30-09-2020

Dev Javascript

Phần 2: Toán tử, câu lệnh điều kiện, vòng lặp, function, HOF, arrow function, call(), apply(), bind() trong Javascript

Nội dung bài viết

Toán tử (operator)

Có đến 10 loại toán tử khác nhau trong Javascript

- Toán tử gán – Assignment operators
- Toán tử so sánh – Comparison operators
- Toán tử toán hạng – Arithmetic operators
- Toán tử bitwase – Bitwise operators
- Toán tử logic – Logical operators
- Toán tử chuỗi – String operators
- Toán tử ba ngôi – Conditional (ternary) operator
- Toán tử phẩy – Comma operator
- Toán tử một ngôi – Unary operators
- Toán tử quan hệ – Relational operators

Toán tử gán – Assignment operators

Toán tử	Ví dụ	Tương đương
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x – y

*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

Toán tử so sánh – Comparison operators

Toán tử	Mô tả
==	bằng giá trị
===	bằng giá trị và kiểu dữ liệu
!=	không bằng giá trị
!==	không bằng giá trị và kiểu dữ liệu
>	lớn hơn
<	bé hơn
>=	lớn hơn hoặc bằng

<=	bé hơn hoặc bằng
?	toán tử ba ngôi

Toán tử toán hạng – Arithmetic operators

Operator	Description
+	cộng
–	trừ
*	nhân
**	lũy thừa(ES2016)
/	chia
%	chia lấy phần dư
++	tăng 1 giá trị
—	giảm 1 giá trị

```
let a = 1
let b = 1
a++
++b
console.log(a) // 2
console.log(b) // 2

// Nhưng lưu ý vị trí đặt ++ hoặc --
let c = 1
let d = 1
console.log(c++) // 1. Vì câu lên sẽ trả về c
console.log(++d) // 2. Câu lệnh sẽ trả về d + 1
```

Toán tử bitwase – Bitwise operators

Toán tử	Tên	Mô tả
&	AND	Nếu cả 2 bit là 1, giá trị trả về là 1, ngược lại trả về 0.
|	OR	Nếu một trong hai bit là 1, giá trị trả về là 1, ngược lại trả về 0.

^	XOR	Nếu hai bit khác nhau, giá trị trả về là 1, ngược lại trả về 0.
~	NOT	Đảo ngược tất cả các bit, 0 thành 1 và 1 thành 0.
<<	Zero fill left shift	Dịch chuyển tất cả các bit sang bên trái.
>>	Signed right shift	Dịch chuyển tất cả các bit sang bên phải ngoại trừ bit đầu tiên.
>>>	Zero fill right shift	Dịch chuyển tất cả các bit sang bên phải.

Ví dụ

Toán tử	Kết quả	Tương đương	Kết quả
5 & 1	1	0101 & 0001	0001
5 | 1	5	0101 | 0001	0101
~ 5	10	~0101	1010
5 << 1	10	0101 << 1	1010
5 ^ 1	4	0101 ^ 0001	0100
5 >> 1	2	0101 >> 1	0010

5 >>> 1	2	0101 >>> 1	0010

Toán tử logic – Logical operators

Toán tử	Mô tả
&&	logic và
||	logic hoặc
!	logic phủ định

Toán tử chuỗi – String operators

Toán tử `+` được dùng để thêm (nối) chuỗi

```
var txt1 = 'John'
var txt2 = 'Doe'
var txt3 = txt1 + ' ' + txt2 // John Doe
```

Toán tử gán `+=` cũng được dùng để thêm (nối) chuỗi

```
var txt1 = 'What a very '
txt1 += 'nice day' // What a very nice day
```

Toán tử ba ngôi – Conditional (ternary) operator

```
function getFee(isMember) {
  return isMember ? '$2.00' : '$10.00'
}
console.log(getFee(true))
// expected output: "$2.00"
console.log(getFee(false))
// expected output: "$10.00"
```

Toán tử phẩy – Comma operator

Toán tử phẩy đánh giá từng toán hạng từ trái sang phải và trả về toán hạng cuối cùng

```
let x = 1
x = (x++, x)
```

```
console.log(x)
// expected output: 2

x = (2, 3)
console.log(x)
// expected output: 3
```

Toán tử một ngôi – Unary operators

Toán tử một ngôi là phép toán chỉ có duy nhất một toán hạng.

```
// Toán tử delete xoá một object, một thuộc tính của object, hoặc một phần tử ở c
delete objectName
delete objectName.property
delete objectName[index]


// Toán tử typeof trả về một chuỗi ký tự thể hiện kiểu của toán hạng
var myFun = new Function('5 + 2')
var shape = 'round'
```

```
typeof myFun // trả về "function"
typeof shape // trả về "string"


<!-- Toán tử void xác định biểu thực cần thực hiện mà không trả về giá trị nào --
<!-- Đoạn code sẽ tiến hành hoàn tất mẫu đơn khi người dùng bấm vào siêu liên kết
<a href="javascript:void(document.form.submit())">Click here to submit</a>
```

Toán tử quan hệ – Relational operators

Toán tử quan hệ so sánh các toán hạng của nó và trả về giá trị **Boolean** tuỳ thuộc phép so sánh có true hay không.

```
// Toán tử in trả về true nếu thuộc tính nhất định có trong object nhất định
var trees = ['redwood', 'bay', 'cedar', 'oak', 'maple']
0 in trees // trả về true
var mycar = { make: 'Honda', model: 'Accord', year: 1998 }
'make' in mycar // trả về true


// Toán tử instanceof trả về true nếu một object nhất định có kiểu của object nhấ
```

```
var theDay = new Date(1995, 12, 17)

if (theDay instanceof Date) {
  // lệnh sẽ được thực thi
}
```

Câu lệnh điều kiện

```
// if else
let a = 2
if (a > 0) {
  console.log('a lớn hơn 0')
} else {
  console.log('a bé hơn 0')
}

// switch case
let step = 1
switch (step) {
```

```
  case 0:
    console.log('a = 0')
    break
  case 1:
    console.log('a = 1')
  default:
    break
}
```

Vòng lặp

Vòng `while` kiểm tra điều kiện trước khi thực hiện code bên trong

```
let a = 1
let sum = 0
while (a <= 10) {
  sum += a
  a++
}
console.log(sum) // 55
```

Vòng `do-while` thực hiện code trước rồi mới kiểm tra điều kiện cho đoạn code tiếp theo

```
let a = 1
let sum = 0
do {
  sum += a
  a++
} while (a <= 10)
console.log(sum) // 55
```

Vòng `for`

```
let sum = 0
for (let i = 1; i <= 10; i++) {
  sum += i
}
console.log(sum) // 55
```

Vòng `for-in` dùng lặp các key của object

```
let person = { fname: 'John', lname: 'Doe', age: 25 }
let text = ''
for (let x in person) {
  text += person[x] + ' '
}
console.log(text) // John Doe 25
```

Vòng `for-of` dùng lặp các value của interable object điển hình như array hay string

```
let people = ['Nick', 'Alan', 'Ben']
let text = ''
for (let value of people) {
  text += value + ' '
}
console.log(text) // Nick Alan Ben
```

Hàm (function)

Một Javasript function là một đoạn code được thiết kế để làm một nhiệm vụ riêng biệt.

Javascript function được thực thi khi gọi nó.

Hàm trong Javascript chia làm 2 loại

1. Khai báo hàm (Function Declaration)
2. Biểu thức hàm (Function Expression)

Khai báo hàm (Function Declaration)

Với cách này thì function được **Hoisting**. Và Javascript cho phép chúng ta gọi một hàm trước khi hàm đó được khai báo.

```
hoisted(); // Output: "This function has been hoisted."
function hoisted() {
  console.log('This function has been hoisted.');
};
```

Biểu thức hàm (Function Expression)

Tuy nhiên với cách khai báo function kiểu này thì sẽ không được hoisting

```
expression(); //Output: "TypeError: expression is not a function
var expression = function() {
```

```
  console.log('Will this work?');
};
```

Giải thích: Biến `var expression` vẫn được **hoisting** và được đẩy lên trên cùng của scope nhưng chỉ là khai báo mà thôi, nó không được gán cho hàm! Vì thế nó sẽ ném ra lỗi `TypeError` .

IIFE (Immediately Invokable Function Expression)

IIFE là khởi tạo một function và thực thi ngay lập tức sau đó.

```
(function () {
  let a = 1
  let b = 2
  console.log('a + b = ' + (a + b))
})()
```

Hàm ẩn danh (Anonymous function)

Hàm ẩn danh là hàm không tên. Nếu bạn để ý thì vế bên phải biểu thức hàm là một **anonymous function**, hay **IIFE** cũng thực thi một hàm ẩn danh. Ngoài ra hàm ẩn danh còn xuất hiện ở **callback**

function bên trong `setTimeout` là một hàm ẩn danh

```
setTimeout(function() {
  console.log('Sau 1s thì sẽ in ra dòng này')
}, 1000);
```

Hàm rút gọn (Arrow function)

Hàm rút gọn ngắn hơn biểu thức hàm (function expression) và không phụ thuộc `this`. Áp dụng tốt cho hàm ẩn danh (anonymous function) nhưng không thể dùng làm hàm khởi tạo

```
const handleClick = () => {
  // thực hiện gì đó
}
```

Phân biệt parameter (tham số) vs argument (đối số)

```
// a, b là tham số
function sum(a, b) {
  return a + b
```

```
}
// 1,2 là đối số
sum(1, 2)
```

Tham số mặc định (default parameter)

```
function gx(x, y = x) {
  console.log(x, y)
}
gx(3) // 3 3
gx(3, 5) // 3 5
```

Rest parameter

```
function sum(...theArgs) {
  console.log(theArgs)
}
console.log(sum(1, 2, 3)) // [1, 2, 3]
console.log(sum(1, 2, 3, 4)) // [1, 2, 3, 4]
```

Từ khóa this

`this` trong Javascript là từ khóa để cập đến object mà nó thuộc về.

this trong một phương thức (method)

Trong phương thức, `this` đề cập đến **object chủ quản**

```
var person = {
  firstName: 'John',
  lastName: 'Doe',
  id: 5566,
  fullName: function() {
    return this.firstName + ' ' + this.lastName
  },
}
console.log(person.fullName()) //John Doe
```

this đứng một mình

Khi đứng một mình, `this` đề cập đến **global object**. Nếu là trình duyệt thì sẽ là **[object Window]**

```
var x = this;
console.log(this) //[object Window]
```

this ở trong một function

Nếu mặc định thì `this` sẽ đề cập đến **global object**

```
function myFunction() {
  return this
}
console.log(myFunction()) // [object Window]
```

Nếu trong chế độc `'strict mode'` thì sẽ là `undefined`

```
"use strict";
function myFunction() {
  return this
}
```

```
console.log(myFunction()) // undefined
```

this ở trong một Event Handler

Trong một HTML event handler, `this` đề cập đến HTML element mà nó nhận event.

Khi nhấn vào button dưới đây thì nó sẽ được set `display:none`

```
<button onclick="this.style.display='none'">Click to Remove Me!</button>
```

this ở trong callback

`this` trong đoạn code này sẽ đề cập đến `setTimeout`

```
function delay() {
  this.name = 'Duoc'
  setTimeout(function() {
    console.log(this.name) // undefined
  }, 1000)
}
delay()
```

để fix vấn đề này thì có thể dùng **arrow function**

```
function delay() {
  this.name = 'Duoc'
  setTimeout(() => {
    console.log(this.name) // Duoc
  }, 1000)
}
delay()
```

call, apply, bind

Chỉ có các function mới có thể gọi 3 hàm này.

call

Gọi hàm và cho phép bạn truyền vào một object và các đối số phân cách nhau bởi dấu phẩy

apply

Tương tự call. Gọi hàm, cho phép truyền vào một object và tiếp theo là các đối số thông qua mảng

bind

Trả về một hàm mới. Cho phép truyền vào một object và các đối số phân cách nhau bởi dấu phẩy

Khi dùng 3 hàm này, `this` sẽ đề cập đến object được truyền vào.

```
let person = { name: 'Duoc', age: 24 }
function say(text1, text2) {
  console.log(text1 + ' ' + text2 + ' ' + this.name + ' ' + this.age)
}

say.call(person, 'Xin', 'chao') // Xin chao Duoc 24
say.apply(person, ['Xin', 'chao']) // Xin chao Duoc 24
const newSay = say.bind(person, 'Xin', 'chao')
newSay() // Xin chao Duoc 24
```

Higher order function

Mình đã viết 1 bài chi tiết tại đây: Chinh phục High Order Function, Closures, Currying và Callback trong Javascript

High order function là một **function** mà nhận vào tham số là **function** hoặc **return** về một **function**

```
const tinhTong = a => b => a + b
const ketQua = [1, 2, 3, 4, 5].map((item) => item * item)
console.log(tinhTong(1)(2)) // 3
console.log(ketQua) // [ 1, 4, 9, 16, 25 ]
```

Callback function

Callback function là một **function** mà được truyền vào một **function khác** như một tham số

```
const num = [2, 4, 6, 8];
num.forEach((item, index) => {
  console.log("STT: ", index, "la ", item);
});
const result = num.map((item, index) => `STT: ${index} la ${item}`);
```

Closure

Closure là cách mà một **function cha** return về một **function con** bên trong nó. Ở trong **function con** đó có thể truy cập và thực thi các biến của **function cha.** Phải đủ 2 điều kiện này mới được gọi là **Closure** nhé.

```
const increase = () => {
  let x = 0;
  const increaseInner = () => ++x;
  return increaseInner;
};
const myFunc = increase();
console.log(increase()()); // 1
console.log(increase()()); // 1
console.log(myFunc()); // 1
console.log(myFunc()); // 2
console.log(myFunc()); // 3
```

Currying

Currying là một kỹ thuật mà cho phép chuyển đổi một **function nhiều tham số** thành những **function liên tiếp có một tham số**.

```
const findNumber = (num) => (func) => {
  const result = [];
  for (let i = 0; i < num; i++) {
    if (func(i)) {
      result.push(i);
    }
  }
  return result;
};
findNumber(10)((number) => number % 2 === 1);
findNumber(20)((number) => number % 2 === 0);
findNumber(30)((number) => number % 3 === 2);
```

Tham khảo

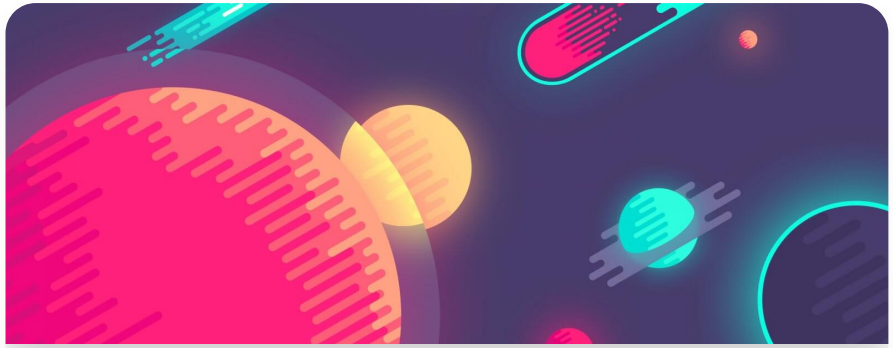
https://www.facebook.com/groups/AngularVietnam

https://www.w3schools.com/

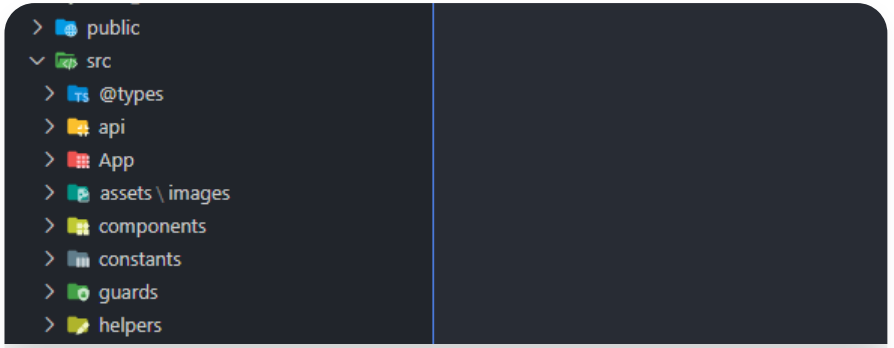
https://developer.mozilla.org/

Mình đã kiếm được 1000$ đầu tiên trong 3 ngày từ lập...

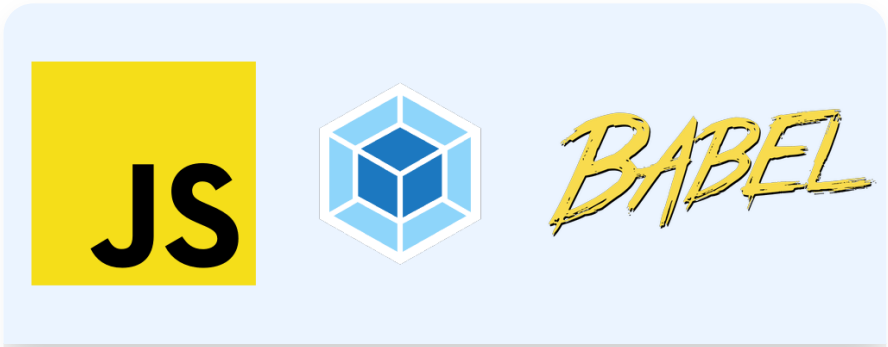
Các nguồn thu nhập của lập trình viên: Bài viết dành...



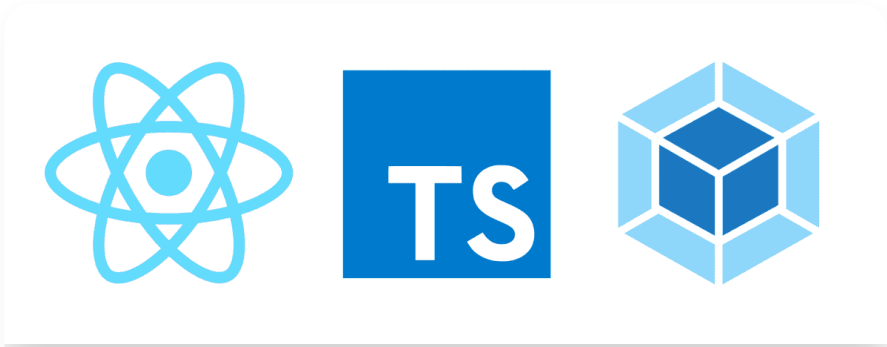
Khóa học Javascript từ căn bản đến nâng cao - Xây...



Cấu trúc React Folder tối ưu, dễ bảo trì, dễ nâng cấp



Webpack siêu tốc 1: Cấu hình Dev Server, Babel...

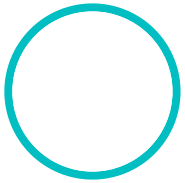


Webpack siêu tốc 3: Cấu hình React, Typescript,...

Share Article:

https://xdevclass.com/phan-2-toan-tu-cau-lenl

Javascript siêu tốc



Dư Thanh Được

Chào mọi người, mình là Founder Xdevclass - Lập trình tinh gọn. Thế mạnh UX/UI, Front-End. Thích đọc tin tức về tech & marketing. Thích chia sẻ về lập trình website và phát triển bản thân. Donate Momo: 0768447467

Bắt đầu bình luận nào

2 BÌNH LUẬN

trưởng

hay ạ!

0 Trả lời

Dư Thanh Được Author

Reply to trưởng

Cảm ơn bro nah 😂

0 Trả lời

Về XdevClass

XdevClass là trang blog được thành lập bởi Thanh Được chuyên về các chủ đề lập trình và phát triển bản thân. Với phương châm lập trình tinh gọn, blog luôn hướng đến cách viết ngắn gọn, dễ hiểu và thực tế.

Bài viết gần đây

11-02-2021

Khóa học Javascript từ căn bản đến nâng cao – Xây dựng 10 project với JS thuần – Kiến thức đầy đủ để bắt đầu với mọi Javascript Framework

02-03-2021

Object methods và “this” trong Javascript, có bài tập thực hành

26-02-2021

Property flags và descriptors – Bí thuật bên trong Javascript Object

17-02-2021

Cuối cùng chúng ta code vì điều gì? Các ngã rẽ của nghề dev

Fanpage