ECMAScript 6의 새로운 것들!

2016.07.27

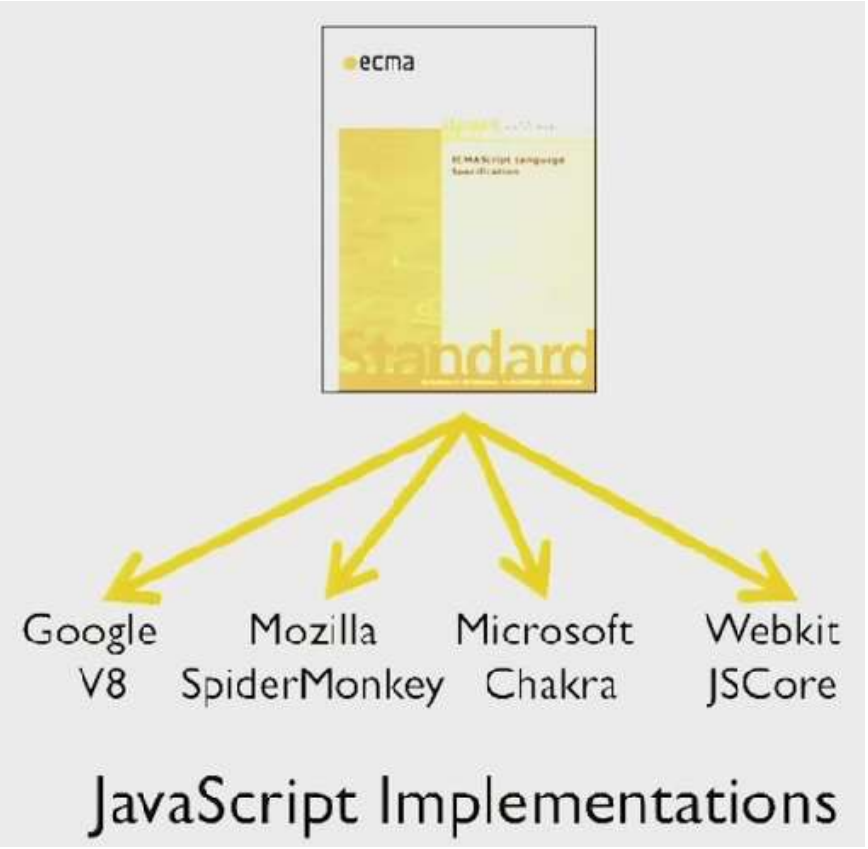
조우영

CONTENTS

1. 에크... 뭐라고요?

What is ECMAScript?

- **ECMAScript는 JavaScript 프로그래밍 언어를 정의하는 국제 표준의 이름**
  - JavaScript는 현재 Oracle의 상표
- **Ecma International의 Technical Committee 39(TC39)에서 논의**
- **ECMA-262 및 ISO/IEC 16262로 발행**
- **W3C와는 별개**
- **Ecma International**
  - 표준화 기구
    - ECMA-262: ECMAScript 언어 규격
    - ECMA-334: C# 언어 규격
    - ECMA-388: OOXML 규격



ECMAScript 6: A Better JavaScript for the Ambient Computing Era

초기의 JavaScript 역사

- 1995년 5월, 넷스케이프의 Brendan Eich씨가 열흘만에 만듦: “Mocha”
- 1995년 9월, 넷스케이프 네비게이터 2.0 베타에 탑재: “LiveScript”
- 1995년 12월, 넷스케이프 2.0b3: “JavaScript”
- 1996년 8월, Microsoft에서 복제하여 IE 3.0에 탑재: “JScript”
- 1996-1997, ECMA-262 1판에서 표준화: “ECMAScript” 또는 “ES1”
- 1999, ES3 – 현대적인 JS의 기반

ECMAScript 6: A Better JavaScript for the Ambient Computing Era

ECMAScript: 방황의 시기

- 2000: ES4, 첫 번째 시도
- 2003-4: E4X, ECMAScript를 위한 XML 확장
- 2005-7: ES4, 두 번째 시도
- 2008: ES4 버려짐
- 2009: ES5, “use strict”, JSON, Object.create, accessors, etc
- 2011: ES5.1, 오류 수정. ISO/IEC 16262와 함께 출판

ECMAScript 6: A Better JavaScript for the Ambient Computing Era

ECMAScript 2015?

- **2015년 6월: ES6 = ES2015, 대규모 수정!! (258p→566p)**
  - ES6 = ECMAScript 6 = ECMAScript 2015 = ES2015
  - ECMAScript 2015가 더 공식적인 이름
  - 아직은 ES6가 더 많이 사용됨
  - ES6부터 TC39는 매년 ECMAScript 표준을 업데이트하기로 결정함
- **2016년 6월: ES7 = ES2016 발표됨 (586p)**
  - 버그 수정
  - Array.prototype.includes
  - 거듭제곱 연산자(**)

TC39가 ES6에서 목표한 것들

- **Interoperability!!!!!**
  - 기존 스펙과의 하위 호환성
  - 기존의 웹페이지 및 JS 코드
    - 스펙과 맞지 않는 코드도 많음
  - 알고리즘까지 포함하는 매우 자세한 스펙
  - 매우 많은 test suite
- **Modularity**
- **Better Abstraction Capability**
  - Better functional programming support
  - Better OO support
- **Expressiveness and Clarity**
  - 예전의 “가능한 실행 오류 없이 진행한다” 정책은 사라지고 있음
- **Better Compilation Target**
  - 현재 JS는 플랫폼 측면에서 독보적인 위치에 있음
  - 다른 프로그래밍 언어를 JS로 컴파일 가능하도록

ECMAScript 6: A Better JavaScript for the Ambient Computing Era

TC39의 모습!

ECMAScript 6: A Better JavaScript for the Ambient Computing Era

2. ES6의 새로운 기능들

- Arrow Functions
- Classes
- Enhanced Object Literals
- Template Literals
- Destructuring (Pattern Matching)
- Default, Rest, Spread Parameters
- Let, Const
- Iterator, For-Of
- Generators
- Promises
- Modules
- Map, Set, WeakMap, WeakSet
- Symbols
- Math, Number, String, Array, Object APIs
- Proxy, Reflect
- 기타

Arrow Functions

- Function expression 대체
- 오른쪽이 expr인 경우 해당 값 리턴
- Block인 경우 return문 필요
- 바깥쪽의 this를 그대로 사용

ES5

```
odds  = evens.map(function (v) { return v + 1; });
pairs = evens.map(function (v) { return { even: v, odd: v + 1 }; });
nums  = evens.map(function (v, i) { return v + i; });
```

ES6

```
odds  = evens.map(v => v + 1);
pairs = evens.map(v => ({ even: v, odd: v + 1 }));
nums  = evens.map((v, i) => v + i);
```

Arrow Functions

- Function expression 대체
- 오른쪽이 expr인 경우 해당 값 리턴
- Block인 경우 return문 필요
- 바깥쪽의 this를 그대로 사용

```
// Expression bodies
var odds = evens.map(v => v + 1);
var nums = evens.map((v, i) => v + i);
var pairs = evens.map(v => ({even: v, odd: v + 1}));

// Statement bodies
nums.forEach(v => {
  if (v % 5 === 0)
    fives.push(v);
});

// Lexical this
var bob = {
  _name: "Bob",
  _friends: [],
  printFriends() {
    this._friends.forEach(f =>
      console.log(this._name + " knows " + f));
  }
}
```

Classes

- 기존 Prototype 기반 OO의 sugar

ES5

```
var Rectangle = function (id, x, y, width, height) {
    Shape.call(this, id, x, y);
    this.width  = width;
    this.height = height;
};
Rectangle.prototype = Object.create(Shape.prototype);
Rectangle.prototype.constructor = Rectangle;
var Circle = function (id, x, y, radius) {
    Shape.call(this, id, x, y);
    this.radius = radius;
};
Circle.prototype = Object.create(Shape.prototype);
Circle.prototype.constructor = Circle;
```

ES6

```
class Rectangle extends Shape {
    constructor (id, x, y, width, height) {
        super(id, x, y);
        this.width  = width;
        this.height = height;
    }
}
class Circle extends Shape {
    constructor (id, x, y, radius) {
        super(id, x, y);
        this.radius = radius;
    }
}
```

Classes

- 기존 Prototype 기반 OO의 sugar
- Constructor 메소드
- Base class 접근 (super)
- Static 메소드
- Getter/setter
- Class expression 가능

```
class SkinnedMesh extends THREE.Mesh {
  constructor(geometry, materials) {
    super(geometry, materials);

    this.idMatrix = SkinnedMesh.defaultMatrix();
    this.bones = [];
    this.boneMatrices = [];
    //...
  }
  update(camera) {
    //...
    super.update();
  }
  get boneCount() {
    return this.bones.length;
  }
  set matrixType(matrixType) {
    this.idMatrix = SkinnedMesh[matrixType]();
  }
  static defaultMatrix() {
    return new THREE.Matrix4();
  }
}
```

향상된 Object Literals

- __proto__로 prototype chain을 직접 지정
- Property 이름과 같은 변수이면 생략
- 함수 쉽게 기술
- Super 접근
- Property 이름을 동적으로 계산 가능
- Class 문법과 비슷

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

Template Literals

- **Multi line**
- **String interpolation**
  - 문자열 중간에 expr 삽입
- **Tag를 통해 추가적인 처리 가능**
  - Tag는 파싱된 template string을 인자로 받고 임의의 값을 리턴할 수 있는 함수
  - 자동으로 파싱이 된다는 점에서 단순 함수 호출과 차이

```
// Basic literal string creation
`In JavaScript '\n' is a line-feed.`

// Multiline strings
`In JavaScript this is
 not legal.`

// String interpolation
var name = "Bob", time = "today";
`Hello ${name}, how are you ${time}?`

// Construct an HTTP request prefix is used to interpret the replacements and construction
POST`http://foo.org/bar?a=${a}&b=${b}
     Content-Type: application/json
     X-Credentials: ${credentials}
     { "foo": ${foo},
       "bar": ${bar}}`(myOnReadyStateChangeHandler);
```

Destructuring (Pattern Matching)

- **Array, Object의 요소를 쉽게 분해**

ES5

```
var list = [ 1, 2, 3 ];
var a = list[0], b = list[2];
var tmp = a; a = b; b = tmp;
```

ES6

```
var list = [ 1, 2, 3 ]
var [ a, , b ] = list
[ b, a ] = [ a, b ]
```

Destructuring (Pattern Matching)

- **Array, Object의 요소를 쉽게 분해**
- **Nesting 된 것도 분해 가능**
- **Property 이름과 변수 이름이 같을 때 간략한 표현**
- **인자에서도 가능**
- **없는 요소의 경우 undefined**
- **Default 값 지정 가능**
- **다른 매치 시 에러**
  - ❯ 단, Iterable은 array로 분해 가능

```
// list matching
var [a, , b] = [1,2,3];

// object matching
var { op: a, lhs: { op: b }, rhs: c }
       = getASTNode()

// object matching shorthand
// binds `op`, `lhs` and `rhs` in scope
var {op, lhs, rhs} = getASTNode()

// Can be used in parameter position
function g({name: x}) {
  console.log(x);
}
g({name: 5})

// Fail-soft destructuring
var [a] = [];
a === undefined;

// Fail-soft destructuring with defaults
var [a = 1] = [];
a === 1;
```

Default + Rest + Spread Parameters

- **호출시에 계산되는 Default 인자**
  - 왼쪽부터 계산됨. Expr 가능
- **나머지 인자들 배열**
  - 기존의 arguments 대체
  - 인자가 없으면 빈 배열
- **배열을 풀어서 인자들로 넘기기**

```
function f(x, y=12) {
  // y is 12 if not passed (or passed as undefined)
  return x + y;
}
f(3) == 15
```

```
function f(x, ...y) {
  // y is an Array
  return x * y.length;
}
f(3, "hello", true) == 6
```

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

Let + Const

- **let - Block 스코프 변수 선언**
  - var 대체
  - Global 객체에 들어가지 않음
  - for 문의 init 위치에서 선언한 경우 iteration마다 새로 생김
  - Forward reference 금지
- **const - 상수 선언**

```
function f() {
  {
    let x;
    {
      // okay, block scoped name
      const x = "sneaky";
      // error, const
      x = "foo";
    }
    // error, already declared in block
    let x = "inner";
  }
}
```

```
for (let i = 0; i < a.length; i++) {
    let x = a[i]
    ...
}
for (let i = 0; i < b.length; i++) {
    let y = b[i]
    ...
}

let callbacks = []
for (let i = 0; i <= 2; i++) {
    callbacks[i] = function () { return i * 2 }
}
callbacks[0]() === 0
callbacks[1]() === 2
callbacks[2]() === 4
```

Iterators + For-Of

- **for-in 객체 property 순회**
- **for-of iterable 순회**
  - Symbol.iterator 메소드가 있으면 iterable
- **Symbol.iterator 메소드**
  - iterator를 반환
- **Iterator**
  - next() 메소드가 다음 값 정보를 반환
  - 필요할 때만 다음 값이 계산 됨
  - 직접 Iterator protocol을 구현해야 하는 경우는 별로 없음 → Generator 이용

```
let fibonacci = {
    [Symbol.iterator]() {
        let pre = 0, cur = 1;
        return {
            next () {
                [ pre, cur ] = [ cur, pre + cur ];
                return { done: false, value: cur };
            }
        };
    }
}

for (let n of fibonacci) {
    if (n > 1000)
        break;
    console.log(n);
}
```

Generators

- **실행 중 멈췄다 다시 실행할 수 있는 특별한 함수**
  - function* 키워드 사용
  - Iterator를 리턴
  - next()를 호출하면 실행됨
  - yield 문에서 멈추고 앞의 next()가 리턴 됨

```
let fibonacci = {
    *[Symbol.iterator]() {
        let pre = 0, cur = 1;
        for (;;) {
            [ pre, cur ] = [ cur, pre + cur ];
            yield cur;
        }
    }
}

for (let n of fibonacci) {
    if (n > 1000)
        break;
    console.log(n);
}
```

```
function* range (start, end, step) {
    while (start < end) {
        yield start;
        start += step;
    }
}

for (let i of range(0, 10, 2)) {
    console.log(i); // 0, 2, 4, 6, 8
}
```

Promises

- **나중에(asynchronously) 값이 채워지는 것**
  - 이미 광범위하게 사용 중 (Q, Bluebird,... library)
  - 3가지 상태(Pending, Fulfilled, Rejected)
  - then() 등의 메소드로 중첩된 callback을 좀 더 flat하게 표현 가능

Callback Style

```
function msgAfterTimeout (msg, who, timeout, onDone) {
    setTimeout(function () {
        onDone(msg + " Hello " + who + "!");
    }, timeout);
}
msgAfterTimeout("", "Foo", 100, function (msg) {
    msgAfterTimeout(msg, "Bar", 200, function (msg) {
        console.log("done after 300ms:" + msg);
    });
});
```

Promise Style

```
function msgAfterTimeout (msg, who, timeout) {
    return new Promise((resolve, reject) => {
        setTimeout(() => resolve(`${msg} Hello ${who}!`), timeout);
    });
}
msgAfterTimeout("", "Foo", 100).then((msg) =>
    msgAfterTimeout(msg, "Bar", 200)
).then((msg) => {
    console.log(`done after 300ms:${msg}`);
});
```

Promises + Generators

- Callback Hell 해결
- Async operation을 sync 스타일로 작성 가능

```
ajaxCallback("http://some.url.1", function(result1){
    var data = JSON.parse(result1);
    ajaxCall("http://some.url.2/?id=" + data.id, function(result2){
        var resp = JSON.parse(result2);
        console.log( "The value you asked for: " + resp.value );
    });
});

ajaxPromise("http://some.url.1")
.then(function(result1){
    var data = JSON.parse(result1);
    return ajaxPromise("http://some.url.2/?id=" + data.id);
}).then(function(result2){
    var resp = JSON.parse(result2);
    console.log( "The value you asked for: " + resp.value );
});

Q.spawn(function*() {
    var result1 = yield ajaxGenerator("http://some.url.1");
    var data = JSON.parse(result1)
    var result2 = yield ajaxGenerator("http://some.url.2/?id=" + data.id);
    var resp = JSON.parse(result2);
    console.log( "The value you asked for: " + resp.value );
});
```

Modules

- **모듈은 파일 단위**
- **export 키워드로 내보낼 toplevel 요소 선언**
- **import 키워드로 가져올 요소 선언**
  - 모듈 통째로도 import 가능
  - 이름을 변경하여 import 가능
- **CommonJS, AMD 모듈과 공존 가능**
- **완전히 static**
  - 의존 관계를 최초 로딩 시점에 모두 파악
  - 쉽게 bundling 가능
- **무조건 strict mode로 해석됨**
- **Toplevel 요소는 module-local**
- **항상 비동기 로딩됨**
- **HTML에서 로딩**
  - <script type=module>

```
//  lib/math.js
export function sum (x, y) { return x + y };
export var pi = 3.141593;

//  someApp.js
import * as math from "lib/math";
console.log("2π = " + math.sum(math.pi, math.pi));

//  otherApp.js
import { sum, pi } from "lib/math";
console.log("2π = " + sum(pi, pi));
```

Map + Set + WeakMap + WeakSet

- **유용한 collection 라이브러리**
- **Map: Any value → any value**
  - Object: string → value
- **Set: Any values**
- **Key/Value Equality**
  - === 사용. 단, NaN==NaN.
- **WeakMap, WeakSet**
  - 다른 reference가 없으면 자동으로 GC됨
    - Cache에 유용
  - 제한된 동작
    - WeakMap의 key와 WeakSet의 value는 객체만 가능
    - WeakMap - .has(), .get(), .set(), .delete()
    - WeakSet - .has(), .add(), .delete()
    - Iterable이 아님

```
// Sets
var s = new Set();
s.add("hello").add("goodbye").add("hello");
s.size === 2;
s.has("hello") === true;

// Maps
var m = new Map();
m.set("hello", 42);
m.set(s, 34);
m.get(s) == 34;

// Weak Maps
var wm = new WeakMap();
wm.set(s, { extra: 42 });
wm.size === undefined

// Weak Sets
var ws = new WeakSet();
ws.add({ data: 42 });
```

다른 참조가 없으므로 자동으로 GC

Symbols

- **JS에 새로 도입된 타입**
  - Symbol([desc])로 생성
- **다른 어느 값과도 다른 unique한 값을 생성**
  - Enum과 비슷하지만 integer가 아님
  - Description을 줄 수 있지만 디버깅 용도
- **Property의 key로 사용 가능**
  - 기존에는 string만 가능했음
  - 하지만 iterate되지 않음
    - 즉, Object.keys(), Object.getOwnProperyNames()에 나오지 않음

```
Symbol("foo") !== Symbol("foo");
const foo = Symbol();
const bar = Symbol();
typeof foo === "symbol";
typeof bar === "symbol";
let obj = {};
obj[foo] = "foo";
obj[bar] = "bar";
JSON.stringify(obj); // {}
Object.keys(obj); // []
Object.getOwnPropertyNames(obj); // []
Object.getOwnPropertySymbols(obj); // [ foo, bar ]
```

Math + Number + String + Array + Object APIs

```
Number.EPSILON
Number.isInteger(Infinity) // false
Number.isNaN("NaN") // false

Math.acosh(3) // 1.762747174039086
Math.hypot(3, 4) // 5
Math.imul(Math.pow(2, 32) - 1, Math.pow(2, 32) - 2) // 2

"abcde".includes("cd") // true
"abc".repeat(3) // "abcabcabc"

Array.from(document.querySelectorAll('*')) // Returns a real Array
Array.of(1, 2, 3) // Similar to new Array(...), but without special one-arg behavior
[0, 0, 0].fill(7, 1) // [0,7,7]
[1, 2, 3].find(x => x == 3) // 3
[1, 2, 3].findIndex(x => x == 2) // 1
[1, 2, 3, 4, 5].copyWithin(3, 0) // [1, 2, 3, 1, 2]
["a", "b", "c"].entries() // iterator [0, "a"], [1,"b"], [2,"c"]
["a", "b", "c"].keys() // iterator 0, 1, 2
["a", "b", "c"].values() // iterator "a", "b", "c"

Object.assign(Point, { origin: new Point(0,0) })
```

Proxy + Reflect

- **Proxy**
  - Target 객체로의 내부 동작(internal method)을 가로챔
  - new Proxy(target, handler)
    - Target: target 객체
    - Handler: internal method 모음
- **Internal method**
  - [[Get]]: x = obj.p, [[Set]]: obj.p = x, [[HasProperty]], …
- **Reflect**
  - 원래의 동작을 호출
  - .get, .set, …
- **Use cases**
  - Mock object
  - Logging, Debugging
  - 없는 property acces 경고
  - 음수 array access 지원
  - Data binding
  - 임의의 REST API 호출
  - Remote Procedure Call
  - DB Access

기타

- **Typed Arrays**
  - Binary 데이터 다루기
- **Binary and Octal Literals**
  - 0b101101, 0o34017
- **Proper Tail Call (Tail Call Optimisation)**
  - 특정한 모양의 재귀함수는 stack overflow 없음을 보장
- **Built-in 객체의 subclassing**
- **Block scoped 함수**
- **Unicode 버전 업그레이드**
- **국제화 스펙**
  - 통화 기호, 날짜, 등등
- **새로운 RegExp 기능**
- **…**

3. ES6를 사용하기

우리 프로젝트에 ES6를 써도 되나요?

- **상황에 따라 다름**
- **각 환경의 ES6 지원 현황**
  - 기본적으로 지원 현황을 잘 파악하고 사용해야 함
- **Node.js**
  - 상당 부분 사용 가능
- **Chrome, Firefox, Edge**
  - 상당 부분 사용 가능
- **IE**
  - Compiler 및 Polyfill 사용 필수
  - 여러 통계에서 5~20% 정도의 점유율을 보임

Compiler(Transpiler) + Polyfill

- **Babel**
  - 현재 ES6에 대한 지원이 가장 많은 Compiler (74%)
  - Polyfill도 쉽게 적용 가능
- **Traceur**
  - 구글에서 만든 Compiler (58%)
- **TypeScript**
  - ES6에 타입 annotation 확장 (60%)
  - Microsoft에서 만듦
- **ES5 엔진 자체의 한계로 위의 도구들로도 구현 불가능하거나 어려운 기능들이 있으므로 잘 살펴보고 써야 함**

참고 자료

- **ECMAScript 2015 Language Specification**
  - http://www.ecma-international.org/ecma-262/6.0/
- **Kangax ECMAScript Compatibility Table**
  - http://kangax.github.io/compat-table/es6/
- **MDN JavaScript Reference (우리의 친구 MDN)**
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference
- **ES6 In Depth series - Mozilla blog (각 기능에 대한 깊이 있는 설명. 재미있음. 이 발표에서 많이 참조함)**
  - http://hacks.mozilla.or.kr/category/es6-in-depth/
- **ECMAScript 6 — New Features: Overview & Comparison (기능별로 ES5와 코드 비교)**
  - http://es6-features.org
- **ES6 Features (ES6 전체에 대한 개괄적인 소개. 이 발표에서 많이 참조함)**
  - http://git.io/es6features
- **ECMAScript 6: A Better JavaScript for the Ambient Computing Era (이 발표에서 많이 참조함)**
  - https://www.youtube.com/watch?v=BnwFYJDes1c
  - http://www.slideshare.net/allenwb/wdc14-allebwb
- **Exploring ES6: Upgrade to the next version of JavaScript (무료 ebook. ES6 전체에 대한 상세하고 친절한 설명)**
  - http://exploringjs.com/
- **Browser Stats**
  - https://www.netmarketshare.com/browser-market-share.aspx?qprid=2&qpcustomd=0
  - https://www.w3counter.com/globalstats.php
  - https://en.wikipedia.org/wiki/Usage_share_of_web_browsers
  - http://www.w3schools.com/browsers/browsers_stats.asp