BIOINFORMATICS

Vol. 00 no. 00 2015
Pages 1–3

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 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 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 making clear

Results:

Availability:

Contact: name@bio.com

0.1 What is ECMA Script

0.1.1 ECMA International ECMA International is a not-for-profit association in Geneva. Their target is to develop and publicate standads and technical reports for information and communication technology and consumer electronics. The corresponding standard is ECMA-262 specification and ISO/IEC 16262

0.1.2 Next standard for JavaScript ECMAScript 6 was standartizede by ECMA International and is the next standard for Javascript. The actual version currently used for JavaScript is ECMAScript5.1 Browser vendors can use it as an orientation for their own JavaScript implementations.

0.2 History

ES1 - 1997 First ECMAScript standard 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 and 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, after a huge discussion about backwards compatibility Version 4 was finally abandoned.

ES5 - 2009 After endless debates and discussion, the plan for breaking changes was temporarily freezzed 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 theISO/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 syntax features on top of ESCMAScript 5.1 the backward compatibility is guaranteed. The major changes made in ES6 also work as a foundation for the following features planned for ES7

ES7 - (tbd) At May 2015 ECMAScript Version 7 is in a very early draft state.

0.3 Benefits of ECMAScript6

0.3.1 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 gives the possibility for clean easily readable code which makes ES6 suitable for bigger projects. Thanks to the syntax improvements the boilerplate is reduced which leads to less lines of code and with proper usage to reduced maintenance effort and easier extensibility.

0.3.2 Straight forward migration of existing code As ES6 only adds new features on top of ECMAScript 5.1 the migration of existing code is very easy, wich means that no code must be rewritten to make the code base fit for the new standard.

1 WEB APP

1.1 ESnake6

1.2 Architectural Overview

1.2.1 Build Process To build ECMAScript 6 applications while the standard is still in a draft state, we used a ES6 to ES5 compiler.

*to whom correspondence should be addressed

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. After the translation into ECMAScript 5.1 the project can be run with RequireJS modules and Sourcemaps

1.2.2 Libraries and Technologies The used technologies in the app include the Bower package manager, which was used to mange 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.

1.2.3 Application Strucutre

1.2.4 Project Structure The app folder contains the nescessary HTML, CSS and JavaScript (ECMAScript6) code. The application entry point is the

2 ECMA SCRIPT LANGUAGE FEATURES

2.1 Language basics

2.1.1 Const & Let A simple new and useful features is the "'const"' keyword, which makes a variable to a constant, which means, that it can only be assigned once. The assignement of a second value will fail silently but can be reognized by Javascript linters.

```
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.

2.1.2 Destructuring Destructuring might remind of pattern matching known from several functional programming languages. The feature used on arrays allows the programmer to assign different kinds of values to specific array elements.

```
// 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

2.1.3 Default, Rest, Spread 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 nescessarity 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.

```
// 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

Spreading of parameters is also indicated by "..." followed by an array.

```
// 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

```
// 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

```
// 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

2.2 Code organization

3 DISCUSSION

As it's possible to extend existing projects with ECMAScript 6 code, the question comes up if a mixture of known concepts like the Module Pattern and the new Module System doesn't create confusion.

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).

ACKNOWLEDGEMENT

Text Text Text Text Text Text Text Text. Bofelli *et al*., 2000 might want to know about text text text text

Funding: Text Text Text Text Text Text Text Text.

REFERENCES

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

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

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.

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

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.

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

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.