ECMAScript 2015(ES6) overview

Peter.cho

Index

Block Scope

var vs let

var is function scope

```
if (true) {
  var x = 3
}
console.log(x) //3
```

let is block scope

```
if (true) {
  let x = 3
}
console.log(x) //ReferenceError
```

var vs let – loop scoping

var

```
for (var i = 0; i < 3; i++) { }
console.log(i) //3
```

let

```
for (let i = 0; i < 3; i++) { }
console.log(i) //ReferenceError
```

let vs const

let is not immutable

```
let num = 0
num = 1 // Fine
```

const is immutable

```
const num = 0
num = 1 // TypeError
```

const

content can be changed

```
const obj = { a: 'a' }

obj.b = 'B' //Working
obj.a = 'A' //Working
delete obj.a //Working
```

freeze

```
const obj = { a: 'a' }
Object.freeze(obj)
obj.b = 'B' //Not Working
obj.a = 'A' //Not Working
delete obj.a //Not Working

const obj = { x: {} }
Object.freeze(obj)
obj.x.a = 'A' //Fine
```

Shorthand

Assignment

Object property

```
const ip = '127.0.0.1'
const port = 1234
const serverInfo = { ip, port }
// { ip: '127.0.0.1', port: 1234 }
```

Method Definition

```
const person = {
  name: '',
  getName () { return this.name },
  setName (name) { this.name = name; }
}
person.setName('Peter')
console.log(person.getName()) //Peter
```

Destructuring

Object

```
// personal.js
const peter = { weight: 72, height: 173 }


// inbody.js
function getBMI (weight, height) {
  height = height / 100
  return weight / (height * height)
}
const { weight, height } = peter
console.log(getBMI(weight, height)) // 24.0569...
```

Array

```
const [a, , b] = [0, 1, 2]
console.log(a, b) //0 2
```

Default value

Parameter

```
const serverInfo = {
  ip: null,
  port: null,
  setDevInfo (ip='127.0.0.1', port=1234) {
    this.ip = ip
    this.port = port
  }
}
serverInfo.setDevInfo()
//ip: 127.0.0.1, port: 1234
```

Destructuring

```
const peter = {
  weight: 72,
  height: 173
}
const { weight, height, age=25 } = peter
console.log(weight, height, age)
//72, 173, 25
```

...

Rest Operator

```
function foo (...args) {} //args : [1,2,3]
foo(1,2,3)

function bar (first, ...args) {} //args : [2,3]
bar(1,2,3)
```

Spread

```
const odd = [1, 3, 5]
const even = [2, 4, 6]
const num = [...odd, ...even] // [1, 3, 5, 2, 4, 6]
sum(...odd) //9

const obj1 = { a: 'a' }
const obj2 = { b: 'b' }
const mergedObj = { ...obj1, ...obj2 } // {a: 'a', b: 'b'}
```

Syntax sugar

String Template

String concatenation

```
const name = 'Peter'
const txt = `Hello World
I'm ${name}`
/*
Hello World
I'm Peter
 */
```

Expression

```
const math = 90
const science = 100
console.log(`Math: ${math}
Sciene: ${science}
Total: ${math + science}
Average: ${(math + science) / 2}`)
```

Arrow function

function declaration

```
function sum (a, b) {
  return a + b
}

function getBMI (weight, height) {
  height /= 100
  return weight / Math.pow(height, 2)
}
```

Arrow function

```
const sum = (a, b) => a + b


const getBMI = (weight, height) => {
  height /= 100
  return weight / Math.pow(height, 2)
}
```

Arrow function

Always anonymous

```
const sum = (a, b) => a + b
```

```
const sum = sum(a, b) => a + b
//SyntaxError
```

Lexical this

```
const obj = {
  data: '',
  updateData () {
    $http.get('/path')
      .then(data => this.data = data)
  }
}
```

Arrow function

It can’t be used constructor

```
const Person = () => {}
new Person() //TypeError

//Because of
Person.prototype //Undefined
```

function declaration

```
const Person = function () {}
new Person() //Fine

//Because of
Person.prototype //Support
```

Module

Export & Import

export

```
export function sum (...numbers) {
  return numbers.reduce((prev, cur) => {
    return prev + cur
  })
}
export function avg (...numbers) {
  const sumResult = sum(...numbers)
  return sumResult / numbers.length
}
```

import

```
import { sum, avg } from './lib'

sum(1,2,3,4) //10
avg(1,2,3,4) //2.5
```

default & alias

default

```
//myFunc.js
export default function () {}

//main.js
import myFunc from './myFunc'
myFunc()
```

alias

```
import { getTime } from './bar'
import { getTime } from './foo'
//Duplicate declaration

import * as bar from './bar'
import * as foo from './foo'

import { getTime as getTimeOfBar } from './bar'
import { getTime as getTimeOfFoo } from './foo'
```

Singleton

```
//instance.js
class Person {}
export const person = new Person()
```

```
//main.js
import barInstance from './bar'
import fooInstance from './foo'
barInstance === fooInstance
```

```
//bar.js
import { person } from './instance'
const barInstance = person
export default barInstance
```

```
//foo.js
import { person } from './instance'
const fooInstance = person
export default fooInstance
```

Import is read-only

```
//main.js
import { counter, incCounter } from './lib'

console.log(counter) // 3
incCounter()
console.log(counter) // 4
counter++
//SyntaxError 'counter' is read-only
```

```
//lib.js
export let counter = 3
export function incCounter() {
  counter++
}
```

Class

Create

Class declaration

```
class MyClass {}
const instance = new MyClass()
```

Class expression

```
const MyClass = class {}
const instance = new MyClass()
```

Sub classing

```
class Point {
  constructor (x, y) {
    this.x = x
    this.y = y
  }
  toString () {
    return `${this.x} ${this.y}`
  }
}
```

extends, super

```
class ColorPoint extends Point {
  constructor (x, y, color) {
    super(x, y) //Must call super
    this.color = color
  }
  toString () {
    return `${super.toString()} in ${this.color}`
  }
}
```

Static

```
class Point {
  static pointMethod () {}
}
```

```
Point.pointmethod()
```

```
class ColorPoint extends Point {
  static pointmethod () {
    super.pointMethod()
  }
}
```

```
ColorPoint.pointmethod()
```

Getter & Setter

```
class Point {
  constructor (x, y) {
    this.x = x
    this.y = y
  }
  get axis () { return [this.x, this.y] }
  set axis ([x, y]) {
    this.x = x
    this.y = y
  }
}
```

```
const point = new Point(0,0)

console.log(point.axis) //[0, 0]
point.axis = [10, 10]
console.log(point.x, point.y) //10, 10
```

Data Structure

Map

```
const map = new Map()


map.set('foo', true)
map.set('bar', false)
map.get('foo') //true
map.has('foo') //true
map.delete('foo')
map.size //2
map.clear() //map.size === 0
```

```
const map = new Map([
  ['foo', true],
  ['bar', false]
])
```

Any value can be a key, even an object

Getting an unknown key produces undefined

WeakMap

Map

```
const map = new Map()
let obj = {}
map.set(obj, false)
console.log(map.get(obj))
obj = null
console.log(map.entries())
//{ [{}, false] }
```

WeakMap

```
const weakMap = new WeakMap()
let obj = {}
weakMap.set(obj, false)
console.log(weakMap.get(obj)) //false
obj = null
// obj in weakMap is garbage-collected


//only get, set, has, delete methods
```

Set

```
const set = new Set()
set.add('red')
set.has('red') //true
set.delete('red')
set.has('red') //false

set.add('red')
set.add('green')

set.size //2
set.clear() //set.size === 0
```

```
const set = new Set(['red', 'green', 'blue'])


//Chainable
set
  .add('purple')
  .add('black')
```

WeakSet

Set

```
const set = new Set()
let obj = {}
set.add(obj)
set.has(obj) //true
obj = null
set.keys() //{ {} }
```

WeakSet

```
const weakSet = new WeakSet()
let obj = {}
weakSet.add(obj)
weakSet.has(obj) //true
obj = null
//obj in weakSet is garbage-collected


//only add, has, delete methods
```

Promise

Usage

resolve / reject

```
const promise = new Promise(
  (resolve, reject) => {
    getData(
      response => resolve(response.data),
      error => reject(error.message)
    )
  }
)
```

then / catch

```
promise
  .then(data => console.log(data))
  .catch(err => console.error(err))
```

Multiple

all

```
Promise.all([
  getPromise(),
  getPromise(),
  getPromise(),
])
  //response all data
  .then(data => console.log(data))
  .catch(err => console.error(err))
```

race

```
Promise.race([
  getPromise(), //1000ms
  getPromise(), //500ms
  getPromise(), //250ms
])
  //response of 250ms
  .then(data => console.log(data))
  .catch(err => console.error(err))
```

Symbol

Symbol type

Unique

```
const RED1 = Symbol('red')
const RED2 = Symbol('red')
console.log(RED1 === RED2) //false
```

Property Keys

```
const height = Symbol('height')
const obj = {age: 25}
obj[height] = 173
Object.getOwnPropertyNames(obj)
//[ 'age' ]

Object.getOwnPropertySymbols(obj)
// [ Symbol(height) ]
```

Clear intention

Bad

```
const SWITCH_OFF = 0
const EQUAL = 0

const getBtnStatus = () => SWITCH_OFF
const compareVersion = () => EQUAL

const btnStatus = getBtnStatus()
const result = compareVersion('0.0.1', '0.0.1')

btnStatus === comparedResult //true
```

Good

```
const SWITCH_OFF = Symbol(0)
const EQUAL = Symbol(0)

const getBtnStatus = () => SWITCH_OFF
const compareVersion = () => EQUAL

const btnStatus = getBtnStatus()
const result = compareVersion('0.0.1', '0.0.1')

btnStatus === comparedResult //false
```

Proxy

Intercept and customize operations

```
const target = { }
const proxy = new Proxy(target, {
  get (target, propKey) {
    console.log('GET', propKey)
    return target[propKey]
  },
  set (target, propKey, value) {
    console.log('SET', propKey)
    target[propKey] = value
  }
})

proxy.foo //GET foo
proxy.bar = 'abc' //SET bar
```

```
const target = { }
const proxy = new Proxy(target, {
  has (target, propKey) {
    console.log('HAS', propKey)
    return propKey in target
  },
  deleteProperty (target, propKey) {
    console.log('DELETE', propKey)
    delete target[propKey]
  }
})

'hello' in proxy //HAS hello
delete proxy.bara //DELETE bar
```

Function

```
const sum = (a, b) => a + b
const handler = {
  apply (target, thisArg, argumentsList) {
    return target(...argumentsList)
  }
}
```

```
const proxySum = new Proxy(sum, handler)

proxySum(1, 2) //3
```

Class

```
class Person {
  constructor (name) {
    this.name = name
  }
  getName () { return this.name }
}
const handler = {
  construct (target, args) {
    return new target(...args)
  }
}
```

```
const ProxyPerson = new Proxy(Person, handler)

const peter = new ProxyPerson('peter.cho')
peter.getName() //peter.cho
```

Other ES6 features

Features

- Iterator
- Generator
- Typed Array
- New Built-in Methods

Thank you