BIOINFORMATICS

Vol. 00 no. 00 2015
Pages 1–5

ECMAScript 6 - Language Features

Corresponding Author [1,*], Co-Author [2] and Co-Author [2*]

[1]Department of XXXXXXX, Address XXXX etc.
[2]Department of XXXXXXXX, Address XXXX etc.

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT

Motivation: ECMAScript is the underlying standard for a cross-platform programming languague usually embedded in HTML Code to add dynamic interactivity to web documents. Almost every Web Developer has heard about this wide spread Language and came to the point using the widely known ECMAScript Implementation "Javscript" for her or his project. As time passes by, the use of JavaScript in practice has shown it's benefits and weaknesses. That makes it important to stay up to date with the current and upcoming language standards for this very potent programming language. In this article we're going to have a closer look at the upcoming JavaScript language standard which is ECMAScript Version 6. After giving some background information about ECMAScript the web app the authors developed for the Seminar "Javascripting Technology" follows. An architectural overview over the applicaton *"ESnake6"* will be given before notable ECMAScript 6 language features will be presented and explained.

Contact: name@bio.com

1 INTRODUCTION

1.1 ECMAScript & JavaScript

ECMAScript is the language specification for JavaScript, standartized by ECMA International. The corresponding standard is the ECMA-262 specification. As ECMAScript is also standartized by the International Organization for Standardization (ISO) in collaboration with the International Electrotechnical Commission (IEC) a ISO/IEC 16262 specification exists. As JavaScript provides additional features than the ones specified in the ECMAScript language standard it can be considered as an ECMAScript variant. Another popular variant of ECMAScript is JScript.

1.1.1 ECMA International ECMA International is a not-for-profit association in Geneva, Switzerland. Their target is to develop and publicate standards and technical reports for information- and communication technology as well as consumer electronics.

1.1.2 Next standard for JavaScript The current standard for JavaScript is ECMAScript 5.1, released in 2011. ECMAScript 6 will be the follow-up standard for the JavaScript programming language. Browser vendors can use it as an orientation for their own JavaScript implementations. Currently there's no official Browser version that suporrts ECMAScript 6, but with the help of ECMAScript Version 6 to Version 5 compilers it can already be used.

1.2 History

In 1996 Netscape submitted JavaScript to ECMA International for standartization.

ES1 - 1997 The first ECMAScript standard was released by ECMA International, named ECMAScript Version 1.

ES2 - 1998 One year after the release of ECMAScript Version 1, editorial changes were made to align ECMAScript with the ISO/IEC international standard 16262.

ES3 - 1999 Modern language features were added in ECMAScript Version 3. Some examples are regular expression support or exception handling with try/catch.

ES4 - (—) After the first set of changes made in ECMAScript Version 3, some members of ECMA International planned a set of breaking changes for Version 4, but there were a lot of disagreements and discussion about backwards compatibility and as no compromise was found, Version 4 was finally abandoned.

ES5 - 2009 After endless debates and discussion, the plan for breaking changes was temporarily freezed and reduced to the clarification of JavaScript ambiguities with the introduction of the strict mode.

ES5.1 - 2011 A second set of editorial changes in ECMAScript History were made to align it again with the ISO/IEC standard 16262.

ES6 - 2015 ECMAScript 6 (aka JS Harmony) is scheduled for june 2015. It will be the most extensive standard since ECMA Script Version 1 was released. Due to the fact, that ES6 just adds syntactical features on top of ESCMAScript 5.1 the backward compatibility is guaranteed. The changes made in ES6 also work as a foundation for the following features planned for ES7.

ES7 - (tbd) In May 2015 ECMAScript Version 7 is in a very early draft state, but already planned and includes among other things the features that were planned for ECMAScript 6, but din't make it into this standard before it reached the final draft state.

*to whom correspondence should be addressed

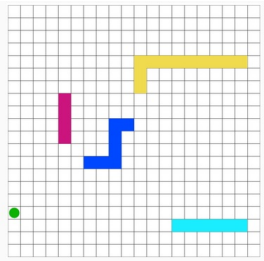

Fig. 1. ECMAScript 6 web app: *ESnake6*

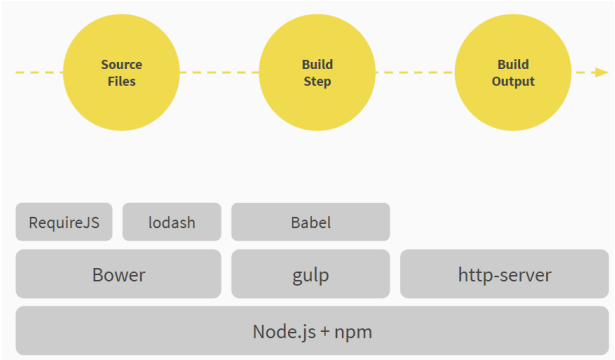

Fig. 2. Libraries and technologies

1.3 Benefits of ECMAScript6

1.3.1 Suitable for large code bases The new ECMAScript Version 6 provides modules an classes which makes the prior used patterns (e.g. module pattern) obsolete and allows better encapsulation. Along with other new features this gives the possibility for clean and easily readable code, which makes ECMAScript6 a good choice for bigger projects. Thanks to the syntax improvements the boilerplate can be reduced which also leads to less lines of code, reduced maintenance effort and easier extensibility, if used properly.

1.3.2 Straight forward migration of existing code As ECMAScript 6 only adds features on top of ECMAScript 5.1 the migration in existing projects is very easy, because there's no necessity to rewrite existing code to make it fit the current standard.

2 WEB APP

2.1 ESnake6

To demonstrate the basics as well as advanced concepts of the ECMAScript 6 programming language, we created a modified multiplayer version of the popular snake game.

2.2 Architectural Overview

2.2.1 Build Process To build ECMAScript 6 applications while the standard is still in a draft state, we used a ES6 to ES5 compiler. After writing the App the ECMAScript Version 6 source files will be translated into ECMAScript 5.1. while the used HTML and CSS files as well as the included libraries will be copied. For this action a build step is necessary. After the translation into ECMAScript 5.1 the project can be run with RequireJS modules and Sourcemaps.

2.2.2 Libraries and Technologies The used technologies in the app include the Bower package manager, which was used to manage client side libraries. The task runner gulp automated the build system. For the sake of testing http-server was used to serve static files. The RequireJS Module system handles asynchronous loading of javascript. The Loadash utility library provides fuctions for working with lists (e.g. map, filter, last). Finally the Babel compiler compiles ECMAScript 6 into ECMAScript 5, which is understood by all modern browsers. This specific compiler accepts a flag which causes the compiler to translate modules to RequireJS syntax.

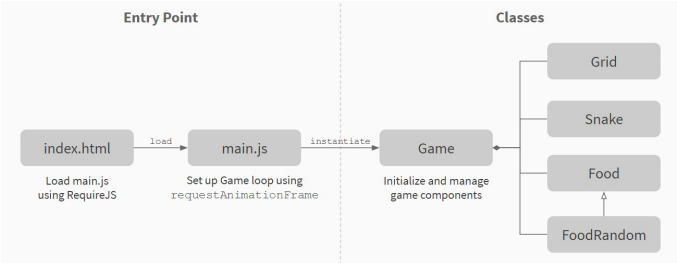

Fig. 3. Applicatoin architecture

2.2.3 Application Architecture The entry point for the application is in the index.html file, which uses RequireJS to load main.js. After setting up the game loop using *requestAnimationFrame* the game class gets instantiated which again instatiates the Grid, Snake and Food class, as well as FoodRandom. To give an example of Inheritance in ECMAScript 6 FoodRandom inherits from Food and appears randomly for a short period of time.

2.2.4 Project Structure The app folder contains the necessary HTML, CSS and JavaScript (ECMAScript6) code. The application entry point is the index.html file, wich loads main.js.

3 ECMA SCRIPT LANGUAGE FEATURES

3.1 Language basics

3.1.1 Const & Let A simple new and useful feature is the "const" keyword, which makes a variable to a constant, which means, that it can only be assigned once. The assignment of a second value will fail silently but can be reognized by Javascript linters. The usage is similar to the usage of var and constants can be defined anywhere a variable can be defined. The naming must differ from functions and names in the same scope.

```
const myConst = 4;
myConst =2; // will fail silently
```

Listing 1. My Javascript Example

The new "'let"' keyword is meant to be a replacement for the currently used "var", which defines variables either globally or in context of the whole codeblock, ignoring block scopes. As it may

lead to unexpected behaviour it's a source for nasty bugs, which is meant to be eliminated with "let". The new keyword declares variable in regard of block scope. Which means that the variable scope expires after leaving the code block it was declared in.

3.1.2 Destructuring Destructuring might remind of pattern matching known from several functional programming languages, which works with function arguments, arrays, also combined with objects and nested data structures.

The feature used on arrays allows the programmer to assign different kinds of values to specific array elements in one step.

```
// arrays
var [a, b] = ["hello", 4];
console.log(a); // => hello
console.log(b); // => 4
```

Listing 2. My Javascript Example

Another example for the use of destructuring on arrays is the standard case of swapping the values of two variables. This is as easily possible as shown in the following example.

```
// Swap variables easily without temp
var a = 1, b = 2;
[b, a] = [a, b];
console.log(a, b); // => 2 1
```

Listing 3. My Javascript Example

```
// objects
var {userId: x} = {userId: 5};
console.log(x); // => 5
```

Listing 4. My Javascript Example

3.1.3 Default, Spread and Rest parameters Default parameters allow the programmer to predefine values for function parameters which will be automatically set if the function is called with less parameters than defined or if the parameters value is "'undefined"'. This feature makes it much easier to make sure function parameters got reasonable values without necessity to use if blocks to do so.

```
// Game.js
function createGame(canvas, cellsX = 20, cellsY =
    20, playerCount = 4) {    }

// main.js
const context = canvas.getContext("2d");
const game = createGame(canvas, 40, 40); // sets
    cellsX and cellsY to 40, playerCount defaults
    to 4
```

Listing 5. My Javascript Example

Rest parameters are indicated by "..." followed by the parameter name. As you can see in the following example the parameters of "testRest" that follow x get stored in an array called y. This makes it easier and cleaner to deal with functions without a specified number of parameters.

```
// put the rest of all function arguments into an
    array
function testRest(x, ...y) {
          return x * y.length; // y is an Array
  }
  testRest(3, "hello", true) == 6
```

Listing 6. My Javascript Example

The spread feature is also indicated by "..." followed by an array passed to a function. It causes the array to be spread into single elements and assigned to the function parameters ordered by their appearance in the parameter list and by the array index (first array element gets assigned to the first parameter, the second one to the second parameter and so on).

```
// turn array into argument list and call function
function testSpread(x, y, z) {
          return x + y + z;
  }
  testSpread(...[1,2,3]) == 6 // Pass each elem of
       array as argument
```

Listing 7. My Javascript Example

3.1.4 Template Strings This feature is also often referred to as string interpolation. The concatenation of texts and variables especially over several lines with double quotes and plus signs was not only difficult to read but also inconvenient to write. Both characteristics made dynamic strings to another source of bugs. In ECMAScript 6 template strings are opened and closed by back ticks and can contain placeholders which use the ${}, they also may go over several lines.

```
// classes/Snake.js
alert(`${this.name} collided with ${player.name}`)
    ; // use back ticks for string interpolation

// ES5
alert(this.name +     collided with     + player.
    name);
```

Listing 8. My Javascript Example

3.1.5 Arrow Functions An important benefit of Arrow functions is the improved scope of "this". In Line 9 of Listing 9 the this operator contains the this value of the enclosing context. In this case the Snake class calls the "draw" function and so the "this" in Line 8 of Listing 9 refers to the same context as Line 9. In EcmaScript 5 it would be necessary to safe the value of "this" in another variable usually called "that" or "self". If the "this" scope wasn't considered correctly its usage in context of nested functions was a big source of bugs. It has to be emphasised that the "this" scoping has only changed in the context of arrow functions to maintain backward compatibility.

```
// Game.js
this.players.forEach(player => player.update());
// ES5
this.players.forEach(function(player) { return
    player.update(); });

// classes/Snake.js
draw(ctx) {
  this.positions.forEach(pos => { // ES5: function
      (pos) { ...
    const { x: pixPosX, y: pixPosY } = this.grid.
        cellPosToPixelPos(pos);
```

```
    ctx.fillStyle = this.color;
    ctx.fillRect(pixPosX, pixPosY, this.grid.
        cellWidth, this.grid.cellHeight);
  });
}
```

Listing 9. My Javascript Example

```
ivar obj = {
    // __proto__
    __proto__: theProtoObj,
    // Shorthand for  handler: handler
    handler,
    // Methods
    toString() {
     // Super calls
     return "d " + super.toString();
    },
    // Computed (dynamic) property names
    [ "prop_" + (() => 42)() ]: 42
};
```

Listing 10. My Javascript Example

3.2 Code organization

A very important part of the new ECMAScript 6 standard is the enhanced code organization in modules and classes. These structures are already known in ECMAScript 5.1 but require patterns to simulate them, which produces a lot of boilerplate, which makes it more difficult to organize bigger code bases without the help of additional tools or libraries.

3.2.1 Modules (Import & Export) The Module System allows the import and export of functions, variables and whole modules itself. Syntactically ”export” is used for exporting and ”import... from” for importing. An example can be found in Listing 11 where the function ”getRandomInt” in the file ”utils.js” gets exported, and imported in ”Grid.js”. A native Module system makes the use of the ”module pattern” obsolete which makes the use of modules cleaner.

```
// utils.js
export function getRandomInt(min, max) {     }

// classes/Grid.js
import { getRandomInt } from '../utils';
getRandomCell() {
  return {
    x: getRandomInt(0, this.cellsX - 1),
    y: getRandomInt(0, this.cellsY - 1)
  };
}
```

Listing 11. My Javascript Example

3.2.2 Classes ECMAScript 6 classes are ”syntactical sugar” over the objects and prototypes which are currently used in ECMAScript 5.1. Listing 12 shows the basic structure of classes in ECMAScript 6. The instantiation of new objects uses the ”new” operator.

```
// classes/Food.js
export default class Food {
  constructor(game, grid, color, respawnInterval =
      5000) {
    this.game = game;
    this.grid = grid;
    ...
  }

  resetRespawn() { ... }
  respawn() { ... }
  static myStaticMethod() { ... }
}

// Game.js
this.food = new Food(this, this.grid, 'green',
    8000);
```

Listing 12. My Javascript Example

3.2.3 Inheritance ECMAScript 6 classes create the same prototype chains known from previous versions of JavaScript, but with a much cleaner syntax. The keyword ”extends” is used to inherit from the super class.

```
export default class FoodRandom extends Food {
  constructor(game, grid, color,
      respawnProbability, respawnInterval = 5000)
      {
    super(game, grid, color, respawnInterval);
    this.respawnProbability = respawnProbability;
  }

  respawn() {
    if (Math.random() < this.respawnProbability) {
      super.respawn();
    } else {
      this.position = { x: -1, y: -1 }; // hide
          the food
      super.resetRespawn();
    }
  }
}
```

Listing 13. My Javascript Example

4 CONCLUSION

- ECMAScript 6 adds on top of ECMAScript 5 and thus maintains backwards compatibility which makes it easy to extend existing projects with new ECMAScript 6 code.
- Many syntax improvements enable the programmer to reduce boilerplate and reduce maintenance efforts.
- Thanks to Modules and Classes code organization gets a lot easier and makes ECMAScript 6 suitable for bigger projects.

To use ECMAScript 6 today, you can use an ES6 to ES5 compiler e.g. Babel or Traceur. The development experience gets very pleasant thanks to sourcemaps. For new runtime features you can also use polyfills (e.g. core-js).

REFERENCES

[1]Zakas, N. C. (2009) Professional javascript for web developers. John Wiley & Sons.

[Learl ES6]Learn ES6 - Babel. (n.d.). Retrieved May 8, 2015, from `https://babeljs.io/docs/learn-es6/`

[2]Bofelli,F., Name2, Name3 (2003) Article title, *Journal Name*, **199**, 133-154.

[3]Bag,M., Name2, Name3 (2001) Article title, *Journal Name*, **99**, 33-54.

[4]Yoo,M.S. *et al*. (2003) Oxidative stress regulated genes in nigral dopaminergic neurnol cell: correlation with the known pathology in Parkinson's disease. *Brain Res. Mol. Brain Res.*, **110**(Suppl. 1), 76–84.

[5]Lehmann,E.L. (1986) Chapter title. *Book Title*. Vol. 1, 2nd edn. Springer-Verlag, New York.

[6]Crenshaw, B.,III, and Jones, W.B.,Jr (2003) The future of clinical cancer management: one tumor, one chip. *Bioinformatics*, doi:10.1093/bioinformatics/btn000.

[7]Auhtor,A.B. *et al*. (2000) Chapter title. In Smith, A.C. (ed.), *Book Title*, 2nd edn. Publisher, Location, Vol. 1, pp. ???–???.

[8]Bardet, G. (1920) Sur un syndrome d'obesite infantile avec polydactylie et retinite pigmentaire (contribution a l'etude des formes cliniques de l'obesite hypophysaire). PhD Thesis, name of institution, Paris, France.