ECMA 各版本特性汇总

news/2024/1/11 14:49:08

<http://wed.xjx100.cn/news/165051.html?action=onClick>

1. ECMA

ecma官方网站  
ecma github版本记录  
w3schools  
ECMAScript是由网景的布兰登·艾奇开发的一种脚本语言的标准化规范；最初命名为Mocha，后来改名为LiveScript，最后重命名为JavaScript。1995年12月，升阳与网景联合发表了JavaScript。1996年11月，网景公司将JavaScript提交给欧洲计算机制造商协会进行标准化。ECMA-262的第一个版本于1997年6月被Ecma组织采纳。ECMA Script是ECMA-262标准化的脚本语言的名称。尽管JavaScript和JScript与ECMAScript兼容，但包含超出ECMA Script的功能。

ecma从2016年开始以年份命名，旧的es版本以数字命名，比如ES5 (2009) 和 ES6 (2015)  
但依然也可以用数字命名

| **版本** | **年份** |
| --- | --- |
| ES2009 Or ES5 | June 2009 |
| ES2015 Or ES6 | June 2015 |
| ES2016 Or ES7 | June 2016 |
| ES2017 Or ES8 | June 2017 |
| ES2018 Or ES9 | June 2018 |
| ES2019 Or ES10 | June 2019 |
| ES2020 Or ES11 | June 2020 |
| ES2021 Or ES12 | June 2021 |
| ES2022 Or ES13 | June 2022 |

接下来整理的是开发中常用的特性，一些作者认为不常用的可能会不记录或者记录的不详细

1. ES13

2022 年 6 月发布

1. 1. await和async解除强绑定

await不再必须写在async里了，在过去，我们通常的写法如下。

import posts from './posts';const getPosts = async() => {let posts = await posts();return posts;}

在es13中，我们可以直接这样。

let posts = await posts();

解除和async的强绑定有以下好处

1. 动态依赖路径  
   当你有依赖运行时值的动态路径时，await 有助于在运行时加载或导入消息。

const messages = await import(`./messages-${language}.js`);

1. 依赖回退  
   如果导入的模块加载失败，则加载的回退模块用于加载依赖。

let lodash;try {lodash = await import('https://first.domain.com/lodash');} catch {lodash = await import('https://second.domain.com/lodash');}

1. 资源初始化  
   此功能可用于使用数据库初始化应用程序。

import { dbConnector} from './dbUtils.js'

//connect to database

const connection = await dbConnector.connect();

export default function(){ connection.list()

}

1. 本身不作为promise类型返回，但内部业务代码处理需要使用await的
2. 2. 支持私有属性和方法

在过去，我们编写类的私有属性和方法时常借助typescript的private来实现，但使用js的话因为没有严格限制，所以需要特殊命名来进行标记，比如在变量命名前加下划线\_。但因为不强制进行检查，可能会导致业务错误。

class Employee {constructor() {this.name = "John"; //publicthis.\_age=35; //private}const employee = new Employee();employee.name = "Jack";employee.\_age =35; //No error

现在，es13原生支持了公开和私有的属性方法，提高了js代码的扩展性和可维护性。  
通过在变量前添加#号，标记其为私有属性或方法。  
在下面的示例中，Employee 类已在构造函数外部使用私有变量定义。

class Employee {name = "John";#age=35;constructor() {}#getAge() {return #age}}const employee = new Employee();employee.name = "Jack";employee.#age = 35; // Throws an error

并且还支持静态字段和方法。只需要在前面加入static标记即可。

class Employee{name = "John";static #employerName="Github"static #getEmployerName() {return #employerName}

}

const employee = new Employee();

employee.emp = "Jack";

employee.#employerName = 35; // Throws an error

1. 3. 数组at()方法

该.at()方法用于通过传递负索引值来访问数组或字符串元素。即，它允许从数组末尾或字符串访问值。  
在 ES2022 之前，您应该必须从末尾访问元素，如下所示。

const array = [1, 2, 3, 4, 5];

console.log(array[array.length - 2]); //4

console.log(array.slice(-2)[0]); //4const string = '12345';

console.log(string[string.length - 2]); // '4'

console.log(string.slice(-2)[0]); // '4'

现在你应该可以写。

const array = [1, 2, 3, 4, 5];

console.log(array.at(-2)); // 4const string = '12345';

console.log(string.at(-2));

1. 4. 自定义错误原因

在下面的示例中，让我们从 JSON 处理中捕获错误，并使用一条新的有意义的消息以及错误的原始原因重新抛出它。

function processUserData(arrayData) {return arrayData.map(data => {try {const json = JSON.parse(data);return json;} catch (err) {throw new Error(`Data processing failed`,{cause: err});}});}

1. 5. Object.hasOwn()

新的Object.hasOwn()方法替代了Object.prototype.hasOwnProperty。它是一个静态方法，比hasOwnProperty写法更简洁，并解决了hasOwnProperty的一些问题。

如果您在 create(null) 函数的帮助下创建新对象，那么新创建的对象不会继承自 Object.prototype。所以它没有 hasOwnProperty 方法。  
让我们举个例子来验证使用 create(null) 函数创建的对象上的 hasOwnProperty。

const user = Object.create(null);

user.age = 35;

user.hasOwnProperty('age'); // throws a TypeError

在这种情况下，可以使用 hasOwn() 方法来确定自己的属性。

const user = Object.create(null);user.age = 35;user.hasOwn('age'); // true

1. 6. 正则表达式匹配索引

正则表达式匹配已升级为包含有关匹配组的更多信息。\d附加信息包括在输入字符串中使用标志的 RegExp 中匹配项的开始和结束索引。

1. ES12

2021年中发布

1. 1. replaceAll()

prototype 中的新replaceAll()方法String用于从另一个字符串值替换所有出现的字符串。早些时候，不使用正则表达式是不可能替换子字符串的所有实例的。  
ES2021之前

console.log('10101010'.replace(new RegExp('0', 'g'), '1')); // 11111111console.log('01010101'.replace(/0/g, '1')); // 11111111

ES2021之后

console.log('10101010'.replaceAll('0', '1')); // 11111111console.log('01010101'.replaceAll('0', '1')); // 11111111

1. 2. promise.any

被传入的promise数组，先成功的将被作为返回值处理。如果全部失败则抛出异常。

let promise1 = new Promise((resolve) => setTimeout(resolve, 100, 'Resolves after 100ms'));

let promise2 = new Promise((resolve) => setTimeout(resolve, 200, 'Resolves after 200ms'));

let promise3 = new Promise((resolve, reject) => setTimeout(reject, 0) );let promises = [promise1, promise2, promise3];Promise.any(promises).then( value => console.log(value)); // Resolves after 100ms

(async () => {try {const output = await Promise.any([Promise.reject('Error 1'),Promise.reject('Error 2'),Promise.reject('Error 3'),]);console.log(`Output: ${output}`);} catch (err) {console.log(`Error: ${err.errors}`);}})(); // Error: Error1,Error2,Error3

1. 3. 弱引用

WeakRef 提供了两个新功能

1. 使用 Wea​​kRef 类创建对对象的弱引用
2. 在对象被垃圾收集后运行用户定义的终结器，使用 FinalizationRegistry 类  
   在 ES12 之前，WeakMaps 和 WeakSets 是在 JavaScript 中弱引用对象的唯一方法。而 ES12 中的 WeakRef 提供了实际的弱引用，为对象的生命周期打开了一个窗口。

const myObject = new WeakRef({name: ‘Sudheer’,age: 34

});

console.log(myObject.deref()); //output: {name: “Sudheer”, age: 35}

console.log(myObject.deref().name); //output: Sudheer

对象FinalizationRegistry允许您在对象被垃圾回收时请求回调。它用作清理回调。

// Create new FinalizationRegistry:

const reg = new FinalizationRegistry((val) => {console.log(val);

});(() => {

// Create new object:const obj = {}// Register finalizer for the "obj" as first argument and value for callback function as second argument:reg.register(obj, 'obj has been garbage-collected.')

})();

1. 4. 数字分隔符

数字分隔符通过使用下划线 (\_) 在数字之间提供分隔，有助于在 JavaScript 中读取大数字（或数字文字）。换句话说，通过在数字组之间创建视觉分隔，数字文字更具可读性。

1. 5. 逻辑运算符

逻辑赋值运算符将逻辑运算符（&&、|| 或 ??）与赋值结合起来。它们对于为变量分配默认值非常有用。  
&&=:比如以下代码。

x = x && (x = y); (OR)if (x) {x = y;}

在es12可以优化为

let x = 10;let y = 20;x &&= y;console.log(x); // 20

||=:比如以下代码

x = x || (x = y); (OR)if (!x) {x = y;}

可以优化为

let x = 0;let y = 20;x ||= y;console.log(x); // 20

??=:比如以下代码。

x ?? (x = y);(OR)if (!x) {x = y;}

可以优化为

let x;let y = 1;x ??= y;console.log(x); // 1

1. ES11

2020 年 6 月

1. 1. 大整数

在早期的 JavaScript 版本中，使用 Number 类型是有限制的。Number即，您不能安全地表示大于 pow(2, 53) 的整数值（原始值）。  
在 ES2020 中，

BigInt作为第 7 种原始类型引入，以表示大于 pow(2, 53) - 1（或 9007199254740991 或 Number.MAX\_SAFE\_INTEGER）的整数（具有任意精度的整数）。

// 1. Current number system

const max = Number.MAX\_SAFE\_INTEGER;

console.log(max + 1) // 9007199254740992

console.log(max + 2) // 9007199254740992// 2. BigInt representation

const bigInt = 9007199254740991n;

const bigIntConstructorRep = BigInt(9007199254740991); // 9007199254740991n

const bigIntStringRep = BigInt("9007199254740991"); // 9007199254740991n// 3. Typeof usageconsole.log(typeof 1)// number

console.log(typeof 1n)// bigint

console.log(typeof BigInt('1'))// bigint// 4. Operatorsconst previousMaxNum = BigInt(Number.MAX\_SAFE\_INTEGER);

console.log(previousMaxNum + 2n); //9007199254740993n (this was not possible before)

console.log(previousMaxNum -2n); //9007199254740990n

console.log(previousMaxNum \* 2n); //18014398509481982n

console.log(previousMaxNum % 2n); //1n

console.log(previousMaxNum / 2n); // 4503599627370495n// 5. comparison

console.log(1n === 1); // false

console.log(1n === BigInt(1)); // true

console.log(1n == 1); // true

1. 2. 动态导入

引入新功能dynamic import是为了有条件地或按需加载模块。由于它返回请求模块的模块命名空间对象的承诺，因此可以解析模块或现在可以使用 async/await 将导入分配给变量，如下所示

<script>

const moduleSpecifier = './message.js';

import(moduleSpecifier).then((module) => {module.default(); // Hello, default exportmodule.sayGoodBye(); //Bye, named export}).catch(err => console.log('loading error'));

</script>

<script>

(async function() {const moduleSpecifier = './message.js';const messageModule = await import(moduleSpecifier);messageModule.default(); // Hello, default exportmessageModule.sayGoodBye(); //Bye, named export

})();

</script>

并且导入的模块出现默认和命名导出

export default () => {return "Hello, default export";

}

export const sayGoodBye = () => {return "Bye, named export"

}

注意：动态导入不需要脚本type=“module”

1. 3. 无效合并运算符

无效合并运算符 (??) 是一个逻辑运算符，当其左侧操作数为nullor时返回其右侧操作数undefined，否则返回其左侧操作数。  
对于传统的||或运算符，如果您将空值或 ‘’、0 和 NaN 视为有效值，则此运算符会替换运算符以提供默认值。这是因为逻辑 OR(||) 运算符将（空值或 ‘’、0 和 NaN）视为假值并返回正确的操作数值，这在这种情况下是错误的。因此，此运算符真正检查的是null值而不是false值。

let vehicle = {car: {name: "",speed: 0}

};console.log(vehicle.car.name || "Unknown"); // Unknown

console.log(vehicle.car.speed || 90); // 90console.log(vehicle.car.name ?? "Unknown"); // ""(empty is valid case for name)

console.log(vehicle.car.speed ?? 90); // 0(zero is valid case for speed)

简而言之，空值运算符返回一个非空值和 || 运算符返回真值。

1. 4. 字符串全匹配

有一种String#match方法可以通过迭代每个匹配项来获取字符串与正则表达式的所有匹配项。但是，此方法会为您提供匹配的子字符串。

这String#matchAll()是一个添加到 String 原型的新方法，它返回一个字符串与正则表达式匹配的所有结果的迭代器。

const regex = /t(e)(st(\d?))/g;

const string = 'test1test2';

const matchesIterator = string.matchAll(regex);

Array.from(matchesIterator, result => console.log(result));

当您在浏览器控制台中编写此代码时，匹配迭代器会为每个匹配项生成一个数组，包括带有一些额外内容的捕获组。

["test1", "e", "st1", "1", index: 0, input: "test1test2", groups: undefined]

["test2", "e", "st2", "2", index: 5, input: "test1test2", groups: undefined]

1. 5. 可选链接

在 JavaScript 中，长链的属性访问很容易出错，如果它们中的任何一个评估为null或undefined值。此外，检查每个项目的属性是否存在也不是一个好主意，这反过来会导致深层嵌套的结构化if语句。  
可选链接是一项新功能，它可以通过附加 (?.) 运算符来停止计算并在项目未定义或为 null 时返回未定义，从而使您的 JavaScript 代码看起来更清晰、更健壮。

let vehicle = {

};let vehicle1 = {car: {name: 'ABC',speed: 90}

};console.log(vehicle.car.name); // TypeError: Cannot read property 'name' of undefinedconsole.log(vehicle.car?.name); // Undefined

console.log(vehicle.car?.speed); // Undefinedconsole.log(vehicle1.car?.name); // ABC

console.log(vehicle1.car?.speed); // 90console.log(vehicle.car?.name ?? "Unknown"); // Unknown

console.log(vehicle.car?.speed ?? 90); // 90

1. 6. allSettled

先讲一下promise.all也是接收一个数组列表，但如果有任何一个失败都会中断执行，对于成功的也无法获取返回值(除非promise.all内部任务中有对错误做处理)。  
所以allSettled针对promise.all的问题进行了优化，直接返回了对象并告知你执行的结果，还可以对promise任务做错误处理。

const promise1 = new Promise((resolve, reject) => setTimeout(() => resolve(100), 1000));const promise2 = new Promise((resolve, reject) => setTimeout(reject, 1000));Promise.allSettled([promise1, promise2]).then(data => console.log(data)); // [// Object { status: "fulfilled", value: 100},// Object { status: "rejected", reason: undefined}// ]

1. 7. globalThis

在 ES2020 之前，你需要在不同的 JavaScript 环境（跨平台）中编写不同的语法来访问全局对象。window, self, or frames这对开发者来说真的是一段艰难的时期，因为你需要在浏览器端、globalnodejs 端、selfweb workers 端使用。

另一方面，this关键字可以在非严格模式的函数内部使用，但它在严格模式下给出 undefined 。如果您考虑将其Function(‘return this’)()作为上述环境的解决方案，那么对于启用 CSP 的环境（禁用 eval()），它将失败。

在旧版本中，你可以像下面这样使用 es6-shim，

var getGlobal = function () {if (typeof self !== 'undefined') { return self; }if (typeof window !== 'undefined') { return window; }if (typeof global !== 'undefined') { return global; }throw new Error('unable to locate global object');

};var globals = getGlobal();if (typeof globals.setTimeout !== 'function') {console.log('no setTimeout in this environment or runtime');

}

在 ES2020 中，globalThis引入了属性以提供一种跨环境访问全局 this 值的标准方法。

if (typeof globalThis.setTimeout !== 'function') {console.log('no setTimeout in this environment or runtime');

}

1. 8. import.meta
2. 9. for…in order

在 ES2020 之前，规范没有指定 (a in b) 的运行顺序。尽管大多数 javascript 引擎/浏览器按照对象的定义顺序循环访问对象的属性，但并非所有情况都是如此。这在 ES2020 中已经被正式标准化。

var object = {'a': 2,'b': 3,'c': 4

}for(let key in object) {console.log(key); // a b c

}

1. ES10

2019 年 6 月

1. 1. Array flat 和 flatMap

在 ES2019 之前，你需要使用reduce() or concat()方法来获取平面数组。

function flatten(arr) {const flat = [].concat(...arr);return flat.some(Array.isArray) ? flatten(flat) : flat;

}

flatten([ [1, 2, 3], ['one', 'two', 'three', [22, 33] ], ['a', 'b', 'c'] ]);

在 ES2019 中，flat()引入了将嵌套数组“扁平化”为顶级数组的方法。该方法的功能类似于 Lodash 的\_.flattenDepth()功能。此方法接受一个可选参数，该参数指定嵌套数组应展平的级别数，默认嵌套级别为 1。 注意：如果数组中有任何空槽，它们将被丢弃。

const numberArray = [[1, 2], [[3], 4], [5, 6]];const charArray = ['a', , 'b', , , ['c', 'd'], 'e'];const flattenedArrOneLevel = numberArray.flat(1);const flattenedArrTwoLevel = numberArray.flat(2);const flattenedCharArrOneLevel = charArray.flat(1);console.log(flattenedArrOneLevel); // [1, 2, [3], 4, 5, 6]console.log(flattenedArrTwoLevel); // [1, 2, 3, 4, 5, 6]console.log(flattenedCharArrOneLevel); // ['a', 'b', 'c', 'd', 'e']

而flatMap()方法将map()and组合flat()成一个方法。它首先使用给定函数的返回值创建一个新数组，然后连接该数组的所有子数组元素。

const numberArray1 = [[1], [2], [3], [4], [5]];console.log(numberArray1.flatMap(value => [value \* 10])); // [10, 20, 30, 40, 50]

1. 2. Object.fromEntries()

在 JavaScript 中，从一种格式转换数据是很常见的。ES2017 引入了Object.entries()将对象放入数组的方法。  
对象到数组：

const obj = {'a': '1', 'b': '2', 'c': '3' };const arr = Object.entries(obj);console.log(arr); // [ ['a', '1'], ['b', '2'], ['c', '3'] ]

但是如果你想从数组中取回对象，那么你需要迭代并转换它，如下所示

const arr = [ ['a', '1'], ['b', '2'], ['c', '3'] ];

let obj = {}

for (let [key, val] of arr) {obj[key] = val;

}

console.log(obj);

我们需要一种直接的方法来避免这种迭代。在 ES2019 中，Object.fromEntries()引入了执行相反Object.entries()行为的方法。上面的循环可以很容易地避免，如下所示  
可迭代（例如数组或映射）到对象

const arr = [ ['a', '1'], ['b', '2'], ['c', '3'] ];const obj = Object.fromEntries(arr);console.log(obj); // { a: "1", b: "2", c: "3" }

使用此方法的一种常见情况是使用 URL 的查询参数

const paramsString = 'param1=foo&param2=baz';const searchParams = new URLSearchParams(paramsString);Object.fromEntries(searchParams); // => {param1: "foo", param2: "baz"}

1. 3. trimStart和trimEnd

//Prior ES2019let messageOne = " Hello World!! ";console.log(messageOne.trimLeft()); //Hello World!!console.log(messageOne.trimRight()); // Hello World!!//With ES2019let messageTwo = " Hello World!! ";console.log(messageTwo.trimStart()); //Hello World!!console.log(messageTwo.trimEnd()); // Hello World!!

1. 4. Symbol description

在创建符号时，您还可以为其添加描述以用于调试目的。但是在ES2019之前没有直接访问描述的方法。考虑到这一点，ES2019 引入了一个只读描述属性来检索包含符号描述的字符串。

这使得访问符号对象的不同变体的符号描述成为可能

console.log(Symbol('one').description); // oneconsole.log(Symbol.for('one').description); // "one"console.log(Symbol('').description); // ''console.log(Symbol().description); // unefinedconsole.log(Symbol.iterator.description); // "Symbol.iterator"

1. 5. 可选的捕获绑定

在 ES9 之前，如果您不需要error变量并省略相同的变量，则不会调用 catch() 子句。此外，linters 抱怨未使用的变量。为了避免这个问题，引入了可选的catch绑定特性，使catch子句中的绑定参数成为可选的。如果您想完全忽略该错误，或​​者您已经知道该错误但您只想对此做出反应，则此功能将很有用。

让我们看看以下版本之间的语法差异

// With binding parameter(<ES9)try {···} catch (error) {···}// Without binding parameter(ES9)try {···} catch {···}

1. 6. JSON 改进
2. 7. 数组稳定排序
3. 8. function.toString()

函数有一个调用的实例方法toString()，它返回一个字符串来表示函数代码。以前版本的 ECMAScript 从函数代码中删除了空格、换行和注释，但它在 ES2020 中保留了原始源代码。

1. 9. 私有类变量

在 ES6 中，引入了类来创建可重用的模块，并在 clousure 中声明了变量以使其私有化。而在 ES2020 中，引入了私有类变量以仅允许在类中使用变量。通过在我们的变量或函数前面添加一个简单的哈希符号，您可以将它们完全保留给类内部。

class User {#message = "Welcome to ES2020"login() { console.log(this.#message) }}const user = new User()user.login() // Welcome to ES2020console.log(user.#message) // Uncaught SyntaxError: Private field '#

1. ES9

2018 年 6 月发布

1. 1. 异步迭代对象

新增用于遍历异步可迭代的对象，for-await-of  
看一下普通的forof，对于异步任务是直接全部一次性执行完的，无法做到串行的目的。

function getPromise(time) { return new Promise((resolve, reject) => {setTimeout(() => {resolve(time) }, time) })

}

let arr=[getPromise(100),getPromise(200),getPromise(300)]for(const i of arr){console.log(i)

}

// Promise{<pending>}

// Promise{<pending>}

// Promise{<pending>}

异步for-await-of可以做到串行的。

function getPromise(time) { return new Promise((resolve, reject) => {setTimeout(() => {resolve(time) }, time) })

}

let arr=[getPromise(100),getPromise(200),getPromise(300)]for await (const i of arr){console.log(i)

}

//100

//200

//300

1. 2. 对象扩展符

es6支持数组的扩展符，比如以下代码

function myfunc(p1, p2, ...p3) {console.log(p1, p2, p3); // 1, 2, [3, 4, 5, 6]}

myfunc(1, 2, 3, 4, 5, 6);

const myArray = [10, 5, 25, -100, 200, -200];console.log( Math.max(...myArray) ); // 200

现在es9支持对象扩展符

function myfunc1({ a, ...x }) {console.log(a, x); // 1, { b: 2, c: 3, d:4 }}myfunc1({a: 1,b: 2,c: 3,d: 4});

const myObject = { a: 1, b: 2, c: 3, d:4 };

const myNewObject = { ...myObject, e: 5 }; // { a: 1, b: 2, c: 3, d: 4, e: 5 }

1. 3. Promise finally

有时您可能需要避免 then() 和 catch() 方法中的重复代码。

myPromise.then(result => {// process the result and then clean up the resources}).catch(error => {// handle the error and then clean up the resources});

如果您希望在 promise 确定（即履行或拒绝）后进行一些处理或资源清理，则该方法很有用。

let isLoading = true;fetch('http://somesite.com/users').then(data => data.json()).catch(err => console.error(err)).finally(() => {isLoading = false;console.log('Finished loading!!');})

1. ES8

2017 年 6 月发布

1. 1. Async functions

es6为了解决回调地狱的问题，增加了promise函数来解决，但promise又带来了新问题，那就是链条过长，导致难维护。

new Promise((resolve, reject) => {this.login(resolve)})

.then(() => this.getInfo())

.then(() => {// do something})

.catch(() => { console.log("Error") })

于是又诞生了Generator函数，允许按需执行任务，通过yield和next来管理和执行任务，但流程依然不够清晰

const gen = function\* () {

const f1 = yield this.login()

const f2 = yield this.getInfo()

};

现在，es8提供async/await，以其极简的语法来完成异步任务

async function logger() {let data = await fetch('http://someapi.com/users'); // pause until fetch returnsconsole.log(data)

}

logger();

1. 2. Object.values()

与遍历 JavaScript 对象的键的 Object.keys 类似，Object.values 将对值做同样的事情。即，引入了 Object.values() 方法，以与循环相同的顺序返回给定对象自身的可枚举属性值的数组for…in

const countries = {IN: 'India',SG: 'Singapore',}Object.values(countries) // ['India', 'Singapore']

非对象参数将被强制转换为对象

console.log(Object.values(['India', 'Singapore'])); // ['India', 'Singapore']

console.log(Object.values('India')); // ['I', 'n', 'd', 'i', 'a']

1. 3. Object.entries()

引入该Object.entries()方法以返回给定对象自身的可枚举字符串键控属性 [key, value] 对的数组，其顺序与for…in循环相同。

const countries = {IN: 'India',SG: 'Singapore',}Object.entries(countries) // [["IN", "India"], ["SG", "Singapore"]]

非对象参数将被强制转换为对象

const countriesArr = ['India', 'Singapore'];console.log(Object.entries(countriesArr)); // [ ['0', 'India'], ['1', 'Singapore']]const country = 'India';console.log(Object.entries(country)); // [["0", "I"], ["1", "n"], ["2", "d"], ["3", "i"], ["4", "a"]]console.log(Object.entries(100)); // [], an empty array for any primitive type because it won't have any own properties

1. 4. Object property descriptors
2. 5. String padding

一些字符串和数字（​​金钱、日期、计时器等）需要以特定格式表示。引入的这两种padStart() & padEnd()方法都是用另一个字符串填充一个字符串，直到结果字符串达到提供的长度。

padStart()：使用此方法，填充应用于字符串的左侧或开头。  
例如，出于安全原因，您可能只想显示信用卡号的最后四位数字

const cardNumber = '01234567891234';

const lastFourDigits = cardNumber.slice(-4);

const maskedCardNumber = lastFourDigits.padStart(cardNumber.length, '\*');

console.log(maskedCardNumber); // expected output: "\*\*\*\*\*\*\*\*\*\*1234"

padEnd()：使用此方法，填充应用于字符串的右侧或结尾。  
例如，为标签和值填充的​​配置文件信息如下

const label1 = "Name";

const label2 = "Phone Number";

const value1 = "John"

const value2 = "(222)-333-3456";console.log((label1 + ': ').padEnd(20, ' ') + value1); // Name: John

console.log(label2 + ": " + value2); // Phone Number: (222)-333-3456

1. ES7

2016 年 6 月发布

1. 1. includes

在 ES7 之前，您必须使用indexOf方法将结果与​​ ‘-1’ 进行比较，以检查数组元素是否包含特定元素。

const array = [1,2,3,4,5,6];if(array.indexOf(5) > -1 ){console.log("Found an element");}

而在 ES7 中，array.prototype.includes()方法被引入作为一种直接方法来确定数组是否在其条目中包含某个值。

const array = [1,2,3,4,5,6];

if(array.includes(5)){console.log("Found an element");

}

除此之外，它Array.prototype.includes()比方法更好地处理 NaN 和 Undefined 值Array.prototype.indexOf()。即，如果数组包含 NaN 和 Undefined 值，则indexOf()在搜索 NaN 和 Undefined 时不会返回正确的索引。

let numbers = [1, 2, 3, 4, NaN, ,];

console.log(numbers.indexOf(NaN)); // -1

console.log(numbers.indexOf(undefined)); // -1

另一方面，includes方法能够找到这些元素

let numbers = [1, 2, 3, 4, NaN, ,];

console.log(numbers.includes(NaN)); // true

console.log(numbers.includes(undefined)); // true

1. 2. 指数运算符

旧版本的 javascript 使用Math.pow函数来查找给定数字的幂。ECMAScript 2016 引入了求幂运算符 \*\*（类似于其他语言，例如 Python 或 F#），使用中缀表示法以清晰的表示形式计算幂计算。

//Prior ES7

const cube = x => Math.pow(x, 3);

console.log(cube(3)); // 27//Using ES7

const cube1 = x => x \*\* 3;

console.log(cube1(3)); // 27

1. ES6

2015 年 6 月发布

因为tc39早期团队内的分歧严重，导致es6在接近十年后才能发布。

1. 1. 变量作用域

变量作用域确定变量在程序或区域的特定部分内的可见性或可访问性。  
在 ES6 中，关键字const和let关键字都允许开发人员在块范围内声明变量。  
该let语句声明了一个可以重新分配的块作用域局部变量。即，let声明创建一个可变变量。

let a = 1;if (a === 1) {let a = 2;console.log(a); //2

}console.log(a); //1

const变量类似于let变量，但它们不能通过重新分配来更改。即，const 声明创建一个对值的只读引用。

const x = 1;if (x === 1) {const y = 2; // You cannot re-assign the value similar to let variableconsole.log(y); //2

}console.log(x); //1

const和let与var的区别

1. const

**var定义的变量，可以预解析提前调用的结果是undefined，const定义的变量不能预解析，提前调用的结果是 报错。  
var定义的变量，变量名称可以重复，效果是重复赋值，const定义的变量不能重复，否则执行报错。  
var定义的变量作用域是全局/局部作用域。const定义的变量如果在{}中只能在{}中调用。  
const 定义的变量存储的数据数值不能改变，也就是const定义的变量，不能重复赋值。**

1. let

**var定义的变量，可以预解析提前调用的结果是undefined，let定义的变量不能预解析，提前调用的结果是 报错。  
var定义的变量，变量名称可以重复，效果是重复赋值，let定义的变量不能重复，否则执行报错。  
var定义的变量作用域是全局/局部作用域。let定义的变量如果在{}中只能在{}中调用。  
在循环语句中var定义的循环变量和使用let定义的循环变量。执行原理和执行效果不同。**

1. 2. 箭头函数

箭头函数高级特性说明  
箭头函数通过使用粗箭头 (=>) 表示法选择退出函数和返回关键字，为编写函数表达式提供了更简洁的语法。让我们看看这个箭头函数的样子

// Function Expression

var multiplyFunc = function(a, b) {return a \* b;

}

console.log(multiplyFunc(2, 5)); // 10// Arrow function

var multiplyArrowFunc = (a, b) => a \* b;

console.log(multiplyArrowFunc(2, 5)); // 10

如果函数只有一个参数（零个或多个参数），您也可以跳过圆括号 (())。除此之外，如果函数体内有多个表达式，您可以用大括号 ({}) 括起来。

让我们列出箭头函数的所有变体

//1. Single parameter and single statement

const message = name => console.log("Hello, " + name + "!");

message("Sudheer"); // Hello, Sudheer!//2. Multiple parameters and single statement

const multiply = (x, y) => x \* y;

console.log(multiply(2, 5)); // 10//3. Single parameter and multiple statements

const even = number => {if(number%2) {console.log("Even");} else {console.log("Odd");}

}

even(5); // odd//4. Multiple parameters and multiple statements

const divide = (x, y) => {if(y != 0) {return x / y;}

}

console.log(divide(100, 5)); // 20//5. No parameter and single statement

const greet = () => console.log('Hello World!');

greet(); // Hello World!

1. 3. 类

这些类作为现有基于原型的继承和构造函数的语法糖引入。所以这个特性并没有给 JavaScript 带来新的面向对象的继承模型。

有两种定义类的方法

1. 类声明

class Square {constructor(length) {this.length = length;}get area() {return this.length \* this.length;}set area(value) {this.area = value;}

}

1. 类表达式

const square = class Square {  
constructor(length) {  
this.length = length;  
}

get area() {  
return this.length \* this.length;  
}

set area(value) {  
this.area = value;  
}  
}  
您可以使用extend关键字来使用继承。这使子类能够获得父类的所有功能。

class Vehicle {constructor(name) {this.name = name;}start() {console.log(`${this.name} vehicle started`);}

}class Car extends Vehicle {start() {console.log(`${this.name} car started`);}

}const car = new Car('BMW');

console.log(car.start()); // BMW car started

1. 4. 增强的对象字面量

对象文字被扩展以支持在构造时设置原型、foo 的简写：foo 赋值、定义方法、进行超级调用以及使用表达式计算属性名称。

1. 属性简写

对象的属性通常是从具有相同名称的变量创建的。  
让我们看看 ES5 的表示。

var a = 1, b = 2, c = 3,obj = {a: a,b: b,c: c};console.log(obj);

它可以用更短的语法表示，如下所示

const a = 1, b = 2, c = 3;

const obj = {a,b,c

};

console.log(obj);

1. 方法速记

在 ES5 中，对象方法需要如下函数语句

var calculation = {sum: function(a, b) { return a + b; },multiply: function(a, b) { return a \* b; }

};console.log(calculation.sum(5, 3)); // 8

console.log(calculation.multiply(5, 3)); // 15

这在 ES6 中可以避免

const calculation = {sum(a, b) { return a + b; },multiply(a, b) { return a \* b; }

};console.log(calculation.sum(5, 3)); // 8

console.log(calculation.multiply(5, 3)); // 15

在 ES5 中，不可能在对象创建阶段使用变量作为键名。

varkey = 'three',obj = {one: 1,two: 2};obj[key] = 3;

在 ES6 中，可以通过将表达式放在方括号 ([]) 中来动态分配对象键

constkey = 'three',computedObj = {one: 1,two: 2,[key]: 3};

1. 5. 模板字符串

在 ES6 之前，JavaScript 开发人员需要进行丑陋的字符串连接才能创建动态字符串。  
与 ES5 相比，模板文字允许您以一种新的方式处理字符串。这些只是字符串文字，允许嵌入由美元符号和大括号 ( ${expression}) 表示的表达式。此外，这些文字由反引号 ( `) 字符而不是双引号或单引号括起来。

1. 模板字面量

跨多行存在的字符串字面量，包括内插表达式（即 ${expression}）。

const firstName = 'John';

console.log(`Hello ${firstName}!

Good morning!`);

1. 标记的模板文字

通过在模板文字之前提及函数而创建的函数调用。

const Button = styled.a`display: inline-block;border-radius: 3px;

`

1. 6. 解构赋值

解构是一种 javascript 表达式，用于从存储在对象（对象的属性）和数组中的数据中提取多个值。

1. 对象解构

此功能用于从对象中提取值。

const user = { firstName: 'John', lastName: 'Kary' };

const {firstName, lastName} = user;

console.log(firstName, lastName); // John, Kary

1. 数组解构

此功能用于从数组中提取值。

const [one, two, three] = ['one', 'two', 'three'];

console.log(one, two, three); // one, two, three

还可以应用在以下地方

1. 变量声明
2. 对象分配
3. 参数定义
4. for循环
5. 7. 参数默认值

如果没有传递任何值或未定义，则默认参数允许使用默认值初始化函数的命名参数。  
在 ES6 之前，您需要检查未定义值并使用 if/else 或三元运算符为未定义值提供默认值。

function add(a, b) {a = (typeof a !== 'undefined') ? a : 10;b = (typeof b !== 'undefined') ? b : 20;return a + b;

}

add(20); // 40

add(); // 30

在 ES6 中，可以使用默认参数避免这些检查。

function add(a = 10, b = 20) {return a + b;

}

add(20); // 40

add(); // 30

1. 8. 剩余参数

rest 参数用于将不定数量的参数表示为数组。这里的重点是只有函数的最后一个参数可以是“剩余参数”。引入此功能是为了减少由参数引起的样板代码。

function sum(...args) {return args.reduce((previous, current) => {return previous + current;});

}console.log(sum(1, 2, 3)); // 6

console.log(sum(1, 2, 3, 4)); // 10

console.log(sum(1, 2, 3, 4, 5)); // 15

1. 9. 数组伸展符

在函数和构造函数调用中，展开运算符将可迭代值转换为参数。

console.log(Math.max(...[-10, 30, 10, 20])); //30

console.log(Math.max(-10, ...[-50, 10], 30)); //30

在数组文字和字符串中，扩展运算符将可迭代值转换为数组元素。

console.log([1, ...[2,3], 4, ...[5, 6, 7]]); // 1, 2, 3, 4, 5, 6, 7

1. 10. 数组迭代器和forof

String、Array、TypedArray、Map 和 Set 都是内置的可迭代对象，但默认情况下对象不是可迭代的。迭代器是一种循环遍历 JavaScript 中任何集合的新方法。这些对象定义了一个序列，并可能在其终止时返回一个值。迭代器通过具有返回具有两个属性的对象的 next() 方法来实现 Iterator 协议。

**value：迭代序列中的下一个值。  
done：如果序列中的最后一个值已经被消耗，则返回 rue。**

您可以通过在其上定义属性来使对象可迭代Symbol.iterator。

const collection = {one: 1,two: 2,three: 3,[Symbol.iterator]() {const values = Object.keys(this);let i = 0;return {next: () => {return {value: this[values[i++]],done: i > values.length}}};}

};const iterator = collection[Symbol.iterator]();console.log(iterator.next()); // → {value: 1, done: false}

console.log(iterator.next()); // → {value: 2, done: false}

console.log(iterator.next()); // → {value: 3, done: false}

console.log(iterator.next()); // → {value: undefined, done: true}for (const value of collection) {console.log(value);

}

1. 11. Generators

生成器是一个可以在中途停止或暂停然后从停止的地方继续同时保持上下文（在重新进入时保存）的函数。它可以使用一个函数关键字后跟一个星号来定义（即，function\* ()）。  
此函数返回一个迭代器对象，此迭代器的next()方法返回一个对象，该对象具有包含产生值的 value 属性和指示生成器是否已产生其最后一个值的 done 属性。

function\* myGenerator(i) {yield i + 10;yield i + 20;return i + 30;

}const myGenObj = myGenerator(10);console.log(myGenObj.next().value); // 20

console.log(myGenObj.next().value); // 30

console.log(myGenObj.next().value); // 40

1. 12. module

重量级特性  
模块是独立的、可重复使用的代码的小单元，用作 Javascript 应用程序中的构建块。  
在 ES6 之前，JavaScript 中没有原生模块支持。使用了 3 个主要的模块标准。

1. 异步模块定义 (AMD)
2. RequireJS 模块
3. CommonJS 模块（在 Node.js 中使用的 module.exports 和 require 语法）

es6 提供了对模块的内置支持。默认情况下，模块内的所有内容都是私有的，并以严格模式运行。公共变量、函数和类都使用 usingexport语句公开并导入相同的 usingimport语句。

和过去的commonjs规范有什么区别？

1. module属于编译时加载，也就是静态加载，在编译时就能确定模块的依赖关系，以及输入和输出的变量；commonjs属于运行时加载，只有代码在运行时才能确定这些东西。
2. module可以做到tree-shaking，也就是可以加载模块部分用到的内容，commonjs需要加载整个模块，再取内容。
3. module输出的是值的引用，commonjs输出的是值的拷贝。
4. module中的import是异步加载，commonjs中的require是同步加载
5. export声明

// module "my-module.js"const PI = Math.PI;function add(...args) {return args.reduce((num, tot) => tot + num);

}function multiply(...args) {return args.reduce((num, tot) => tot \* num);

}// private function

function print(msg) {console.log(msg);

}export { PI, add, multiply };

// module "my-module.js"export default function add(...args) {return args.reduce((num, tot) => tot + num);

}

1. import声明

// 1. Import an entire module's contents

import \* as name from "my-module";//2.Import a single export from a module

import { export1 } from "my-module";//3.Import multiple exports from a module

import { export1 , export2 } from "my-module";//4.Import default export from a module

import defaultExport from "my-module";//5.Import an export with an alias

import { export1 as alias1 } from "my-module";

1. 13 set

Set 是一个内置对象，用于存储任何类型的唯一值的集合。

let mySet = new Set()mySet.add(1);

mySet.add(2);

mySet.add(2);

mySet.add('some text here');

mySet.add({one: 1, two: 2 , three: 3});

console.log(mySet); // Set [ 1, 2, 'some text here', {one: 1, two: 2 , three: 3} ]

console.log(mySet.size) // 4

console.log(mySet.has(2)); // true

1. 14. map

Map 是元素的集合，其中每个元素都存储为键值对。它可以将对象和原始值作为键或值保存，并按插入顺序迭代其元素。

let typeMap = new Map();var keyObj = {'one': 1}typeMap.set('10', 'string'); // a string key

typeMap.set(10, 'number'); // a numeric key

typeMap.set(true, 'boolean'); // a boolean key

typeMap.set(keyObj, 'object'); // an object keyconsole.log(typeMap.get(10)); // number

console.log(typeMap.get('10')); // string

console.log(typeMap.get(keyObj)); // object

console.log(typeMap.get({'one': 1})); // undefinedconsole.log(typeMap.size ); // 3for(let item of typeMap) {console.log(item);

}for(let item in typeMap) {console.log(item);

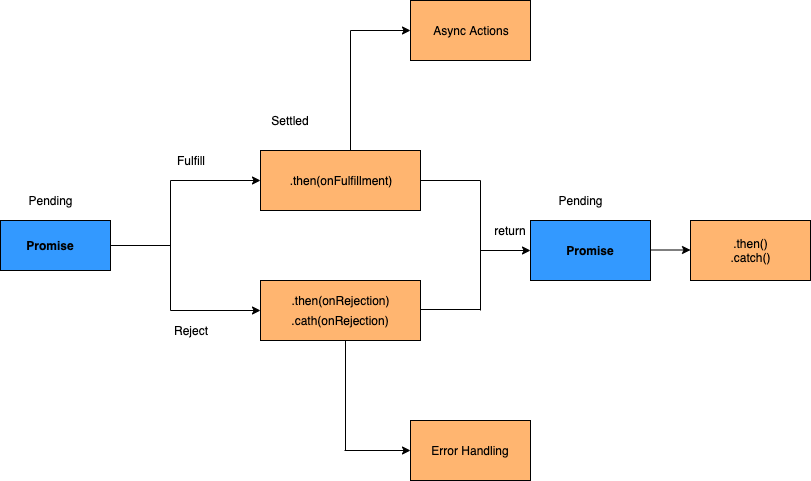
}

1. 15. Symbol
2. 16. promise

Promise 是一个对象，表示异步操作的最终完成或失败。

它处于以下状态之一：

pending：代表初始状态，既没有完成也没有被拒绝。 fulfilled：表示操作成功完成。 rejected：表示操作失败。

如果一个承诺要么被履行要么被拒绝，但不是未决的，就被称为已解决。实例方法promise.then()、promise.catch()和promise.finally()用于将进一步的操作与已确定的承诺相关联。这些方法还返回一个新生成的 promise 对象，它可以选择性地用于链接。  


const promise = new Promise(function(resolve, reject) {setTimeout(() => resolve(1), 1000);

});promise.then(function(result) {console.log(result); // 1return result \* 2;}).then(function(result) {console.log(result); // 2return result \* 3;}).then(function(result) {console.log(result); // 6return result \* 4;}).catch(function(error){console.log(error);});

1. 17. 数组方法

S6 引入了一些数组方法，其中两个是Array.find()和Array.findIndex()。

1. ES5

w3schools  
2009年年中发布

1. 1. “use strict”
2. 2. String[number] access
3. 3. Multiline strings
4. 4. String.trim()
5. 5. Array.isArray()
6. 6. Array forEach()
7. 7. Array map()
8. 8. Array filter()
9. 9. Array reduce()
10. 10. Array reduceRight()
11. 11. Array every()
12. 12. Array some()
13. 13. Array indexOf()
14. 14. Array lastIndexOf()
15. 15. JSON.parse()
16. 16. JSON.stringify()
17. 17. Date.now()
18. 18. Date toISOString()
19. 19. Date toJSON()
20. 20. Property getters and setters
21. 21. Reserved words as property names
22. 22. Object methods
23. 23. Object defineProperty()
24. 24. Function bind()
25. 25. Trailing commas

文章来源:https://blog.csdn.net/qq\_42427109/article/details/130564336