.keyWords -> KW_ENDDECLARE
413.keyWords -> KW_AS
414.keyWords -> KW_ENDFOR
415.keyWords -> KW_ENDFOREACH
416.keyWords -> KW_ELSE
417.keyWords -> KW_ENDIF
```

```
418.keyWords -> KW_ELSEIF
419.keyWords -> KW_ENDSWITCH
420.keyWords -> KW_CASE
421.keyWords -> KW_DEFAULT
422.keyWords -> KW_ENDWHILE
423.keyWords -> KW_BREAK
424.keyWords -> KW_CONTINUE
425.keyWords -> KW_RETURN
426.keyWords -> KW_GLOBAL
427.keyWords -> KW_THROW
428.keyWords -> KW_INCLUDE
429.keyWords -> KW_INCLUDE_ONCE
430.keyWords -> KW_REQUIRE
431.keyWords -> KW_REQUIRE_ONCE
432.keyWords -> KW_STATIC
433.keyWords -> KW_EXIT
434.keyWords -> KW_DIE
435.keyWords -> KW_LIST
436.keyWords -> KW_EMPTY
437.keyWords -> KW_T_LOGICAL_AND
438.keyWords -> KW_T_LOGICAL_OR
439.keyWords -> KW_T_LOGICAL_XOR
440.keyWords -> KW_CLONE
441.keyWords -> KW_ARRAY
442.keyWords -> KW_INT
443.keyWords -> KW_FLOAT
444.keyWords -> KW_STRING
445.keyWords -> KW_BOOL
446.keyWords -> KW_OBJECT
447.keyWords -> KW_TRUE
448.keyWords -> KW_FALSE
449.keyWords -> KW_NULL
450.keyWords -> KW_NEW
451.keyWords -> KW_PRIVATE
452.keyWords -> KW_PUBLIC
453.keyWords -> KW_PROTECTED
454.keyWords -> KW_FINAL
455.keyWords -> KW_ABSTRACT
456.keyWords -> KW_FUNCTION
457.keyWords -> KW_CLASS
458.keyWords -> KW_EXTEND
459.keyWords -> KW_IMPLEMENT
460.keyWords -> KW_CONST
461.keyWords -> KW_VAR
462.keyWords -> KW_INTERFACE
463.keyWords -> KW_INSTANCEOF

464.expressionEnd -> SEMICOLON
465.expressionEnd -> PHP_CLOSE_TAG HTML PHP_OPEN_TAG
```

Řešení

Lexikální analyzátor

Lexikální analyzátor zajišťuje třída Lexan, v souboru class.lexan.php (http://pjp.yosarin.net/?c=kod2). Rozlišuje základní prvky jazyka. Výstupem je zvýrazněný kód (pro uživatele) a řetězec obsahující jednotlivé jazykové elementy (pro syntaktický analyzátor). V případě že na vstupu je řetězec který neodpovídá požadavkům na prvky jazyka, vyhodí skript vyjímku a upozorní k jaké chybě a kde došlo. Vstupem lexikálního analyzátoru je objekt typu KeyWord, který sdružuje všechna klíčová slova a speciální symboly (jako * pro násobení a podobně) jazyka.

Syntaktický analyzátor

Syntaktický analyzátor zajišťuje třída Syntan, v souboru class.lexan.php (http://pjp.yosarin.net/?c=kod3). S pomocí rozkladové tabulky (převedené do asociativního pole) poté prochází jednotlivé symboly jejichž posloupnost vytvořil lexikální analyzátor a kontroluje zda odpovídají dané gramatice. V případě že neodpovídají, vyhodí skript vyjímku s informací který prvek ji způsobil. Tato informace se předá zpět lexikálnímu analyzátoru a ten zjistí kde se daný prvek nachází a jaká je jeho přesná hodnota.

Vstupem pro syntaktický analyzátor je seznam terminálů, neterminálů, výstupních symbolů a asociativní pole reprezentující rozkladovou tabulku. Všechna tato pole se generují dalšími skripty přímo z gramatiky, uloží pomocí serialize do souborů a následně jsou načteny a použity.

Rozkladová tabulka

Rozkladová tabulka je zpracována do asociativního pole. Klíčem pole je stav ve kterém se syntaktický analyzátor zrovna nachází (na počátku je to v našem případě stav 'file') a prvkem je další asociativní pole,

kde jsou klíči povolené terminály jazyka a obsahem pravidla (seznam stavů a terminálů) která po příchodu daného terminálu mají následovat.

Závěr

Jak lexikální tak syntaktický analyzátor pracují a fungují s vysokou mírou spolehlivosti. U syntaktického analyzátoru jsou možné menší odchylky, způsobené lehkými nepřesnostmi v použité gramatice. Úplně do konce není dotažené použití výstupních symbolů, které se v současné době používají pouze pro určení funkcí, interfaců a tříd, jejich proměnných, metod a konstant a k určení jejich vstupních parametrů a výchozích hodnot.