Yass - Documentazione tecnica

Xhorxho Papallazi, Marco Rodolfi, Simone Ronzoni

12 dicembre 2023

1 Introduzione

Yass è un linguaggo di stile che estende il CSS andando ad aggiungere funzionalità e feature quali variabili, cicli, annidamenti e mixin.

1.1 Nomenclatura

Un file CSS è costituito da un insieme di regole, ciascuna costituita da uno o più selettori che specificano gli elementi del DOM e un blocco di proprietà, ciascuna definita tramite il nome della proprietà e il valore che assume.

2 Requisiti

2.0.1 Variabili

E' possibile definire delle variabili da utilizzare sia per specificare nomi di proprietà, valori di proprietà, identificativi per poter essere riutilizzati. Le variabili devono necessariamente essere definite prima del loro utilizzo e la loro tipizzazione è dinamica. Una variabile non la si può riassegnare.

I tipi di dati supportati sono:

- Stringhe
- Liste
- Dizionari

E' inoltre possibile assegnare a una variabile il valore di un'altra variabile

Yass

```
bg-color = "blue";

body .${bg-color} {
  background-color: ${bg-color};
}
```

CSS

```
body .blue {
  background-color: blue;
}
```

2.0.2 Cicli

I cicli permettono di iterare su variabili di tipo lista e dizionario (le uniche iterabili), e per ogni elemento vengono generate le proprietà definite nel corpo del ciclo.

Si permettono due sintassi per definire i cicli: la prima accetta come parametro una variabile iterabile e predispone nello scope delle variabili disponibili nel corpo del ciclo `value` che contiene il valore dell'elemento i-esimo e `index` l'indice.

Yass

```
bg-colors = ["blue", "white",
↪  "green", "yellow", "black"];

$foreach(bg-colors) {
  body .${value} {
    background-color: ${value};
  }
}
```

CSS

```

body .blue {
    background-color: blue;
}

body .white {
    background-color: white;
}

body .green {
    background-color: green;
}

body .yellow {
    background-color: yellow;
}

body .black {
    background-color: black;
}
```

Nella seconda sintassi i parametri da passare sono `$foreach(index, item : list)`

Yass

```
bg-colors = ["blue", "white",
↪  "green", "yellow", "black"];
$foreach(index, color : bg-colors) {
  body .color-${index} {
    background-color: ${color};
  }
}
```

CSS

```
body .color-0 {
    background-color: blue;
}

body .color-1 {
    background-color: white;
}

body .color-2 {
    background-color: green;
}

body .color-3 {
    background-color: yellow;
}

body .color-4 {
    background-color: black;
}
```

Sono supportati cicli annidati e la possibilità di richiamare mixin e variabili definite esternamente.

E' inoltre permesso utilizzare gli annidamenti all'interno di cicli.

2.0.3 Annidamento

Questa funzionalità permette di specificare stili annidati andando a definire all'interno di un blocco CSS altri blocchi CSS. Il risultato sarà la creazione di blocchi CSS con selettori la concatenazione dei selettori padre con quelli del figlio.

E' stato introdotto l'operatore & per riferirsi ai selettori della classe padre, potendo specificare la posizione in cui inserirli, ad esempio, per applicare alla classe padre un modificatore di classe (e.g. `btn` e `btn-primary`) o una pseudo classe (e.g. `:hover`).

Yass

```
body {
  font-size: 1rem;
  font-family: Arial;

  &:hover {
    cursor: pointer;
  }

  h1 {
    font-size: 3rem;
  }
}
```

CSS

```
body {
    font-size: 1rem;
    font-family: Arial;
}

body:hover {
    cursor: pointer;
}

body h1 {
    font-size: 3rem;
}
```

2.0.4 Mixin

I mixin permettono di definire delle proprietà CSS riutilizzabili, permettendo di parametrizzare determinati valori.

Yass

```
button = (bg-color) {
  padding: 12px;
  background-color: ${bg-color};
};

button-primary-color = #2236e1;
button-secondary-color = #0a1043;

button .btn-primary {
  $button(button-primary-color);
  font-size: 1.5rem;
}

button .btn-secondary {
  $button(button-secondary-color);
}
```

CSS

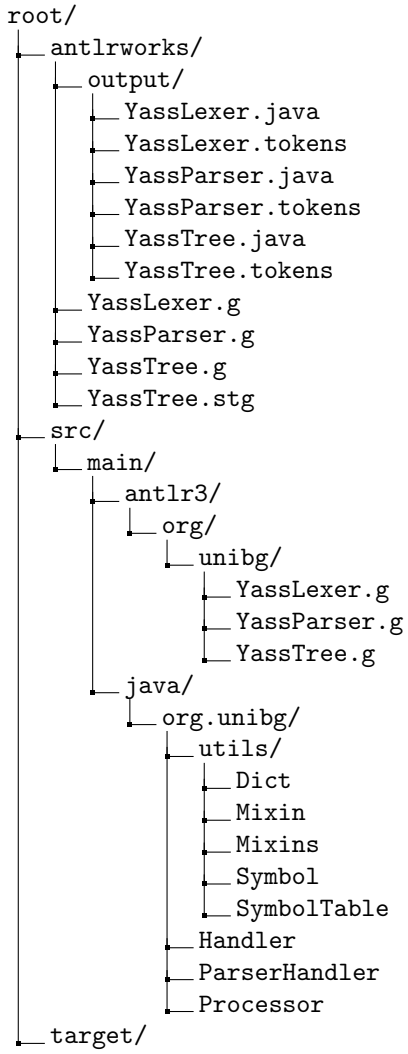
```
button .btn-primary {
  padding: 12px;
  background-color = #2236e1:
  font-size: 1.5rem;
}

button .btn-secondary {
  padding: 12px;
  background-color: #0a1043;
}
```

3 Tecnologie utilizzate

Per la realizzazione di questo linguaggio sono stati utilizzati ANTLR v3 per quanto riguarda il riconoscimento sintattico, semantico e la costruzione dell'albero, con linguaggio di output Java. Per la traduzione dell'albero in CSS si è deciso di integrare codice Java all'interno di ANTLR e non utilizzare le StringTemplate, con limitazioni dovute all'implementazione dei cicli.

4 Struttura del progetto

Il progetto è strutturato basandosi su Maven come archetipo.



Si è deciso di dividere la grammatica di ANTLR in 3 file:

1. `YassLexer.g` che contiene il lexer
2. `YassParser.g` che contiene il parser
3. `YassTree.g` che contiene l'albero della grammatica

Con ANTLR si è creata una rappresentazione intermedia della grammatica sottoforma di Abstract Syntax Tree (AST) che poi è stato visitato per generare la traduzione in CSS.

Nella cartella `antlrworks` sono presenti i file .g di ANTLR, mentre in `output` i token e file Java generati utilizzando ANTLRWorks.

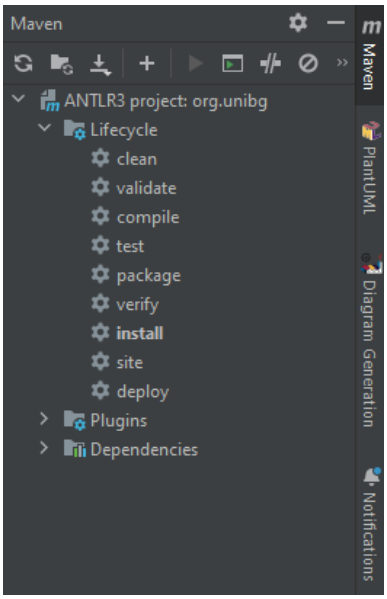
Nella cartella `src` sono contenuti i package e le classi Java.

Nella cartella `target` è contenuto il risultato della compilazione con Maven.

5 Compilazione del progetto

Come ambiente di sviluppo è stato utilizzato IntelliJ IDEA con il plugin per Maven che rende la compilazione molto intuitiva.
Nell'interfaccia di IDEA si può trovare la sezione dedicata al plugin di Maven a destra dello schermo (5). Premendo sopra il pulsante si aprirà l'interfaccia dedicata con le varie operazioni permesse:

- `clean`: elimina la cartella `target`
- `compile`: compila il progetto generando la cartella `target`
- `package`: crea all'interno della cartella `target` due file `jar` che contengono le classi compilate e, in uno dei due, anche le dipendenze del progetto.



6 Dettagli implementativi

Nella classe `Handler.java` è presente il Core dell'applicazione. Nel fare il parsing dell'albero, ad ogni `ruleset` che si incontra viene creato un nuovo `StringBuilder` per la costruzione della stringa dei selettori e uno per le proprietà nel linguaggio tradotto di quel blocco.

La stringa dei selettori è costituita da uno o più selettori separati da una virgola. Per poter permettere l'annidamento, un selettore potrebbe essere costituito dal simbolo `&`. In questo caso, in fase di costruzione della stringa dei selettori, viene fatto l'append del valore ritornato dal metodo `getCurrentSelector()`, ovvero il selettore padre.

Nell'attributo `level` è contenuto il livello di profondità a cui ci si trova, necessario poi per gestire gli annidamenti. Alla chiusura di un blocco, viene fatto il merge della traduzione attuale con i blocchi del livello più alto.

6.1 Variabili

Le variabili sono gestite mediante l'oggetto `SymbolTable`, al cui interno sono memorizzate le variabili in una `HashMap`. In fase di dichiarazione di una variabile, viene creata una nuova entry nella HashMap con chiave il nome della variabile e come valore un oggetto `Symbol` contenente il tipo della variabile (stringa, lista o dizionario) e il rispettivo valore. Prima di inserire una variabile nella `SymbolTable` viene controllato che non sia già stata definita quindi non è permesso cambiare il valore di una variabile.

Per quanto riguarda variabili di tipo stringa, nell'albero è definita la regola `variableInterpolation` che permette l'interpolazione solo di variabili di tipo stringa andando a chiamare il metodo `getVarValue` dell'`Handler` che va a prendere la variabile dalla `SymbolTable`, controlla la correttezza del tipo e ne restituisce il valore.

Per quanto riguarda liste e dizionari, anche questi sono memorizzati nella `SymbolTable`. Quando viene matchata la regola `get` per ottenere l'elemento i-esimo, viene richiamato il metodo `getSpecificValue` con parametri la lista o dizionario e l'indice o chiave. Viene controllato che se l'elemento passato come parametro è una variabile di tipo lista allora il secondo parametro deve essere un valore intero, altrimenti se il primo è un dizionario, il secondo parametro deve essere una stringa.

6.2 Scope dell variabili

Come nella maggior parte dei linguaggi, le variabili definite all'interno di costrutti di iterazione o nelle funzioni, hanno uno scope limitato al solo corpo del ciclo. Tale meccanismo è stato implementato anche in Yass.

Quando viene matchata la regola di cicli e mixin, viene creata una nuova `SymbolTable` con riferimento alla `SymbolTable` padre. In questo modo, nel momento in cui si utilizzano cicli e mixin annidati, per ognuno verrà creato uno scope di variabili. In fase di ricerca del valore, se una determinata variabile non è stata trovata nella `SymbolTable` corrente, verrà ricercata ricorsivamente nelle `SymbolTable` padri finché non si raggiungere la radice.

Questo meccanismo ha permesso la definizione di variabili predefinite all'interno di cicli (come la variabile value per accedere al valore i-esimo) e la definizione di parametri nei mixin senza andar a rovinare eventuali variabili definite esternamente con lo stesso nome.

6.3 Cicli

Esistono due sintassi per definire i cicli.

Primo controllo che viene effettuato è verificare se la variabile passata come argomento è di tipo iterable (lista o dizionario). La logica vera e propria è contenuta nel metodo `foreachBody()`.

Non potendo effettuare cicli durante la visita dell'AST, in quanto impossibile con l'implementazione base del parser dell'albero offerta da ANTLR, viene inizializzato un nuovo `TreeParser` a partire dal sotto albero in modo iterativo.

6.4 Gestione errori

Per la gestione degli errori semantici, è stato predisposto il metodo `handleError()` che prende come parametro l'errore di tipo enum e il token che ha originato l'errore, genera il messaggio di errore e lo aggiunge all'attributo `errorList`.

Gli errori gestiti sono i seguenti:

- `UNDECLARED_VAR_ERROR`: si verifica quando si tenta ad accedere a una variabile non preventivamente inizializzata.
- `DECLARED_VAR_ERROR`: si sta tentando di definire una variabile già definita.

- `NOT_ITERABLE_VAR_ERROR`: la variabile passata come argomento di un ciclo non è di tipo iterable (LIST o MAP).
- `DECLARED_MIXIN_ERROR`: Mixin già dichiarato.
- `UNDECLARED_MIXIN_ERROR`: Mixin non dichiarato.
- `NULL_VAR_ERROR`: La variabile è presente nella `SymbolTable` ma ha valore null.
- `MISMATCH_ARGUMENTS_MIXIN_ERROR`: Il numero di parametri passati ad un mixin non corrispondono a quelli dichiarati.
- `NOT_STRING_VAR_ERROR`: Quando la variabile che si vuole ottenere non ha il tipo corrispondente a quello richiesto.
- `INDEX_OUT_OF_RANGE_ERROR`: Si sta tentando di accedere a un elemento i-esimo di indice superiore alla lunghezza della struttura dati.

7 Diagrammi

7.1 Class diagram