10 ویژگی جدید و جذاب جاوااسکریپت برای عاشقان وب

توسعه‌دهندگانی که از جاوااسکریپت استفاده می‌کنند همواره در تلاش هستند تا کاربردی‌ترین تکنیک‌ها را فراگیرند. ECMAScript 6 یکسری قابلیت‌های کاربردی در اختیار توسعه‌دهندگان قرار داده است که در این مطلب تعدادی از آن‌ها را مورد بررسی قرار می‌دهیم.

کلمات کلیدی const و let

Const به شما اجازه می‌دهد ثابت‌ها را تعریف کنید. let به شما اجازه می‌دهد تا متغیرها را تعریف کنید. این ترکیب عالی است اما سوال مهمی که وجود دارد این است که آیا ما پیش از این قادر نبودیم در جاوااسکریپت متغیرها را تعریف کنیم؟ جواب مثبت است. اما متغیرهایی که همراه با var تعریف می‌شوند دامنه دسترسی وسیعی دارند. این حرف به معنای آن است که یک متغیر را می‌توان پیش از آن‌که تعریف کرد مورد استفاده قرار دارد. اما متغیرهایی که همراه با کلمات کلیدی let و const تعریف می‌شوند این مشکل را ندارند و پیش از تعریف نمی‌توان از آن‌ها استفاده کرد. متغیرهایی که همراه با let و const تعریف می‌شوند دارای بلوکی از کدها هستند که با سمبل‌های {} احاطه شده‌اند و نمی‌توان پیش از تعریف از آن‌ها استفاده کرد.

```
function f() {
  var x = 1
  let y = 2
  const z = 3
  {
    var x = 100
    let y = 200
    const z = 300
    console.log('x in block scope is', x)
    console.log('y in block scope is', y)
    console.log('z in block scope is', z)
  }
  console.log('x outside of block scope is', x)
  console.log('y outside of block scope is', y)
  console.log('z outside of block scope is', z)
}

f()
```

const_let-example.js hosted with ♥ by GitHub

```
x in block scope is 100
y in block scope is 200
z in block scope is 300
x outside of block scope is 100
y outside of block scope is 2
z outside of block scope is 3
```

const_let-output.txt hosted with ♥ by GitHub

مقادیر پیش فرض پارامترها

در ECMAScript5 هیچ راهی وجود نداشت تا بتوانید مقادیر پیش‌فرض را برای پارامترهای توابع تعیین کنید. اما در نسخه جدید این مشکل با اتکا بر یک راه‌کار ساده برای تعریف مقادیر پیش‌فرض برای توابع حل شد. قطعه کد زیر نحوه انجام این‌کار را نشان می‌دهد. اگر به سومین پارامتر تابع test نگاه کنید، مشاهده می‌کنید که مقدار پیش‌فرض پارامتر می‌تواند یک عبارت باشد.

```
function testFunction(x, y = 1, z = x + 1){
     return x / y + z;
};
let result = testFunction(1);
console.log(result); // 3
```

عملگر Spread

یک عملگر گسترشی با سمبل سه نقطه .... نشان داده می‌شود و به منظور جدا کردن یک شی تکرار شونده از

درون مقادیر منحصر به فرد مورد استفاده قرار می‌گیرد. تابع testFunction نحوه به‌کارگیری این تکنیک را نشان می‌دهد.

```
let testFunction = (x, y) => x + y;
let data = [2,3];
let result = testFunction(...data);
console.log(result); // 5
```

با استفاده از عملگر گسترشی می‌توانید مقادیر یک آرایه را به عنوان بخشی از آرایه دیگری ایجاد کنید.

```
let firstArray = [3,4];
let secondArray = [1,2,...firstArray, 5];
console.log(secondArray); // Array [1, 2, 3, 4, 5]
```

Array destructuring assignment

این قابلیت به منظور استخراج مقادیر از درون یک شی تکرار شونده و اختصاص آن مقادیر به متغیرهای منحصر به فرد مورد استفاده قرار می‌گیرد. به‌طور مثال در ECMAScript6 می‌توانید از تکنیک زیر استفاده کنید:

```
let testArray = [1, 2, 3];
let x, y, z;
[x, y, z] = testArray;
console.log(x, y, z); // 1 2 3
```

به جای اختصاص مقادیر به x، y و z می‌توانید از عناصر خاصی از یک آرایه برای این منظور استفاده کنید.

```
let testArray = [1, 2, 3];
let x, y, z;
x = testArray[0];
y = testArray[1];
z = testArray[2];
console.log(x, y, z); // 1 2 3
```

تمام آن کاری که باید انجام دهید این است که متغیرهایی که در نظر دارید مقادیر به آن‌ها تخصیص داده شود را در سمت چپ عملگر تخصیص دهنده قرار دهید.

کلاس‌ها

توسعه‌دهندگان جاوا زمانی که به سراغ پروژه‌های جاوااسکریپت می‌آیند به سرعت نبود کلاس‌ها را احساس می‌کنند.

چه برنامه‌نویسی را سراغ دارید که تمایل نداشته باشد از ارث‌بری در برنامه‌نویسی خود استفاده کند؟ این شکایت توسعه‌دهندگان باعث شد تا کلاس‌ها همراه با ES6 معرفی شوند. در این نسخه تغییری در مفهوم ارث‌بری به وجود نیامد، بلکه ترکیب نحوی برای ارث‌بری پروتوتایپ‌ها ارائه شد.

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

    toString() {
        return '[X=' + this.x + ', Y=' + this.y + ']'
    }
}

class ColorPoint extends Point {
    static default() {
        return new ColorPoint(0, 0, 'black')
    }

    constructor(x, y, color) {
        super(x, y)
        this.color = color
    }

    toString() {
        return '[X=' + this.x + ', Y=' + this.y + ', color=' + this.color + ']'
    }
}

console.log('The first point is ' + new Point(2, 10))
console.log('The second point is ' + new ColorPoint(2, 10, 'green'))
console.log('The default color point is ' + ColorPoint.default())
```

Find

این تابع اولین عنصری که برابر با شرط تعیین شده است را پیدا می‌کند. مقدار برگشتی این تابع برابر با true یا false است.

```
var people = [
  {name: 'Jack', age: 50},
  {name: 'Michael', age: 9},
  {name: 'John', age: 40},
  {name: 'Ann', age: 19},
  {name: 'Elisabeth', age: 16}
]

function teenager(person) {
    return person.age > 10 && person.age < 20
}

var firstTeenager = people.find(teenager)

console.log('First found teenager:', firstTeenager.name)
```

Arrow functions

پیاده‌سازی توابع ساده‌ای همچون sum یا product به کدنویسی‌های تکراری زیادی نیاز دارد. اما برای حل این مشکل می‌توانیم از توابع Arrow Function استفاده کنیم. توابع فوق کدها را کوتاه‌تر می‌کنند.

```
var array = [1, 2, 3, 4]

const sum = (acc, value) => acc + value
const product = (acc, value) => acc * value

var sumOfArrayElements = array.reduce(sum, 0)
var productOfArrayElements = array.reduce(product, 1)
```

مزیت دیگری که این توابع دارند این است که به صورت خطی می‌توان از آن‌ها استفاده کرد که همین موضوع کدنویسی را بیش از پیش ساده می‌کند.

```
var array = [1, 2, 3, 4]

var sumOfArrayElements = array.reduce((acc, value) => acc + value, 0)
var productOfArrayElements = array.reduce((acc, value) => acc * value, 1)
```

Arrow functions را می‌توان به شکل پیچیده‌تری نیز مورد استفاده قرار داد.

```
var array = [1, 2, 3, 4]

const sum = (acc, value) => {
  const result = acc + value
  console.log(acc, ' plus ', value, ' is ', result)
  return result
}

var sumOfArrayElements = array.reduce(sum, 0)
```

الگوی رشته‌ها

چه کسی عاشق نوشتن رشته‌ای از متغیرهای طولانی است. اغلب توسعه‌دهندگان از خواندن رشته‌هایی طولانی از متغیرها بیزار هستند. خوشبختانه ES6 الگوهای ساده‌ای برای به‌کارگیری رشته‌ها همراه با placeholderها را برای متغیرها تعریف کرد.

```
function hello(firstName, lastName) {
  return `Good morning ${firstName} ${lastName}!
How are you?`
}

console.log(hello('Jan', 'Kowalski'))
```

Promise

تابع Promise به شما اطمینان می‌دهد یکسری عملیات ترتیبی را پشت سر هم اجرا کنید. در دنیای برنامه نویسی بعضی مواقع برنامه‌نویسان نمی‌توانند بر روند اجرای ترتیبی عملیات نظارت کنند. در جاوااسکریپت برای حل این مشکل می‌توانید از Promise استفاده کنید. این تابع دارای دو کانال است. کانال اول برای نتایج است و کانال دوم به منظور نشان دادن خطاها است. برای به دست آوردن نتایج، شما باید یک تابع بازگشتی را به عنوان پارامتر تابع then تعیین کنید. برای مدیریت خطاها نیز باید یک تابع بازگشتی را به عنوان پارامتر تابع catch تعیین کنید. (با توجه به این‌که ما از یک تابع بازگشتی استفاده می‌کنیم در هر بار اجرا ممکن است نتایج مختلفی را مشاهده کنید.)

```
function asyncFunc() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
          const result = Math.random();
          result > 0.5 ? resolve(result) : reject('Oppps....I cannot calculate')
        }, 1)
    });
}

for (let i=0; i<10; i++) {
        asyncFunc()
        .then(result => console.log('Result is: ' + result))
        .catch(result => console.log('Error: ' + result))
}
```

تاریخ انتشار:
30 مهر 1396

نشانی منبع:
https://www.shabakeh-mag.com/news/world/10278/10-%D9%88%DB%8C%DA%98%DA%AF%DB%8C-%D8%AC%D8%AF%DB%8C%D8%AF-%D9%88-%D8%AC%D8%B0%D8%A7%D8%A8-%D8%AC%D8%A7%D9%88%D8%A7%D8%A7%D8%B3%DA%A9%D8%B1%DB%8C%D9%BE%D8%AA-%D8%A8%D8%B1%D8%A7%DB%8C-%D8%B9%D8%A7%D8%B4%D9%82%D8%A7%D9%86-%D9%88%D8%A8