Infra-estrutura de Testes para Implementações de Referência do Standard ECMAScript

Diogo Costa Reis
ist187526
diogo.costa.reis@tecnico.ulisboa.pt

Instituto Superior Técnico
Av. Rovisco Pais, 1
1049-001 Lisboa
Tel: +351 218 417 000
mail@tecnico.ulisboa.pt

Abstract. Keywords: ECMAScript · Specification Language · Reference Interpreters · Test262

Table of Contents

1 Introduction

2 Goals

3 Background

This chapter provides an overview on the ECMAScript standard, the Test262 that are used to test the correct implementation of the ECMAScript standard, and finally an outline of the new metadata generated.

3.1 ECMAScript

JavaScript (JS) is a programming language mainly used in the development of client side web applications, also being one of the most popular programming languages. According to both GitHub and StakeOverflow statistics, JavaScript is one the most active languages on GitHub[1] and the second most active on StackOverflow [2].

JavaScript is specified in the ECMAScript standard[2], which is written in English, and subsequently implemented by various compilers and interpreters. Most common compilers are Hop JavaScript compiler [1], o JSC [4]. Most popular interpreters being nodejs [**?**] and spidermonkey [3]. The standard defines the types, values, objects, properties, syntax, and semantics of JavaScript that must be the same in every JavaScript compiler and interpreter, while allowing to define additional types, values, object, properties, and functions. ECMAScript standard is the official document in which the JavaScript language is defined. This document is in constant evolution, being updated by the ECMA Technical Committee 39 (TC39), which is responsible for maintaining the standard. The standard is currently in its twelfth version.

The JavaScript language can be divided into three major components, those being expressions and commands, built-in libraries, and finally internal functions.

- Expressions and commands describe the behavior of static constructions, detailing the semantics of the diverse expressions (e.g., assignment expressions, built-in operators, etc.), commands (e.g., loop commands, conditions command, etc.), and built-in types (Undefined, Null, Boolean, Number, String and Object).
- The internal functions of the language are used to define the semantics for both expressions and commands, as well as the built-in libraries. Internal functions are not exposed beyond the internal context of the language. In other words, no JavaScript program uses internal functions directly.
- Finally, built-in libraries encompass all the internal objects available when a JavaScript program is executed. Internal objects expose many functions implemented by the language itself, including functions to manipulate numbers, text, arrays, objects, amongst other things.

[1] Most utilized language based GitHub pull requests - https://madnight.github.io/githut/

[2] Tendencies based on the Tags used - https://insights.stackoverflow.com/trends

The figure 1 shows a snippet of the ECMAScript standard description of the if command, that evaluates the command:

```
if (Expression) Statement1 else Statement2
```

the language begins by evaluating the `Expression` storing the result in the vareable `exprRef` (step 1). The previous step will be used as Boolean, therefore, the result of the previous step will be converted to a Boolean using the functions `ToBoolean` and `GetValue`, and having the result stored in the `exprValue` (step 2).A different `Statement` will be followed depending on `exprValue`. If `exprValue` has the value `true` the variable `stmtCompletion` will have the evaluation of the first `Statement` (step 3). Otherwise, if the variable `stmtCompletion` will store the result of evaluating the second `Statement` (step 4). Finally, a `Completion` will be returned, if the `stmtCompletion` has non empty value then it will be returned, however, when the value is empty it will be replaced with undefined (step 5).

IfStatement : **if** (*Expression*) *Statement* **else** *Statement*

1. Let *exprRef* be the result of evaluating *Expression*.
2. Let *exprValue* be ! ToBoolean(? GetValue(*exprRef*)).
3. If *exprValue* is **true**, then
    a. Let *stmtCompletion* be the result of evaluating the first *Statement*.
4. Else,
    a. Let *stmtCompletion* be the result of evaluating the second *Statement*.
5. Return Completion(UpdateEmpty(*stmtCompletion*, **undefined**)).

Fig. 1. ECMAScript definition of an if-else statement

Figure 2 show a snippet of the ECMAScript standard description of the pop function in the Array Built-in. To begin with, the array will be converted to and Object using the `ToObject` function, and stored in the `O` vareable (step 1). Afterwards, the array length of the previously calculated variable will be calculated with the `LengthOfArrayLike` function, and storing the result on the `len` variable (step 2). At this point there are to ways to proceed depending on the value of `len`. If the value is zero then the property `length` of `O` is set to zero and `undefined` is returned (step 3). Otherwise, when `len` is different from zero, the language will assert that it is positive (step 4.a). Afterwards, the `newLen` variable will store the value of `len` decremented by 1 (step 4.b). The variable `index` will store the variable calculated in the previous step represented as a String converted with the `toString` function (step 4.c). Then, storing in the `element` variable the value in the `O` variable at the property corresponding to `index` using the `Get` function (step 4.d). Subsequently, deleting the previously mentioned property from the `O` variable with the `DeletePropertyOrThrow` function (step 4.e). In addition to, setting the `length` property of the `O` variable to the `newLen`

using the `Set` function (step 4.f). Finally, returning the value of the variable `element`.

1. Let *O* be ? ToObject(**this** value).
2. Let *len* be ? LengthOfArrayLike(*O*).
3. If *len* = 0, then
    a. Perform ? Set(*O*, **"length"**, **+0**$_{\mathbb{F}}$, **true**).
    b. Return **undefined**.
4. Else,
    a. Assert: *len* > 0.
    b. Let *newLen* be F(*len* - 1).
    c. Let *index* be ! ToString(*newLen*).
    d. Let *element* be ? Get(*O*, *index*).
    e. Perform ? DeletePropertyOrThrow(*O*, *index*).
    f. Perform ? Set(*O*, **"length"**, *newLen*, **true**).
    g. Return *element*.

Fig. 2. ECMAScript definition of Array.pop

The figure 3 shows a snippet of the ECMAScript standard description of the `LengthOfArrayLike` internal function, that evaluates the function:

```
LengthOfArrayLike (obj)
```

The language starts by asserting that `obj` is an `Object` (step 1). Afterwards, getting the value of the property `length` from `obj` using the function `Get`. Converting the previously mentioned value to an Integer that represents the length with the `ToLength` function, and finally returning said Integer.

1. Assert: Type(*obj*) is Object.
2. Return ℝ(? ToLength(? Get(*obj*, **"length"**))).

Fig. 3. ECMAScript definition of the LengthOfArrayLike

3.2 Test262

Implementing a JavaScript engine is particularly difficult since it involves dealing with the many corner cases explain meaning of corner case that exist

in the language. In addition, while testing JavaScript code, test batteries will also need to test all corner cases to guarantee that they are correctly implemented.

Every test of test262 has 2 parts, those being the frontmatter with some metadata about the test and the second part is the code of the test. As in the example 4.

```
1  // Copyright (c) 2012 Ecma International.  All rights reserved.
2  // This code is governed by the BSD license found in the LICENSE file.
3
4  /*---
5  es5id: 15.4.5-1
6  description: Array instances have [[Class]] set to 'Array'
7  ---*/
8
9  var a = [];
10 var s = Object.prototype.toString.call(a);
11
12 assert.sameValue(s, '[object Array]', 'The value of s is expected to be "[object Array]"');
```

Fig. 4. Test262 es5id: 15.4.5-1

- formato de um teste - as várias secções e o seu próposito
- metadados oficialmente incluídos nos testes - que metadados é que os testes contém actualmente
- quais são os metadados que nós achamos serem relevantes e que estão em falta

3.3 An Infrastructure for testing reference implementations of the ECMAScript standard

- metadados da Test262 -¿ estruturados
- quais são os novos metadados incluídos
- sumário das abordagens utilizadas para os calcular
- shortcomings...

4 Related Work

5 Design and Methodology

6 Evaluation and Planning

7 Conclusion

References

1. Ambiente de programação multi-tier para o javascript, https://github.com/manuel-serrano/hop, acedido a 2020-12-21

2. especificação da linguagem ecmascript® , 6.0 edition / june 2015, http://www.ecma-international.org/ecma-262/6.0/ECMA-262.pdf, acedido a 2020-12-21
3. Spidermonkey is mozilla's javascript engine written in c and c++, https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey, acedido a 2020-12-21
4. Um motor javascript que corre em cima do motor javascript do browser, https://github.com/mbbill/JSC.js, acedido a 2020-12-21