1.– 4. September 2014
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Die ganze Wahrheit

Ecmascript 6

Lars Gersmann

Content Management AG

ECMAScript 6 – the future is now

About me

Lars Gersmann

Senior Software Developer : Content Management AG

AngularJS, Node.js, Gulp, ES6, HTML5 / Less.js, Sass / Docker / Bleeding edge browser technologies

ECMAScript 6 – the future is now

The big picture

Language enhancements

Modules

How can I use it today ?

Language enhancements

`let` statement

block scoped variant of `var`

```
let i='alpha';
...
for (let i=0; i<10; i++) {
  ...
}
  // i === 'alpha'
...
if (i==='alpha') {
  let i=10;
  ...
}
  // i == 'alpha'

/* CAVEAT : no hoisting characteristics ! */
if(condition) {
  console.log(value);     // ReferenceError!
  let value = "blue";
}
```

Language enhancements

`const` statement

constants must be initialized at declaration

```
const MAX_ITEMS = 30;
const NAME;             // Syntax error: missing initialization
```

constants are block scoped

```
if (condition) {
  const MAX_ITEMS = 5;
    ...
}
```

Language enhancements

Default parameters

```
function render(obj, options=getDefaults()) {
  // es5 : options = options || getDefaults();
  ...
}
...
render(myObj);                              // -> render( myObj, getDefaultRenderOptions())
```

caveat : `undefined` will be replaced by default parameter

```
function render (obj, options=getDefaults(), callback) {
  ...
}

render(myObj, undefined, myCallback);     // -> render( myObj, getDefaults(), myCallback)
```

Language enhancements

Rest parameters

Rest parameter is an true `Array` instead of `arguments`

```
function foo(first, ...rest) {
  return rest.concat( 'last');
}

console.log( foo(1, 2, 3));              // -> [2, 3, 'last']
```

caveat : no other named arguments can follow in function declaration

```
function foo(first, ...numbers, last) {
    // -> Syntax error: Can't have a named parameter after rest parameters
```

Language enhancements

... Spread operator

```
var v = [25, 50], max;

max = Math.max( ...v);                          // es5 : Math.max( v[0], v[1]);
max = Math.max( 17, ...v, 27);                  // es5 : Math.max( 17, v[0], v[1], 27);
array2.push( ...v);                             // es5 : array2.push( v[0], v[1]);
```

Tip : convert array like structures into a true `Array` on the fly

```
[...document.querySelectorAll( "div.big")].forEach(  // jQuery( 'div.big')
   elm=>elm.classList.remove( 'big')                 //   .removeClass( 'big')
);
```

Language enhancements

Object literal improvements

Method initializer shorthand

```
function createPerson( name, age) {
  return {
    name,                                          // es5 : name : name,
    familyName,                                    // es5 : familyName : familyName,

    fullName() {                                   // es5 : fullName : function() {
      return this.name + ' ' + this.familyName;
    }
};
```

Compound property names

```
var property = "foo", value="bar";

var obj = {                                        // es5 : var obj = {};
  [property + '_bar'] : value                      // obj[ property + '_bar'] = value;
}
```

Language enhancements

Destructuring assignment

syntactic sugar for `Array`'s

```
[a,b] = [b,a];                                          // es5 : let tmp=a; a=b; b=tmp;

let [x, y] = f();                                       // -> f() returns an array[2]
console.log('x=',x,', y=' + y);

  // es5 : var match = /(\d+)-(d+)-(\d+)/.match( '2014-6-30');
  // var year = match[1], month = match[2], day = match[3];
let [, year, month, day] = /(\d+)-(\d+)-(\d+)/.match( '2014-6-30');
```

syntactic sugar for `Object`'s

```
var {protocol,hostname, port}  = document.location;          // direct mapping

var {hostname : server, port} = document.location;           // mapped to { server, port}
```

Language enhancements

`hello ${name}` Template strings

featuring multiline strings and expression substitution

```
let url="http://example.com";
let decoration = { /* target : "_blank", */ label : "example" };

let anchor = `<a url="${url}" target="${decoration.target || 'blank'}">
  ${decoration.label}
</a>`;
```

Tag can be used for custom DSL building (real world example : XRegExp - multiline regular expression definition)

```
function safehtml( literals, ...values) {
  // process evaluated literals and values an return a safe html encoded string
  ...
}

let anchor = safehtml`<a url="${url}" target="${decoration.target || 'blank'}">
  ${decoration.label}
</a>`;
```

Language enhancements

Arrow functions

```
jQuery( "button").on( "click", event => console.log( event.target));

let base = () => document.location.href;     // no arguments require parenthesis
console.log( base());

let toUpper = s => s.toUpperCase();          // single argument require doesnt require parenthesis
console.log( toUpper( "huhu"));              // "HUHU"

var values = [9,7,1,5];                      // multiple arguments require parenthesis
values.sort( (a, b) => a - b);               // => [1,5,7,9];

values.sort( (a, b) => {                     // traditional styled arrow function with return statement
  return a - b;
});
```

**Implicit `this` binding to definition scope. `call()`, `apply()`, `bind()` will have no effect. Arrow functions cannot be used with `new`.*

Language enhancements

`for( ...of ...)` statement

`Array`, `Iterator`/`Iterable` support

```
for (let v of [1,3,5,7]) {
  ...

for (let [index, value] of ["a", "b", "c"].entries()) {
  ...
```

`Array`, `Iterator`/`Iterable` comprehension

```
let squared    = [for (x of [1, 2, 3]) x * x];            // -> [1,4,9]
let squaredOdd = [for (x of [1, 2, 3]) if(x % 2) x * x];  // -> [1,9]

let v = [for ([index, value] of ["a", "b", "c"].entries())
    [value.toUpperCase(), index]
];                                                        // v = [["A",0],["B",1],["C",2]]
```

Language enhancements

`function*` Generator

A Generator is a function `yield`ing an unlimited amount of values wrapped in a *automagic* `Iterator` instance.

```
function* generate() {
  let i=0;
  while(i<5) {
    yield i++;
  }
}

for (let v of generate()) {                // using for...of statement
  console.log(v);                          // -> 0,1,2,3,4
}

let iter = generate(), next;               // using while statement
while (!(next=iter.next()).done) {
  console.log(next.value);                 // -> 0,1,2,3,4
}
```

Language enhancements

`function*` Generator

`yield*` delegates control to another `Generator`

```
function* walk(node) {
  for (let child of node.children()) {
    yield* walk(child);
  }
  yield node;
}

for(let n of walk(root)) {
  console.log( n);
}
```

Language enhancements

`function*` Generator

Passing arguments between consumer and `Iterator` around

```
function* summarize( sum = 0) {
  while( true) {
    sum += yield sum;
    console.log( sum);
  }
}

let total = summarize( 10);
total.next();
for( let v of [1,3,6]) {
  total.next( v);
}

    // "11", "14", "20"
```

Language enhancements

`function*` Generator

Bonus : use `yield*` to execute async functions sequential !

```
function spawn( gen) {
  var iter=gen( msg => iter.next( msg.toUpperCase()));

  iter.next();                                          // resume (aka start)
}

spawn(function* (resume) {
  console.log("before");
  console.log(yield setTimeout(()=>resume("hello"), 1000)); // suspend
  console.log(yield setTimeout(()=>resume("world"), 1000)); // suspend
  console.log("after");
});

// "before"
// "HELLO"
// "WORLD"
// "after"
```

Language enhancements

Classes

```
class Position {
  constructor (x, y) {
    this.data = {x, y};
  }

  set x(x) {
    this.data.x = x;
  }

  get x() {
    return this.data.x;
  }

  toString() {
    return `Position(=${this.x},${this.data.y})`;
  }
  ...
}

let p = new Position(0, 20);
p.x = 15;
console.log( `p is ${p}`);
  // "p is Position(=15,20)"
...
```

Language enhancements

Classes

Inheritance, `super` access

```
...
class Shape extends Position {
  constructor (x, y, width, height) {
    super (x, y);
    Object.assign(this.data, {width, height});
  }

  get dimension() {
    return {width : this.data.width, height : this.data.height};
  }

  toString() {
    return `Shape(=${super.toString()}, ${this.dimension.width}, ${this.dimension.height})`;
  }
}

let s = new Shape( 0, 20, 100, 70});
s.x = 15; s.width = 50;
console.log( `s is ${s}`);                                  // "s is Shape(=Position(=15,20), 50, 70)"
console.log( s instanceof Position, s instanceof Shape);    // "true true"
```

Modules

import / export

Modules allows the definition of code with explicit `export`s, `import` specific exported names from those modules, and keep these name
separate.

```
// lib/foo.js
export default class Foo { ... }

export const VERSION = '1.0';
export var util = {
  escape( s) { ... }
};
```

```
// lib/bar.js
export function bar() { ... }
export const VERSION = '1.2';
```

```
import Foo from './lib/foo';                          // import default export
import {util, VERSION as UTIL_VERSION} from './lib/foo'; // import util as util, VERSION as UTIL_VERSIO
import {bar, VERSION as BAR_VERSION} from './lib/bar';

console.log(UTIL_VERSION, BAR_VERSION);               // "1.0 1.2"
util.escape("hello world");
```

Modules

Module Loader (`System.*`)

ES6 defines a module loader interface for loading scripts dynamically (in the browser)

```
<script>
  Promise.all([System.import('./lib/foo'), System.import('./lib/bar')])
  .then(function([foo, bar]) {
    var obj = new foo.Foo();

    console.log(foo.VERSION, bar.VERSION);
      // "1.0 1.2"

    foo.util.escape("hello world");
  }).catch( err) {
    throw e;
  });
</script>
```

Can I use it today ?

Most modern browsers supports ES6 partially (see ECMAScript 6 compatibility table)

NodeJS supports many ES6 features (`--harmony` flag)

es6-shim provides ES6 API for legacy JavaScript engines

Native supported ES6 features growing monthly

Can I use it today ?

YES, You Can !

Traceur is a JavaScript.next-to-JavaScript-of-today compiler
that allows you to use future Ecmascript features today.

Traceur transpiles most* ES6 + some of ES7 features into today's Javascript (ES5).

** except of non shimmable/transpilable features like `Proxy`, `Object.observe()`, new `RegExp` flags ...*

Traceur ES6 transpiler

Tooling

transpiles ES6 sourcecode to ES5
can be used as commandline tool, Grunt task or GulpJS task :

```
var gulp = require('gulp'), traceur = require('gulp-traceur');

gulp.task('default', function () {
  return gulp.src('src/app.js').pipe(traceur({sourceMap: true})).pipe(gulp.dest('dist'));
});
```

supports AMD, CommonJS, SystemJS or all-in-one ES5 code as output. Sourcemap support for easy debugging. provides new ES6 API's (`Set`, `Map`, `Promise` etc.). Compile ES6 on the fly.

Testing ES6 today is easy using Mocha/Jasmine and Karma/Protractor :

```
import NestedMap from "../../src/core/nestedmap";

describe("NestedMap",() => {
  it("constructor",() => {
    let nm = new NestedMap();
    expect(nm.parent).toBeNull();

    let _nm = new NestedMap(nm);
    expect(_nm.parent).toBe(nm);
  });
});
```

Traceur ES6 transpiler

NodeJS

```
$ npm install traceur
```

make Traceur the default JS loader for NodeJS

```
// test.es6
import {b} from './resources/b.es6';
console.log(b);

// resources/b.es6
export var b = 'BBB';

// app.js
var traceur = require('traceur');
traceur.require.makeDefault(function(filename) {
  return filename.endsWith('.es6');
});
require('./test.es6');
```

Traceur ES6 transpiler

Browser

In-browser transpilation

```
<script src="https://traceur-compiler.googlecode.com/git/bin/traceur.js" type="text/javascript"></script>
<script src="https://traceur-compiler.googlecode.com/git/src/bootstrap.js" type="text/javascript"></scrip
<script> traceur.options.experimental = true;</script>
<script type="module">
  class Greeter {
    constructor(message) {
      this.message = message;
    }

    greet() {
      document.querySelector('#message').innerHTML = this.message;
    }
  };

  let greeter = new Greeter('Hello, world!');
  greeter.greet();
</script>
```

or using already transpiled files :

```
<script src="bin/traceur-runtime.js"></script>
<script src="out/greeter.js"></script>
```

Should I use ES6 today ?

PRO

small standard based vanilla code

much improved readability will result in **better productivity**

less legacy dependencies (like jQuery, Underscore etc.) making your code more **future proof**

less code rewrites over the years

huge performance speedup when ES6 will be arrived natively at browsers

ES6 modules simplify larger applications sourece code management

ES6 adaption in browsers and NodeJS is **increasing monthly**

CONTRA

using ES6 today in legacy JS engines makes your toolchain more complicated

Traceur is just a *workaround* : transpiled code is larger and slower than it's ES6 source. Native ES6 language features will not be used in generated code

ES6 is a huge step requiring learning to use new language features the right way

Should I use ES6 today ?

My experience

If your toolchain is well done (GulpJS, gulp-watch, Traceur, Jasmine/Karma for testing) **ES6 is a pleasure to use**.

Learning curve is steep.

Traceur is getting very stable.

My guess

ES6 will be adapted by most browsers until 2016 - including Internet Explorer.

NodeJS will support ES6 mostly in 2014. ES6 features will be adapted very fast by popular NodeJS modules (see Koa as an good ES6 example) because of its massive simplifying async features.

Modern browser libraries like AngularJS 2.0 will be based on ES6 (not a guess - reality !)

My advise

Start learning !

Use it in non critical/inhouse applications/modules to get experienced.

Be prepared when everybody supports ES6.

What I did'nt told you ...

New classes

Symbol, Set, Map, Promise, Iterator ...

Object enhancements

String, Object, Array, ...

Questions ?

FINI

Thank you !!

Slides / Updates will be available at orangevolt.blogspot.com

Lars Gersmann (lars.gersmann@gmail.com)

Github : https://github.com/lgersman

Content Management AG (cm4all.com)