JavaScript to jQuery converter

Documentazione progetto

Jacopo Boffelli [1053217]
Mauro Riva [1053644]

A.A. 2022/2023

1 Introduzione

L'obiettivo del progetto è la traduzione automatica di un codice in JavaScript nativo nel corrispettivo in jQuery. L'applicazione si presenta con un'interfaccia grafica che permette all'utente di selezionare un file di input, la directory in cui salvare l'output e la console per la comunicazione di eventuali messaggi.
La filosofia generale dell'analisi sintattica è quella di essere più permissiva rispetto agli analizzatori sul mercato in quanto lo scopo finale non è l'esecuzione del codice, la cui correttezza non dipende dalla traduzione ma dal solo codice fornito in input.
Per quanto riguarda i principi su cui si basa di traduzione sono di convertire solo quando vi è la sicurezza di non alterare il senso logico del codice altrimenti si preferisce segnalare all'utente il potenziale problema indicando la posizione nel file senza effettuare la traduzione.

2 Tecnologie utilizzate

- ANTLR 3.4: è un framework open-source utilizzato per generare il parser.
- AntlrWorks 1.5.2: è l'*IDE* utilizzato per la modifica dei file contenenti la grammatica (estensione .g).
- Java: linguaggio di programmazione impiegato per lo sviluppo dell'applicazione.
- TokenRewriteStream: oggetto usato in sostituzione del *CommonTokenStream* che consente la manipolazione del flusso di token in input.
- Swing: framework Java per lo sviluppo dell'interfaccia grafica.
- Eclipse: ambiente di sviluppo per il software.
- GitHub: per la condivisione del codice e il controllo di versione.
- Overleaf: compilatore *LaTeX* usato per la stesura della documentazione.
- Canva: strumento di progettazione grafica per la creazione della presentazione.

3 Diagrammi

3.1 JS2JQConverter package

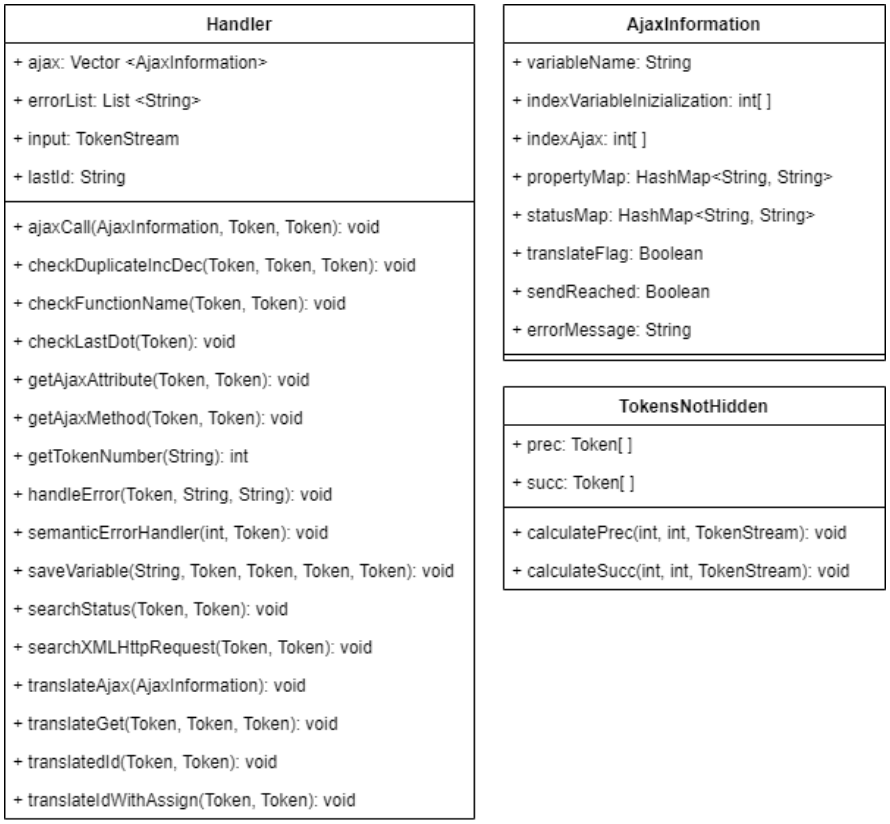

Figura 1: Diagramma UML delle classi del package JS2JQConverter

3.1.1 Handler.java - Attributi

- **Vector <AjaxInformation> ajax**: memorizza le informazioni relative a tutti gli oggetti *XMLHttpRequest* inizializzati nel file di input.
- **List <String> errorList**: contiene tutti i messaggi di errore.
- **TokenStream input**: oggetto per la gestione del flusso di token, viene effettuato il casting esplicito in *TokenRewriteStream*.
- **String lastId**: variabile di supporto per controlli sui dati presenti nell'oggetto ajax.

3.1.2 Handler.java - Metodi

- **private void ajaxCall(AjaxInformation, Token, Token)**: riceve come parametri l'oggetto con le informazioni relative alla specifica chiamata ajax e i token start e

stop della regola in cui è presente l'oggetto di tipo *XMLHttpRequest*, se l'inizio del blocco ajax non è ancora inizializzato lo imposta all'indice di start, in ogni caso imposta la fine a stop.

- **public void checkDuplicateIncDec(Token, Token, Token)**: controllo semantico della presenza simultanea di incremento o decremento che precedono e seguono il token passato come terzo parametro.
- **public void checkFunctionName(Token, Token)**: controllo semantico della presenza di un nome quando una funzione viene dichiarata al di fuori di un assegnamento.
- **public void checkLastDot(Token)**: controllo semantico della presenza del punto come ultimo token di una variabile.
- **public void getAjaxAttribute(Token, Token)**: acquisisce le informazioni relative ad una chiamata ajax e le riscrive come parametri della chiamata effettuata con jQuery.
- **public void getAjaxMethod(Token, Token)**: acquisisce le informazioni relative ad una chiamata ajax e se non ci sono problemi una volta raggiunta la fine della chiamata, cioè quando viene trovato il metodo send(), avvia la traduzione.
- **public int getTokenNumber(String)**: riceve come parametro il nome di un token e restituisce il suo numero corrispondente nel file *.tokens*.
- **public void handleError(Token, String, String)**: inserisce nuovi messaggi di errore (lessicali e sintattici) all'errorList.
- **public void semanticErrorHandler(int, Token)**: inserisce nuovi messaggi di errore semantico all'errorlist.
- **public void saveVariable(String, Token, Token, Token, Token)**: controlla se il token è un ID e se il suo testo è già presente come nome di variabile ajax, in quel caso chiama l'*ajaxCall*.
- **public void searchStatus(Token, Token)**: cerca se all'interno delle funzioni eseguite in seguito ad una chiamata ajax è presente un blocco di if che specifica cosa fare in base allo status restituito, in tal caso memorizza le informazioni.
- **public void searchXMLHttpRequest(Token, Token)**: controlla se vi è una nuova inizializzazione di un oggetto *XMLHttpRequest*, in quel caso salva il nome della variabile nel vettore ajax.

- **public void translateAjax(AjaxInformation)**: traduce il blocco ajax in jQuery se tutti i parametri della specifica sono soddisfatti, altrimenti comunica l'impossibilità della conversione tramite console.
- **public void translateGet(Token, Token, Token)**: traduce i metodi di manipolazione del DOM.
- **public void translatedId(Token, Token)**: traduce gli attributi JavaScript negli equivalenti in jQuery.
- **public void translateIdWithAssign(Token, Token)**: traduce gli attributi JavaScript che hanno alla loro sinistra un assegnamento.

3.1.3 AjaxInformation.java - Attributi

- **String variableName**: memorizza il nome dell'oggetto
- **int[] indexVariableInizialization;**: array bidimensionale che contiene l'indice iniziale e finale della riga di codice in cui viene inizializzato l'oggetto *XMLHttpRequest*.
- **int[] indexAjax**: array bidimensionale che contiene l'indice iniziale e finale del blocco della chiamata ajax.
- **HashMap<String, String> propertyMap**: memorizza le proprietà definite nel blocco.
- **HashMap<Integer, String> statusMap**: memorizza le istruzioni associate agli stati.
- **Boolean translateFlag**: booleano per stabilire se effettuare la traduzione
- **Boolean sendReached**: booleano per stabilire se il metodo send() di questa chiamata ajax è stato raggiunto.
- **String errorMessage**: contiene i messaggi di errore.

3.1.4 JavaScriptToJQueryConverterLexer.java

Classe generata automaticamente da Antlrworks.

3.1.5 JavaScriptToJQueryConverterParser.java

Classe generata automaticamente da Antlrworks.

3.1.6 TokensNotHidden.java - Attributi

- **Token[] prec**: contiene gli n token non nascosti precedenti al token passato come parametro alla funzione.
- **Token[] succ**: contiene gli n token non nascosti successivi al token passato come parametro alla funzione.

3.1.7 TokensNotHidden.java - Metodi

- **public void calculatePrec(int, int, TokenStream)**: calcola il vettore dei precedenti.
- **public void calculateSucc(int, int, TokenStream)**: calcola il vettore dei successivi.

3.1.8 JavaScriptToJQueryConverter.g

Contiene la grammatica decorata dell'applicazione.

3.1.9 JavaScriptToJQueryConverter.tokens

Contiene l'elenco del nome dei token associati ad un numero.

3.2 JS2JQGui package

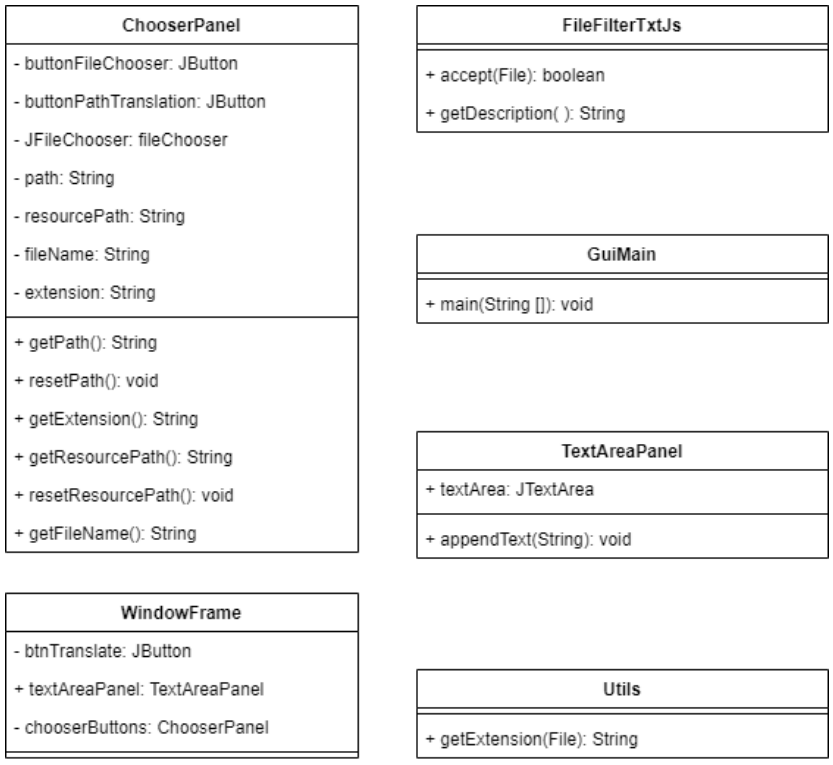

Figura 2: Diagramma UML delle classi del package JS2JQGui

3.2.1 GuiMain - Metodi

- **public void main(String[])**: funzione main per l'avvio dell'applicazione.

3.2.2 ChooserPanel.java - Attributi

- **JButton buttonFileChooser**: pulsante per la selezione della directory del file di input.
- **JButton buttonPathTranslation**: pulsante per la selezione della directory del file di output.
- **JFileChooser fileChooser**: oggetto per la selezione della directory.
- **String path**: percorso del file contente la traduzione.
- **String resourcePath**: percorso del file di input.
- **String fileName**: variabile contenente il nome del file.
- **String extension**: variabile contenente l'estensione del file.

3.2.3 WindowFrame.java - Attributi

- **JButton btnTranslate**: pulsante per l'avvio della traduzione.
- **TextAreaPanel textAreaPanel**: area di testo per la visualizzazione dei messaggi.
- **ChooserPanel chooserButtons**: pannello contenente i due pulsanti per la selezione delle directory.

3.2.4 FileFilterTxtJs - Metodi

- **boolean accept(File)**: verifica che l'estensione del file sia una di quelle indicate, ritorna true se è corretta.
- **String getDescription()**: restituisce una stringa che specifica la descrizione del tipo di file accettati.

3.2.5 TextAreaPanel - Attributi

- **JTextArea textArea**: oggetto che costituisce l'area di testo per la visualizzazione dei messaggi.

3.2.6 TextAreaPanel - Metodi

- **public void appendText(String)**: permette di aggiungere del testo alla text area.

3.2.7 Utils - Metodi

- **String getExtension(File)**: permette di recuperare l'estensione del file selezionato per l'input.

3.3 JS2JQStarter package

3.3.1 JS2JQParserStarter - Attributi

- **String consoleOutput**: stringa contenente i messaggi da visualizzare sulla console.
- **JavaScriptToJQueryConverterParser parser**: oggetto per richiamare le funzioni di parsing generate da Antlr.

3.3.2 JS2JQParserStarter - Metodi

- **public void main(String [])**: funzione main per l'avvio della traduzione.

4 Azioni semantiche principali

4.1 Traduzione manipolatori DOM

Al raggiungimento di un token DOCUMENT viene riconosciuta la regola sintattica chiamata *getRule* la quale al termine invoca il metodo *translateGet(Token, Token, Token)* che riceve come parametri il token che rappresenta l'ID che segue DOCUMENT e rappresenta la tipologia di traduzione da effettuare, il token start e il token stop. L'idea alla base della conversione è di costruire una stringa che contiene il testo o la variabile (o concatenazione di più elementi) passati alla funzione JavaScript aggiungendoci un prefisso. Si distinguono quattro diversi casi:

- **getElementById**: prefisso '#'
- **getElementsByClass**: prefisso '.'
- **getElementsByName**: prefisso '[name = ...]'
- **getElementsByTagName**: nessun prefisso

L'intero testo della *getRule* viene sostituito da $(prefisso + parametro), seguito da eventuali postfissi non modificati.

4.2 Attributi speciali JavaScript

Quando una *getRule* è seguita da una assegnamento (cioè da un uguale o da un più-uguale) o quando la regola inizia con un ID (*idStartingRule*) viene invocato il metodo *translateIdWithAssign(Token, Token)* che controlla la presenza di attributi di JavaScript che hanno una corrispettiva funzione in jQuery:

- object.**value**: viene tradotto in object.*val()* se non è seguito da un assegnamento, altrimenti in object.*val(parametro)*.
- object.**innerHTML**: viene tradotto in object.*html()* se non è seguito da un assegnamento, altrimenti in object.*html(parametro)*.
- object.**textContent**: viene tradotto in object.*text()* se non è seguito da un assegnamento, altrimenti in object.*text(parametro)*.

Se l'assegnamento consiste in un più-uguale il parametro avrà concatenato un prefisso contenente il valore dell'oggetto stesso su cui si sta eseguendo l'assegnamento.

- object.**style.display = ”none”**: viene tradotto in object.*hide()*.
- object.**style.display = ”block”**: viene tradotto in object.*show()*.

L’ultimo attributo considerato in questo progetto è *style*, il quale permette di modificare il *CSS* di un oggetto:

- object.**style.proprietà**: viene tradotto in object.*css(”proprietà”)* se non è seguito da un assegnamento, altrimenti in object.*css(”proprietà”, ”valore”)*.

4.3 Chiamate AJAX

La traduzione delle chiamate AJAX è la componente più complessa e critica del progetto. È suddivisa in due operazioni tra loro sequenziali: la raccolta delle informazioni e la riscrittura nel formato jQuery. La prima fase ha inizio quando viene identificata la dichiarazione di un nuovo oggetto di tipo *XMLHttpRequest*. Il nome della variabile a cui è stato assegnato questo oggetto viene memorizzato in un vettore di tipo *AjaxInformation* e si comporterà come identificatore della specifica chiamata. Ogni qual volta questa variabile verrà ritrovata nel resto del codice, sia per assegnamenti ad un attributo o per la chiamata di un metodo della classe *XMLHttpRequest*, si ricava l'informazione e la si memorizza nell'elemento corrispondente nel vettore. Il processo di acquisizione dati continua fino a quando viene invocato sull'oggetto il metodo *send()*. Per garantire una corretta traduzione è stato necessario inserire alcuni vincoli:

- È richiesta la presenza di alcuni parametri obbligatori: URL, metodo (GET o POST) e almeno una funzione assegnata ad *onload* o ad *onreadystatechange*.
- Nelle funzioni assegnate ai sopracitati attributi è possibile specificare in un *if-statement* le azioni da compiere sulla base dello stato restituito dalla chiamata. I vincoli sono suddivisi in varie categorie:
  - All'esterno del blocco if contenente gli stati non devono esserci altre istruzioni, ne prima ne dopo, quindi la funzione deve contenere esclusivamente l'if-statement.
  - Uno stato non può avere assegnati più blocchi di codice.
  - Tutte le condizioni nell'if-statement degli stati non possono avere altre operazioni logiche (concatenate con &&, ||).
  - Nell'if-statement è consentita la presenza di molteplici *else if* ma non di un *else*.

Se anche uno solo di questi vincoli non è rispettato la chiamata AJAX non viene tradotta e quindi risulterà invariata nel file di output, altrimenti verrà sostituita con la chiamata in formato jQuery costruita a partire dalle informazioni acquisite.

5 Grammatica non decorata

```
grammar JavaScriptToJQueryConverter;
options {
        language = Java;
        k = 1;
}

@header {
        package JS2JQConverter;
}

@lexer::header {
        package JS2JQConverter;
}

@members {
        Handler h;
        public Handler getHandler () {
                return h;
        }
        public void displayRecognitionError(String[] tokenNames,
                                            RecognitionException e) {
            String hdr = " * " + getErrorHeader(e);
            String msg = " - " + getErrorMessage(e, tokenNames);
            Token tk = input.LT(1);
                h.handleError(tk, hdr, msg);
  }
        void initParser () {
                h = new Handler(input);
        }
}
```

```
parseJava
        :
                (instructionRule
                | blockRule
                | classRule)*
        ;

idDotIdRule
        :
                ID (DOT ID)*
        ;

factorTypologyRule
        :
                (STRING
                | INTEGER
                | FLOAT
                | TRUE
                | FALSE
                | idDotArrayRule
                | getRule )
        ;

assignTypologyRule
        :
        (objectRule | arrayRule | NULL | UNDEFINED
        | functionDefinitionRule | newRule)
        ;
```

```
idDotArrayRule
        :
                incDecRule?
                (THIS DOT)?
                (idDotIdRule
                        (LB expressionRule RB)* DOT?
                        (LP parametersRule? RP DOT?)?
                )+
                incDecRule?
        ;

parametersRule
        :
                (assignTypologyRule | expressionRule)(CM
            (assignTypologyRule | expressionRule))*
        ;
expressionRule
        :
                termRule ((ADD | SUB | MOD) termRule)*
        ;
termRule
        :
                factorRule ((STAR | DIV | EXP | XOR
                | AND_BIT | OR_BIT
                | LSHIFT | RSHIFT | URSHIFT) factorRule)*
        ;

factorRule
        :
                factorTypologyRule
                | LP expressionRule RP
        ;
```

```
instructionRule
        :
                (BREAK
                | CONTINUE
                | tryCatchRule
                | functionDefinitionRule
                | documentRule
                | ifStatementRule
                | switchCaseRule
                | forRule
                | whileRule
                | doWhileRule
                | throwRule
                | typeOfRule
                | idStartingRule)SC?
        ;

throwRule
        :
                THROW (STRING | TRUE | FALSE | INTEGER
            | FLOAT | idDotArrayRule)
        ;

typeOfRule
        :
                TYPEOF assignTypologyRule
        ;



tryCatchRule
        :
                TRY blockRule
                CATCH LP ID RP blockRule
                (FINALLY blockRule)?
        ;
```

```
returnRule
        :
                RETURN (expressionRule|assignTypologyRule) SC?
        ;

functionDeclarationRule
        :
        FUNCTION ID?
        LP ((assignTypologyRule|expressionRule)
        ((CM (assignTypologyRule|expressionRule))*))? RP
        ;

functionDefinitionRule
        :
                functionDeclarationRule
                LBR
                        instructionRule*
                        returnRule?
                RBR
        ;

arrayRule
        :
                LB
                        (
                        (expressionRule|assignTypologyRule)
                        (CM (expressionRule|assignTypologyRule)?)*
                        )?
                RB
        ;
```

```
objectRule
        :
        LBR
                (
                (ID | STRING)
                CL
                (expressionRule|assignTypologyRule)
                (
                CM
                (ID | STRING)
                CL
                (expressionRule|assignTypologyRule)
                )*
                )?
        RBR
        ;
variableDefinitionRule
        :
            (VAR | LET | CONST)?
            idDotArrayRule
            ((ASSIGN|PLUSEQ|MINUSEQ|STAREQ|DIVEQ|MODEQ|EXPEQ)
            (expressionRule|assignTypologyRule))?

        ;
documentRule
        :
            getRule ((ASSIGN|PLUSEQ)
            (expressionRule|assignTypologyRule))?
        ;
getRule
        :
            DOCUMENT DOT ID LP ((idDotArrayRule|STRING)
            (ADD (idDotArrayRule | STRING))*) RP
            (DOT idDotArrayRule)?
        ;
```

```
idStartingRule
        :
        (VAR | LET | CONST)?
        idDotArrayRule
        (((ASSIGN|PLUSEQ|MINUSEQ|STAREQ|DIVEQ|MODEQ|EXPEQ)
        (expressionRule|assignTypologyRule))? | INSTANCEOF ID)
        ;


newRule
        :
            NEW ID LP ((expressionRule|assignTypologyRule)
            (CM(expressionRule|assignTypologyRule))*)? RP
        ;


comparatorRule
        :
            (EQ | NEQ | LT | LE | GT | GE | TEQ | NTEQ)
        ;

conditionRule
        :
            NOT? expressionRule
            (comparatorRule NOT? expressionRule)?
            ((AND|OR) conditionRule)*
        ;
blockRule
        :
                LBR (instructionRule|blockRule)* RBR
        ;
```

```
ifStatementRule
        :
                IF LP conditionRule RP
                (blockRule | instructionRule)
                (ELSE (IF LP conditionRule RP)?
                (blockRule | instructionRule))*
        ;

switchCaseRule
        :
                SWITCH LP (expressionRule|assignTypologyRule) RP
                LBR
                        (CASE (expressionRule|assignTypologyRule)
                CL instructionRule*
                        )*
                        (DEFAULT CL
                                instructionRule*
                        )?
                        (CASE (expressionRule|assignTypologyRule)
                CL instructionRule*
                        )*
                RBR
        ;

forRule
        :
          FOR LP forInitVarRule? SC conditionRule?
          SC idDotArrayRule? RP
          (blockRule | instructionRule)
        ;

forInitVarRule
        :
                (VAR | LET)? idDotArrayRule ASSIGN
                (expressionRule|assignTypologyRule)
        ;
incDecRule
```

```
        :
                DEC     {tk = $o1;}
                |INC    {tk = $o2;}
        ;

whileRule
        :
                WHILE LP conditionRule RP
                        (blockRule | instructionRule)
        ;

doWhileRule
        :
                DO
                        (blockRule | instructionRule)
                WHILE LP conditionRule RP SC?
        ;

classRule
        :
                CLASS ID
                LBR
                        functionDefinitionRule*
                RBR
        ;
```

```
fragment
EXPONENT : ('e'|'E') ('+'|'-')? ('0'..'9')+ ;

fragment
HEX_DIGIT : ('0'..'9'|'a'..'f'|'A'..'F') ;

fragment
ESC_SEQ
    :   '\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\'|'\/')
    |   UNICODE_ESC
    |   OCTAL_ESC
    ;

fragment
OCTAL_ESC
    :   '\\' ('0'..'3') ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7')
    ;

fragment
UNICODE_ESC
    :   '\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
    ;

fragment
LETTER : 'a'..'z'|'A'..'Z';
fragment
DIGIT : '0'..'9';
DOCUMENT
        :'document';

ASSIGN  : '=';
```

```
// comparatori
EQ        : '==';
TEQ       : '===';
NEQ       : '!=';
NTEQ      : '!==';
GT        : '>';
GE        : '>=';
LT        : '<';
LE        : '<=';
QUEST     : '?';

// punteggiatura
AT        : '@';
CL        : ':';
CM        : ',';
DOT       : '.';
SC        : ';';

// parentesi
LP        : '(';            // Parenthesis
RP        : ')';
LB        : '[';            // Brackets
RB        : ']';
LBR       : '{';            // BRaces
RBR       : '}';

// operatori
ADD               : '+';
SUB               : '-';
STAR              : '*';
DIV               : '/';
MOD               : '%';
INC               : '++';
DEC               : '--';
EXP               : '**';
PLUSEQ            : '+=';
MINUSEQ           : '-=';
```

```
STAREQ          : '*=';
DIVEQ           : '/=';
MODEQ           : '%=';
EXPEQ           : '**=';

// operatori logici
NOT             : '!';
AND             : '&&';
OR              : '||';
XOR             : '^';
AND_BIT         :'&';
OR_BIT          : '|';
LSHIFT          : '<<';
RSHIFT          : '>>';
URSHIFT         : '>>>';

// keywords
AWAIT           :'await';
BREAK           :'break';
CASE            :'case';
CATCH           :'catch';
CLASS           :'class';
CONST           :'const';
CONTINUE        :'continue';
DEBUGGER        :'debugger';
DEFAULT         :'default';
DELETE          :'delete';
DO              :'do';
ELSE            :'else';
ENUM            :'enum';
EXPORT          :'export';
EXTENDS         :'extends';
FALSE           :'false';
FINALLY         :'finally';
FOR             :'for';
FUNCTION        :'function';
IF              :'if';
```

```
IMPLEMENTS      :'implements';
IMPORT          :'import';
IN              :'in';
INSTANCEOF      :'instanceof';
INTERFACE       :'interface';
LET             :'let';
NEW             :'new';
NULL            :'null';
PACKAGE         :'package';
PRIVATE         :'private';
PROTECTED       :'protected';
PUBLIC          :'public';
RETURN          :'return';
STATIC          :'static';
SUPER           :'super';
SWITCH          :'switch';
THIS            :'this';
THROW           :'throw';
TRY             :'try';
TRUE            :'true';
TYPEOF          :'typeof';
VAR             :'var';
VOID            :'void';
WHILE           :'while';
WITH            :'with';
YIELD           :'yield';

UNDEFINED       : 'undefined';


ID      :
        ( LETTER |'_' | '$')( LETTER |DIGIT |'_' | '$')*
        ;


INTEGER :       DIGIT+ ('n')?
        ;
```

```
FLOAT
    :   DIGIT+ '.' DIGIT* EXPONENT?
    |   '.' DIGIT+ EXPONENT?
    |   DIGIT+ EXPONENT
    ;

COMMENT
    :   ('//' ~('\n'|'\r')* '\r'? '\n'
    |   '/*' ( options {greedy=false;} : . )* '*/')
        {$channel=HIDDEN;}
    ;

WS  :   ( ' '
        | '\t'
        | '\r'
        | '\n'
        )+ {$channel=HIDDEN;}
    ;

STRING  :   '"' ( ESC_SEQ | ~('\\'|'"') )* '"'
        |   '\'' ( ESC_SEQ | ~('\\'|'\'') )* '\''
        ;

ERROR_TK        : . ;
```

6 Grammatica decorata

```
grammar JavaScriptToJQueryConverter;
options {
        language = Java;
        k = 1;
}
@header {
        package JS2JQConverter;
}
@lexer::header {
        package JS2JQConverter;
}
@members {
        Handler h;
        public Handler getHandler () {
                return h;
        }
        public void displayRecognitionError(String[] tokenNames,
                                            RecognitionException e) {
            String hdr = " * " + getErrorHeader(e);
            String msg = " - " + getErrorMessage(e, tokenNames);
            Token tk = input.LT(1);
                h.handleError(tk, hdr, msg);
  }
        void initParser () {
                h = new Handler(input);
        }
}
```

```
parseJava
@init {initParser();}
        :
                (instructionRule
                | blockRule
                | classRule)*
        ;

idDotIdRule
        :
                ID (DOT ID)*
        ;

factorTypologyRule
        :
                (o1=STRING
                | o2=INTEGER
                | o3=FLOAT
                | o4=TRUE
                | o5=FALSE
                | o6=idDotArrayRule
                | o7=getRule )
        ;

assignTypologyRule
        :
        (objectRule | arrayRule | NULL | UNDEFINED
        | functionDefinitionRule | newRule)
        ;
```

```
idDotArrayRule
@after {h.checkLastDot($stop);
        h.translateId($start, $stop);
        h.saveVariable($text, $start, $stop, o1, o2);
        h.getAjaxMethod($start, $stop);
        }
        :
                o1=incDecRule?
                (THIS DOT)?
                (idDotIdRule
                        (LB expressionRule RB)* DOT?
                        (LP parametersRule? RP DOT?)?
                )+
                o2=incDecRule?
            {h.checkDuplicateIncDec(o1, o2, $start);}
        ;

parametersRule
        :
                (assignTypologyRule | expressionRule)(CM
            (assignTypologyRule | expressionRule))*
        ;
expressionRule
        :
                termRule ((ADD | SUB | MOD) termRule)*
        ;
termRule
        :
                factorRule ((STAR | DIV | EXP | XOR | AND_BIT | OR_BIT
            | LSHIFT | RSHIFT | URSHIFT) factorRule)*
        ;

factorRule
        :
                factorTypologyRule
                | LP expressionRule RP
        ;
```

```
instructionRule
        :
                (BREAK
                | CONTINUE
                | tryCatchRule
                | functionDefinitionRule
                | documentRule
                | ifStatementRule
                | switchCaseRule
                | forRule
                | whileRule
                | doWhileRule
                | throwRule
                | typeOfRule
                | idStartingRule)SC?
        ;

throwRule
        :
                THROW (STRING | TRUE | FALSE | INTEGER
            | FLOAT | idDotArrayRule)
        ;

typeOfRule
        :
                TYPEOF assignTypologyRule
        ;



tryCatchRule
        :
                TRY blockRule
                CATCH LP ID RP blockRule
                (FINALLY blockRule)?
        ;
```

```
returnRule
        :
                RETURN (expressionRule|assignTypologyRule) SC?
        ;

functionDeclarationRule
        :
        func =FUNCTION name = ID?
        {h.checkFunctionName($name, $func);}
        LP ((assignTypologyRule|expressionRule)
        ((CM (assignTypologyRule|expressionRule))*))? RP
        ;

functionDefinitionRule
        :
                functionDeclarationRule
                LBR
                        instructionRule*
                        returnRule?
                RBR
        ;

arrayRule
        :
                LB
                        (
                        (expressionRule|assignTypologyRule)
                        (CM (expressionRule|assignTypologyRule)?)*
                        )?
                RB
        ;
```

```
objectRule
        :
        LBR
                (
                (ID | STRING)
                CL
                (expressionRule|assignTypologyRule)
                (
                CM
                (ID | STRING)
                CL
                (expressionRule|assignTypologyRule)
                )*
                )?
        RBR
        ;
variableDefinitionRule
        :
            (VAR | LET | CONST)?
            idDotArrayRule
            ((ASSIGN|PLUSEQ|MINUSEQ|STAREQ|DIVEQ|MODEQ|EXPEQ)
            (expressionRule|assignTypologyRule))?

        ;
documentRule
@after{h.translateIdWithAssign($start, $stop);}
        :
            getRule ((ASSIGN|PLUSEQ)
            (expressionRule|assignTypologyRule))?
        ;
getRule
        :
            DOCUMENT DOT get=ID LP ((idDotArrayRule|STRING)
            (ADD (idDotArrayRule | STRING))*) end=RP
            (DOT idDotArrayRule)?
            {h.translateGet($get, $start, $end);}
        ;
```

```
idStartingRule
@after{h.translateIdWithAssign($start, $stop);
        h.searchXMLHttpRequest($start, $stop);
        h.getAjaxAttribute($start, $stop);}
        :
        (VAR | LET | CONST)?
        id=idDotArrayRule
        (((ASSIGN|PLUSEQ|MINUSEQ|STAREQ|DIVEQ|MODEQ|EXPEQ)
        (expressionRule|assignTypologyRule))? | INSTANCEOF ID)
        ;


newRule
        :
            NEW ID LP ((expressionRule|assignTypologyRule)
            (CM(expressionRule|assignTypologyRule))*)? RP
        ;


comparatorRule
        :
            (EQ | NEQ | LT | LE | GT | GE | TEQ | NTEQ)
        ;

conditionRule
        :
            NOT? expressionRule
            (comparatorRule NOT? expressionRule)?
            ((AND|OR) conditionRule)*
        ;
blockRule
        :
                LBR (instructionRule|blockRule)* RBR
        ;
```

```
ifStatementRule
@after{h.searchStatus($start, $stop);}
        :
                IF LP conditionRule RP
                (blockRule | instructionRule)
                (ELSE (IF LP conditionRule RP)?
                (blockRule | instructionRule))*
        ;

switchCaseRule
        :
                SWITCH LP (expressionRule|assignTypologyRule) RP
                LBR
                        (CASE (expressionRule|assignTypologyRule)
                CL instructionRule*
                        )*
                        (DEFAULT CL
                                instructionRule*
                        )?
                        (CASE (expressionRule|assignTypologyRule)
                CL instructionRule*
                        )*
                RBR
        ;

forRule
        :
          FOR LP forInitVarRule? SC conditionRule?
          SC idDotArrayRule? RP
          (blockRule | instructionRule)
        ;

forInitVarRule
        :
                (VAR | LET)? idDotArrayRule ASSIGN
                (expressionRule|assignTypologyRule)
        ;
```

```
incDecRule returns[Token tk]
        :
                o1=DEC      {tk = $o1;}
                |o2=INC     {tk = $o2;}
        ;

whileRule
        :
                WHILE LP conditionRule RP
                        (blockRule | instructionRule)
        ;

doWhileRule
        :
                DO
                        (blockRule | instructionRule)
                WHILE LP conditionRule RP SC?
        ;

classRule
        :
                CLASS ID
                LBR
                        functionDefinitionRule*
                RBR
        ;
```

```
fragment
EXPONENT : ('e'|'E') ('+'|'-')? ('0'..'9')+ ;

fragment
HEX_DIGIT : ('0'..'9'|'a'..'f'|'A'..'F') ;

fragment
ESC_SEQ
    :   '\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\'|'\/')
    |   UNICODE_ESC
    |   OCTAL_ESC
    ;

fragment
OCTAL_ESC
    :   '\\' ('0'..'3') ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7')
    ;

fragment
UNICODE_ESC
    :   '\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
    ;

fragment
LETTER : 'a'..'z'|'A'..'Z';
fragment
DIGIT : '0'..'9';
DOCUMENT
        :'document';

ASSIGN  : '=';
```

```
// comparatori
EQ        : '==';
TEQ       : '===';
NEQ       : '!=';
NTEQ      : '!==';
GT        : '>';
GE        : '>=';
LT        : '<';
LE        : '<=';
QUEST     : '?';

// punteggiatura
AT        : '@';
CL        : ':';
CM        : ',';
DOT       : '.';
SC        : ';';

// parentesi
LP        : '(';            // Parenthesis
RP        : ')';
LB        : '[';            // Brackets
RB        : ']';
LBR       : '{';            // BRaces
RBR       : '}';

// operatori
ADD               : '+';
SUB               : '-';
STAR              : '*';
DIV               : '/';
MOD               : '%';
INC               : '++';
DEC               : '--';
EXP               : '**';
PLUSEQ            : '+=';
MINUSEQ           : '-=';
```

```
STAREQ          : '*=';
DIVEQ           : '/=';
MODEQ           : '%=';
EXPEQ           : '**=';

// operatori logici
NOT             : '!';
AND             : '&&';
OR              : '||';
XOR             : '^';
AND_BIT         :'&';
OR_BIT          :'|';
LSHIFT          : '<<';
RSHIFT          : '>>';
URSHIFT         : '>>>';

// keywords
AWAIT           :'await';
BREAK           :'break';
CASE            :'case';
CATCH           :'catch';
CLASS           :'class';
CONST           :'const';
CONTINUE        :'continue';
DEBUGGER        :'debugger';
DEFAULT         :'default';
DELETE          :'delete';
DO              :'do';
ELSE            :'else';
ENUM            :'enum';
EXPORT          :'export';
EXTENDS         :'extends';
FALSE           :'false';
FINALLY         :'finally';
FOR             :'for';
FUNCTION        :'function';
IF              :'if';
```

```
IMPLEMENTS      :'implements';
IMPORT          :'import';
IN              :'in';
INSTANCEOF      :'instanceof';
INTERFACE       :'interface';
LET             :'let';
NEW             :'new';
NULL            :'null';
PACKAGE         :'package';
PRIVATE         :'private';
PROTECTED       :'protected';
PUBLIC          :'public';
RETURN          :'return';
STATIC          :'static';
SUPER           :'super';
SWITCH          :'switch';
THIS            :'this';
THROW           :'throw';
TRY             :'try';
TRUE            :'true';
TYPEOF          :'typeof';
VAR             :'var';
VOID            :'void';
WHILE           :'while';
WITH            :'with';
YIELD           :'yield';

UNDEFINED       : 'undefined';


ID      :
        ( LETTER |'_' | '$')( LETTER |DIGIT |'_' | '$')*
        ;


INTEGER :       DIGIT+ ('n')?
        ;
```

```
FLOAT
    :   DIGIT+ '.' DIGIT* EXPONENT?
    |   '.' DIGIT+ EXPONENT?
    |   DIGIT+ EXPONENT
    ;

COMMENT
    :   ('//' ~('\n'|'\r')* '\r'? '\n'
    |   '/*' ( options {greedy=false;} : . )* '*/')
        {$channel=HIDDEN;}
    ;

WS  :   ( ' '
        | '\t'
        | '\r'
        | '\n'
        )+ {$channel=HIDDEN;}
    ;

STRING  :   '"' ( ESC_SEQ | ~('\\'|'"') )* '"'
        |   '\'' ( ESC_SEQ | ~('\\'|'\'') )* '\''
        ;

ERROR_TK        : . ;
```