**‘JSPer’ 기술문서**

제작순서:

1. ANTLR의 JavaScript Grammar를 이용하여 JavaScript parser(.Java) 제작.

(Structure Diagram parser, Flowchart parser)

1. 제작된 parser를 사용하여 원하는 정보를 추출하는 Controller단(.java) 제작.
2. HTML, CSS, JavaScript를 이용하여 View제작. (Jquery, Jquery.layout, ACE js, Rapael 사용)
3. Apache Tomcat v7.0사용, servlet, Java beans를 이용하여 서버구성.

ANTLR (Another Tool for Language Recognition)

: Grammar파일(.g)을 이용해서 특정 언어의 parser를 제작할 수 있는 도구.

1. 정의된 Token을 이용하여 필요한 함수정의.
2. 중간에 Java 문법을 Action Code({~})에 삽입하여 원하는 정보를 Java 형식으로 추출.

: Structure Grammar (함수를 기반으로 코드의 구조를 파악하기 위해 정보를 출력하는 파서)

주요수정부분:

|  |
| --- |
| grammar JS;  options  {  output=AST;  backtrack=true;  //k = 1;  memoize=true;  }  program  : {System.out.println(" ----- ANTLR Total Parser Start! ----- ");}  LT!\* sourceElements LT!\* EOF!  ;    sourceElements  : sourceElement (LT!\* sourceElement)\*  ;    sourceElement  : functionDeclaration  | statement  ;    // functions  functionDeclaration  : functionComment\* LT!\* 'function' LT!\* functionName {type="Declaration";} LT!\* formalParameterList LT!\* functionBody  ;  functionExpression  //: functionComment\* LT!\* 'var'? LT!\* functionName {fList.get(fList.size()-1).setType("Expression");} LT!\* '=' LT!\* 'function' LT!\* formalParameterList LT!\* functionBody  : functionComment\* LT!\* 'var'? LT!\* functionName {type="Expression";} LT!\* '=' LT!\* 'function' LT!\* formalParameterList LT!\* functionBody  ;  functionAnonymous  : functionComment\* '(' LT!\* 'function' {name="Anonymous"; type="Anonymous";} LT!\* formalParameterList LT!\* functionBody LT!\* ')'  ;    functionName  :  ( Identifier )  {  name = $Identifier.text;  //insertFunction();  }  ;  functionComment  :  ( Comment LT!\* )  {  comment = $Comment.text;  }  ;    formalParameterList  : '(' (LT!\* Identifier (LT!\* ',' LT!\* Identifier)\*)? LT!\* ')'  ;  functionBody  //: '{' {depth++;} LT!\* sourceElements? LT!\* {depth--;}'}'  : '{' {insertFunction(); depth++; } LT!\* functionCode? LT!\* {depth--; cList.add(new CodeMap(depth,code));}'}'  ;  functionCode  : ( sourceElements )  {  code = $sourceElements.text;  }  ; |

: Flowchart Grammar (함수의 내부코드를 분석하여 흐름을 분석하기 위해 정보를 출력하는 파서)

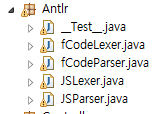
주요수정부분:

|  |
| --- |
| grammar fCode;  options  {  output=AST;  backtrack=true;  //k = 1;  memoize=true;  }  program  : {System.out.println(" ----- ANTLR Function Parser Start! ----- ");}  LT!\* sourceElements LT!\* EOF!  ;    sourceElements  : sourceElement (LT!\* sourceElement)\*  ;    sourceElement  : functionDeclaration  | functionExpression  | functionAnonymous  | statement  ;    // functions  functionDeclaration  : functionComment\* LT!\* 'function' LT!\* functionName {stmType="func";} LT!\* formalParameterList LT!\* functionBody  ;  functionExpression  //: functionComment\* LT!\* 'var'? LT!\* functionName {fList.get(fList.size()-1).setType("Expression");} LT!\* '=' LT!\* 'function' LT!\* formalParameterList LT!\* functionBody  : functionComment\* LT!\* 'var'? LT!\* functionName {stmType="func";} LT!\* '=' LT!\* 'function' LT!\* formalParameterList LT!\* functionBody  ;  functionAnonymous  : functionComment\* '(' LT!\* 'function' {stmType="func"; stmText="func Anonymous";} LT!\* formalParameterList LT!\* functionBody LT!\* ')'  ;    functionName  :  ( Identifier )  {  stmText = "func "+$Identifier.text;  }  ;  functionComment  :  ( Comment LT!\* )  /\*{  comment = $Comment.text;  }\*/  ;    formalParameterList  : '(' (LT!\* Identifier (LT!\* ',' LT!\* Identifier)\*)? LT!\* ')'  ;  functionBody  : '{'{insertStment(); fDepth++;} LT!\* sourceElements? {fDepth--; } LT!\*'}'  ;  // statements  statement  : statementBlock  | variableStatement  | variableChangeStatement  | emptyStatement  | expressionStatement  | ifStatement  | elseifStatement  | elseStatement  | iterationStatement  | continueStatement  | breakStatement  | returnStatement  | withStatement  | labelledStatement  | switchStatement  | throwStatement  | tryStatement  ;    statementBlock  : '{' LT!\* statementList? LT!\* '}'  ;    statementList  : statement (LT!\* statement)\*  ;  variableStatement  : 'var' LT!\* { stmType = "var"; stmText = "var "; } variableDeclarationList { insertStment();}(LT | ';')!  ;    variableChangeStatement  : { stmType = "var"; stmText = ""; } expression1 LT!\* { stmText +=";"; insertStment();}(LT | ';')!  ;    variableDeclarationList  : variableDeclaration (LT!\* ',' { stmText = stmText+","; } LT!\* variableDeclaration)\* {stmText += ";"; }  ;    variableDeclarationListNoIn  : variableDeclarationNoIn (LT!\* ',' LT!\* variableDeclarationNoIn)\*  ;    variableDeclaration  : variableName LT!\* initialiser?  ;  variableName  :  ( Identifier )  {  stmText = stmText + $Identifier.text;  }  ;    variableDeclarationNoIn  : LT!\* initialiserNoIn?  ;    initialiser  //: '=' LT!\* initialization  : '=' LT!\* assignmentExpression { stmText += "="+$assignmentExpression.text; }  //: ('=' LT!\* {stmText+="=";} initialization | '+=' LT!\* {stmText+="+=";}initialization)  ;  initialiserNoIn  : '=' LT!\* assignmentExpressionNoIn  ;    emptyStatement  : ';'  ;    expressionStatement  : expression (LT | ';')!  ;    ifStatement  : 'if' LT!\* '(' LT!\* expression LT!\* ')' LT!\* {stmType="if";stmText="if("+$expression.text; stmText+=")"; insertStment(); stmDepth++;} statement {stmDepth--;}  ;    elseifStatement  : 'else' LT!\* 'if' LT!\* '(' LT!\* expression LT!\* ')' LT!\* {stmType="elif";stmText="else if("+$expression.text; stmText+=")"; insertStment(); stmDepth++;} statement {stmDepth--;}  ;  elseStatement  : 'else' LT!\* {stmType="else"; stmText="else"; insertStment(); stmDepth++;} statement {stmDepth--;}  ;    iterationStatement  : doWhileStatement  | whileStatement  | forStatement  | forInStatement  ;    doWhileStatement  : 'do' LT!\* {stmType="do"; stmText="do"; insertStment(); stmDepth++;} statement {stmDepth--;} LT!\* 'while' LT!\* '(' expression ')' {stmType="while";stmText="while("+$expression.text; stmText+=");"; insertStment();}(LT | ';')!  ;    whileStatement  : 'while' LT!\* '(' LT!\* expression LT!\* ')' LT!\* {stmType="while";stmText="while("+$expression.text; stmText+=")"; insertStment(); stmDepth++;} statement {stmDepth--;}  ;    forStatement  : 'for' LT!\* '(' {stmType="for"; stmText="for(";} (LT!\* forStatementInitialiserPart1)? LT!\* ';' {stmText+= ";";}(LT!\* expression1)? LT!\* ';' {stmText+= ";";}(LT!\* expression2)? LT!\* ')' {stmText+= ")"; insertStment(); stmDepth++;} LT!\* statement {stmDepth--;}  ;  forStatementInitialiserPart1  :  ( forStatementInitialiserPart )  {  stmText += $forStatementInitialiserPart.text;  }  ;  expression1  :  ( expression )  {  stmText += $expression.text;  }  ;  expression2  :  ( expression )  {  stmText += $expression.text;  }  ;  forStatementInitialiserPart  : expressionNoIn  | 'var' LT!\* variableDeclarationListNoIn  ;    forInStatement  : 'for' LT!\* '(' LT!\* forInStatementInitialiserPart LT!\* 'in' LT!\* expression LT!\* ')' LT!\* statement  ;  /\*  fluctuationStatement  : LT!\* fluctuationOperation (LT | ';')!  ;  fluctuationOperation  : '++' | '--'  ;  \*/    forInStatementInitialiserPart  : leftHandSideExpression  | 'var' LT!\* variableDeclarationNoIn  ;  continueStatement  : 'continue' Identifier? {stmType="continue"; stmText="continue;"; insertStment();} (LT | ';')!  ;  breakStatement  : 'break' Identifier? {stmType="break"; stmText="break;"; insertStment();} (LT | ';')!  ;  returnStatement  : 'return' expression? (LT | ';')!  ;    withStatement  : 'with' LT!\* '(' LT!\* expression LT!\* ')' LT!\* statement  ;  labelledStatement  : Identifier LT!\* ':' LT!\* statement  ;    switchStatement  : 'switch' LT!\* '(' LT!\* expression LT!\* ')' LT!\* {stmType="switch"; stmText="switch("+$expression.text; stmText+=")"; insertStment();} caseBlock {}  ;    caseBlock  : '{' (LT!\* caseClause)\* (LT!\* defaultClause (LT!\* caseClause)\*)? LT!\* '}'  ;  caseClause  : 'case' LT!\* expression LT!\* ':' LT!\* {stmType="case"; stmText="case "+$expression.text; stmText+=":"; insertStment(); stmDepth++;} statementList? {stmDepth--;}  ;    defaultClause  : 'default' LT!\* ':' LT!\* {stmType="default"; stmText="default:"; insertStment(); stmDepth++;} statementList? {stmDepth--;}  ;    throwStatement  : 'throw' expression (LT | ';')!  ;  tryStatement  : 'try' LT!\* {stmType="try"; stmText="try"; insertStment(); stmDepth++;} statementBlock {stmDepth--;} LT!\* (finallyClause | catchClause (LT!\* finallyClause)?)  ;    catchClause  : 'catch' LT!\* '(' LT!\* Identifier LT!\* ')' LT!\* {stmType="catch"; stmText="catch("+$Identifier.text; stmText+=")"; insertStment(); stmDepth++;} statementBlock {stmDepth--;}  ;    finallyClause  : 'finally' LT!\* {stmType="finally"; stmText="finally"; insertStment(); stmDepth++;} statementBlock {stmDepth--;}  ; |

: 제작된 Parser, Lexer;

Structure files: JSLexer.java, JSParser.java.

Flowchart files: fCodeLexer.java, fCodeParser.java



(\_TEST\_.java 는 테스트파일)

데이터 흐름

View (.java)

Code

: 사용자에게 코드를 입력 받아 Servlet으로 전송

Diagram

: Run클릭시 분석된 코드 구조를 Diagram으로 출력.

Flowchart

: Diagram클릭시 해당하는 Function의 Flowchart를 출력

Model (.java)

: parser에서 추출된 정보를 담을 수 있는 Class (Bean)정의

Function.java

: 전체적인 함수의 구조를 Diagram으로 그리기 위한 클래스

Stment.java

: 함수내부의 Flowchart를 그리기 위한 정보를 담는 클래스.

CodeMap.java

: flowchart를 그리기 전에 정보들이 순서를 정렬하기 위해 사용되는 객체

Controller (.java)

: 제작된 parser, lexer를 사용하여 정보를 추출해내고 Model과 View를 연결한다.

MainController.java

: 최초로 코드를 받아서 해당하는 parser로 전송하여 정리된 Data를 받는다.

ChangeStmlist.java

MainController로 들어온 2차원 배열 Data를 그림을 그리기 위해 1차원 배열로 정리해주는 클래스

chartSetting.java

: Flow Chart를 그리기 위한 배열을 제작하는 클래스

