ECMAScript 2015:ES6

EGCI427-Lecture05

Reference: Teppei Sato, "Effective ES6", Cybozu, Inc., Tokyo, Japan, 21 Aug, 2015

Example: No classs

No Class

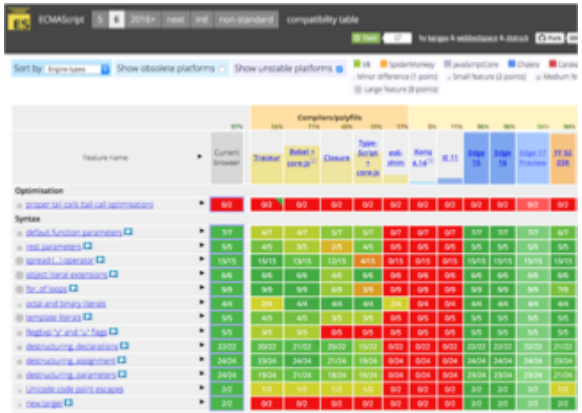
```
function Person(name) {
  this.name = name;
}
Person.prototype.greet = function() {
   console.log("Hello, I'm " + this.name);
};
var bob = new Person("Bob");
bob.greet();
// "Hello, I'm Bob"
```

If you miss"new", dangerous!

```
function Person(name) {
  this.name = name;
}
// Oh! You forget `new`
var bob = Person("Bob");
console.log(bob); // undefined
// Global leak!!!!!!!!!!
console.log(window.name); // "Bob"
```

ECMAScript 6

- Published on June 17, 2015
  - Formally "ECMAScript 2015"
  - 6 years! from ES5
- Modern syntax fixing pitfalls
- Better support for large applications
- No (or few) breaking changes
- ES6 Compatibility Table
  - https://kangax.github.io/compat-table/es6/



Transpiler and polyfill

- ES6 Transpiler:
  - Source code converter from ES6 to ES5/ES3
- ES6 Polyfill:
  - Library to provide ES6 built-in classes,functions and objects for ES5/ES3

Babel

- The most compatible (71%) ES6 transpiler
- Integrated with core-js (polyfill library)

ES6 NEW FEATURES

- New syntax
- New built-in classes and objects
- Improvement of existing classes

ES6 FEATURES: NEW SYNTAX

NEW SYNTAX

- Arrow Function
- Classes
- Modules
- Block Scope (let/const)
- Extended Object Literal
- Default Params
- Rest Params
- Spread Operator
- Destructuring
- Iterator
- Generator
- Template Literal
- Tail Call Optimization

ARROW FUNCTION

ARROW FUNCTION

```
// ES5 old function
var add = function(a, b) {
return a + b; };
console.log(add(1,2));

// ES6 arrow function!
var add = (a, b) => {
return a + b; };
console.log(add(1,2));
```

```
//ES6
var add = (a, b) => a + b;
console.log(add(1,2));

var square=n=>n*n;
console.log(square(2));

// good for array filter chains
var result = [1, 2, 3, 4].filter(n => n % 2 === 0).map(n => n * n);
console.log(result);
```

‘this’ and ‘self’

```
//ES5
var john = {
   name: "John",
   helloLater: function() {
      // save `this` as `self`
      var self = this;
      setTimeout(function() {
        // `this` is not available. use `self`.
        console.log("Hello, I'm " + self.name);
      }, 1000);
   }
}
john.helloLater();
// "Hello, I'm John" after 1sec
```

```
//ES6
let john = {
   name: "John",
   helloLater: function() {
   // use arrow function
   setTimeout(
      () => {
         // `this` is available here!
         console.log("Hello, I'm " + this.name);
      }, 1000);
   }
}
john.helloLater();
// "Hello, I'm John" after 1sec
```

Classes

```
//ES5
function Person(name) {
   this.name = name;
}
Person.prototype.greet = function() {
   console.log("Hello, I'm " + this.name);
};
var bob = new Person("Bob");
bob.greet();
// "Hello, I'm Bob"
```

```
//ES6
class Person {
 constructor(name) {
 this.name = name;
 }
 greet() {
  console.log("Hello, I'm " + this.name);
 }
        }
var bob = new Person("Bob");
bob.greet();
```

Class Inheritance

```
//ES6
class Person {
 constructor(name) {
  this.name = name;
 }
 greet() {
  console.log("Hello, I'm " + this.name);
 }
}
class Programmer extends Person {
   constructor(name, language) {
      super(name);
      this.language = language;
   }
   greet() {
      super.greet();
      console.log("I like " + this.language);
   }
}
```

```
var bob = new Programmer("Bob", "JavaScript");
bob.greet();
// "Hello, I'm Bob"
// "I like JavaScript"
```

Block Scope (let/const)

```
//es5
function foo() {
  var num=1;
  // ... too many statements
  if (num == 1) {
    // same scope! overwrite above `num`!
    var num = 2;
    // .. some process
  }
  console.log(num);
  // 2 !!!
}
foo();
```

```
//es6
function foo() {
  let num=1;
  // ... too many statements
  if (num==1) {
    // different scope!
    let num=2;
  }
  console.log(num);
  // 1
}
foo()a
```

Block Scope (let/const)

```
//es5
function foo() {
  var num=1;
  // ... too many statements
  if (num == 1) {
    // same scope! overwrite above `num`!
    var num = 2;
    // .. some process
  }
  console.log(num);
  // 2 !!!
}
foo();
```

```
//es6
function foo() {
  let num=1;
  // ... too many statements
  if (num==1) {
    // different scope!
    let num=2;
  }
  console.log(num);
  // 1
}
foo()a
```

Block Scope (let/const)

```
//es6
const foo = 1;
foo = 100; // Error!
const foo = 1000; // Error!
// properties are mutable
const obj = {};
obj.foo = 1; // No error
```

Default Parameters

Incorrect default params

//es5

```
function add(a, b) {
// if "a" is 0, 1 is assigned to "a".
  a=a||1;
  b=b||2;
  return a + b;
}
let result = add(0, 0); // 1 + 2 = 3
console.log(result);
```

//es6

```
// default value for each param
function add(a = 1, b = 2) {
  return a + b;
}
console.log(add()); // 1 + 2 = 3
console.log(add(0)); // 0 + 2 = 2
console.log(add(undefined, 0)); // 1 + 0 = 1
console.log(add(0, 0)); // 0 + 0 = 0
```

Rest Parameters

```
//es5
function foo(first, second) {
  console.log("first:", first);
  console.log("second:", second);
  // arguments is an ArrayLike, not an Array.
  var rest = Array.prototype.slice.call(arguments, 2);
  console.log("rest:", rest);
}
foo(1, 2, 3, 4, 5);
// first: 1
// second: 2
// rest: [3, 4, 5]
```

```
//es6
function foo(first, second, ...rest) {
  console.log("first:", first);
  console.log("second:", second);
  console.log("rest:", rest);
}
foo(1, 2, 3, 4, 5);
// first: 1
// second: 2
// rest: [3, 4, 5]
```

Destructuring

```
//es6
let match = /(\d{4})(\d{2})(\d{2})/.exec("20151231");
// match: [2015151231, 2015, 12, 31]
let [, year, month, day] = match;
console.log(year, month, day); // 2015 12 31

//es6
let {name: a, age: b} = {name: "Bob", age: 20};
console.log(a, b); // "Bob" 20

// shorthand
let {name, age} = {name: "Bob", age: 20};
console.log(name, age); // "Bob" 20
```

Template Literal

```
//es5
// concat with variables
var name = 'Bob';
var str = "Hello, I'm " + name + ".";
// create multiple lines
var multi = ["line1", "line2", "line3"].join("\n");
console.log(multi);
```

```
//es6
// interpolation
var name = 'Bob';
var str = `Hello, I'm ${name}.`;
// multiple lines
var multi = `line1 line2 line3`;
console.log(multi);
```

Extended Object Literal

```
//es6
let foo=1;
let bar=2;

// shorthand
let obj = {foo, bar};
// same as: {foo: foo, bar: bar};
console.log(obj)
```

```
//es6
let foo=1;
let bar=2;
let prefix = 'foo';
let obj = {
  // computed property
  [prefix + 'abc']: 1,
  // method definition without "function" keyword
  foo() {
    console.log('foo!');
  }
};
console.log(obj['fooabc'])
console.log(obj.foo())
```

New built-in classes and objects

New built-in classes and objects

- Promise
- Map
- Set
- WeakMap/WeakSet
- TypedArray
- Symbol
- Proxy/Reflect

Promise

```
//es5
function asyncTask(a, b, callback) {
  // ...some async task
   if (error) {
     callback(error);
   }else{
     callback(null, result);
  }
}
asyncTask(1, 2, function(error, result) {
  if (error) {
    // ...error handling
  }
  console.log(result);
});
```

```
//es6
function asyncTask(a, b, callback) {
  // ...some async task
  return new Promise((resolve, reject) => { if (error)
{
      reject(error);
    }else{
      resolve(result);
    }
  });
}

asyncTask(1, 2).then(result => {
     console.log(result);
   }).catch(error => {
     // ...error handling
});
```

Map/Set

```
//es5
var obj = {};
var key = "toString";
obj[key] = "value1";
String(obj);
// TypeError: can't convert obj to string
```

```
//es5
var key1 = {name: "key1"};
var key2 = {name: "key2"};
var obj = {};
obj[key1] = "value1";
obj[key2] = "value2";
console.log(obj[key1]);
// "value2"
console.log(Object.keys(obj));
// ["[object Object]"]
```

```
//es6
let map = new Map();
map.set("toString", "value1");
map.get("toString"); // "value1"
var result = String(map); // "[object Map]"
console.log(map.get("toString"));
console.log(map);
console.log(result);
```

Improvement of existing classes

Improvement of existing classes

- String
- RegExp
- Array
- Object
- Math
- Number

Object

```
//ES6
var target = {a: 1, b: 2};
var s1 = {b: 3, c: 4};
var s2 = {c: 5, d: 6};
var ret = Object.assign(target, s1,
s2);
console.log(target); // {a: 1, b: 3, c:
5, d: 6}
```

Spread Operator

```
//es5
var arr = ['a', 'b', 'c'];
for (var i in arr) {
  if (arr.hasOwnProperty(i)) {
    console.log("No. "+i+", value: "+arr[i]);
  }
}
// '0'
// '1'
// '2'
```

```
//es6
for (let n of ['a', 'b', 'c']) {
  console.log(n);
}
// 'a'
// 'b'
// 'c'
//es6
// Spread Operator
console.log([..."abcd"]);
// Array ["a", "b", "c", “d"]

// Destructure Assignment
[a, b] = "xy";
console.log("a: "+a+", b: "+b);

console.log(Array.from("123"));
// Array [ "1", "2", "3" ]
```