

[Công nghệ ▾](#)[Tài liệu tham khảo & hướng dẫn ▾](#)[Phản hồi ▾](#)[Đăng nhập !\[\]\(d66ff64371a51729ac8c1cdaa685ba6f_img.jpg\)](#)

delete operator

Bản dịch này chưa hoàn thành. Xin hãy giúp dịch bài viết này từ tiếng Anh.

Toán tử `delete` của JavaScript loại bỏ một thuộc tính khỏi object; nếu không tồn tại tham chiếu tới thuộc tính, nó sẽ tự động giải phóng.

Khảo sát MDN



Hãy giúp chúng tôi hiểu 10 nhu cầu hàng đầu của các nhà phát triển Web và nhà thiết kế.

Tham gia khảo sát



JavaScript Demo: Expressions - delete operator

```
1 var Employee = {  
2   firstname: "John",  
3   lastname: "Doe"  
4 }  
5  
6 console.log(Employee.firstname);  
7 // expected output: "John"  
8  
9 delete Employee.firstname;  
10  
11 console.log(Employee.firstname);  
12 // expected output: undefined  
13
```

Run ›

Reset

Khảo sát MDN



Hãy giúp chúng tôi hiểu 10 nhu cầu hàng đầu của các nhà phát triển Web và nhà thiết kế.

Tham gia khảo sát

với *expression* thực thi thành tham chiếu đến thuộc tính nào đó, từ



```
delete object.property  
delete object['property']
```

Tham số [↗](#)

object

Tên object, hoặc biểu thức thực thi tới object.

property

Thuộc tính muốn xoá.

Giá trị trả về [↗](#)

true cho mọi trường hợp trừ khi thuộc tính là own non-configurable, trong trường hợp đó, trả về false trong chế độ non-strict.

Ngoại lệ [↗](#)

Quăng **TypeError** trong chế độ strict nếu thuộc tính là own non-configurable.

Khảo sát MDN



Hãy giúp chúng tôi hiểu 10 nhu cầu hàng đầu của các nhà phát triển Web và nhà thiết kế.

Tham gia khảo sát

thêm chi tiết.



Toán tử **delete** loại bỏ thuộc tính xác định trong object. Nếu xoá thành công, nó sẽ trả về `true`, ngoài ra thì `false`. Tuy nhiên, hãy lưu ý những kịch bản có thể xảy đến sau đây:

- Nếu thuộc tính muốn xoá không tồn tại, `delete` sẽ không có tác dụng và sẽ trả về `true`
- Nếu tồn tại thuộc tính có cùng tên trong prototype nối với object, thì sau khi xoá xong, object sẽ dùng thuộc tính từ prototype đó (nói cách khác, `delete` chỉ có tác dụng với những thuộc tính của riêng object).
- Bất cứ thuộc tính nào được khởi tạo bằng **var** không thể bị xoá khỏi phạm vi toàn cục hoặc phạm vi hàm.
 - Ví thế, `delete` không thể xoá bất cứ hàm nào trong phạm vi toàn cục (cho dù là một phần của định nghĩa hàm hay biểu thức hàm).
 - Các hàm là một phần của object (tách biệt với phạm vi toàn cục) có thể bị xoá với `delete`.
- Bất cứ thuộc tính nào được khởi tạo bởi **let** hoặc **const** không thể bị xoá khỏi phạm vi mà chúng được khai báo.
- Thuộc tính không-thể-cấu-hình không thể bị loại bỏ. Các thuộc tính này bao gồm các object dựng sẵn như `Math`, `Array`, `Object` và thuộc tính được tạo ra như thuộc tính không-thể-cấu-hình bằng phương thức như là `Object.defineProperty()`.

Khảo sát MDN



Hãy giúp chúng tôi hiểu 10 nhu cầu hàng đầu của các nhà phát triển Web và nhà thiết kế.

Tham gia khảo sát

```
6  
7 console.log(delete Employee.name); // trả về true  
8 console.log(delete Employee.age); // trả về true  
9  
10 // Khi cố xoá một thuộc tính không tồn tại  
11 // sẽ trả về giá trị true  
12 console.log(delete Employee.salary); // trả về true
```



Thuộc tính không-thể-cấu-hình

Khi một thuộc tính được đánh dấu không-thể-cấu-hình, delete không có tác dụng nào, và sẽ trả về false. Trong chế độ strict, lỗi TypeError sẽ nhảy ra.

```
1 var Employee = {};  
2 Object.defineProperty(Employee, 'name', {configurable: false});  
3  
4 console.log(delete Employee.name); // trả về false
```

var, **let** và **const** tạo ra thuộc tính không-thể-cấu-hình mà không thể xoá bằng toán tử delete:

Khảo sát MDN



Hãy giúp chúng tôi hiểu 10 nhu cầu hàng đầu của các nhà phát triển Web và nhà thiết kế.

Tham gia khảo sát

```
7 // ...
8 //      enumerable: true,
9 //      configurable: false}
10
11 // Bởi vì "nameOther" được thêm vào nhờ dùng
12 // từ khoá var, nên nó được đánh dấu là "không-thể-cấu-hình"
13
14 delete nameOther; // trả về false
```

Trong chế độ strict, ngoại lệ sẽ quăng ra.

Chế độ strict và non-strict [↗](#)

Khi ở trong chế độ strict, nếu delete được dùng để tham chiếu trực tiếp tới một biến, một đối số của hàm hoặc tên hàm, nó sẽ quăng ra **SyntaxError**.

Bất cứ biến nào được định nghĩa với var đều được đánh dấu là không-thể-cấu-hình. Trong ví dụ sau đây, salary là không-thể-cấu-hình và không thể xoá. Trong chế độ non-strict, phép toán delete sẽ trả về false.

```
1 function Employee() {
2   delete salary;
```

Khảo sát MDN



Hãy giúp chúng tôi hiểu 10 nhu cầu hàng đầu của các nhà phát triển Web và nhà thiết kế.

Tham gia khảo sát



```
1  "use strict";
2
3  function Employee() {
4      delete salary; // SyntaxError
5      var salary;
6  }
7
8  // Tương tự, bất cứ truy nhập trực tiếp nào vào hàm
9  // dùng delete đều quăng ra SyntaxError
10
11 function DemoFunction() {
12     //vài đoạn code
13 }
14
15 delete DemoFunction; // SyntaxError
```

Ví dụ

Khảo sát MDN



Hãy giúp chúng tôi hiểu 10 nhu cầu hàng đầu của các nhà phát triển Web và nhà thiết kế.

Tham gia khảo sát

```
8
9 EmployeeDetails = {
10   name: 'xyz',
11   age: 5,
12   designation: 'Developer'
13 };
14
15 // adminName là thuộc tính trên phạm vi toàn cục.
16 // Nó có thể bị xoá bởi được khởi tạo mà không dùng var,
17 // và vì thế khả cấu.
18 delete adminName;      // trả về true
19
20 // Ngược lại, empCount không khả cấu
21 // bởi dùng var.
22 delete empCount;       // trả về false
23
24 // Có thể dùng delete để loại bỏ thuộc tính khỏi object.
25 delete EmployeeDetails.name; // trả về true
26
27 // Thậm chí thuộc tính không tồn tại, delete vẫn trả về "true".
28 delete EmployeeDetails.salary; // trả về true
29
30 // delete không có tác dụng với thuộc tính dựng sẵn.
```

Khảo sát MDN



Hãy giúp chúng tôi hiểu 10 nhu cầu hàng đầu của các nhà phát triển Web và nhà thiết kế.

Tham gia khảo sát


```
38 var z = 44;  
39  
40 // delete không có tác dụng với tên biến cục bộ  
41 delete z;    // trả về false  
42 }
```



delete và prototype chain

Trong ví dụ sau, ta sẽ xoá một thuộc tính riêng của object mà vẫn tồn tại thuộc tính cùng tên trong prototype chain:

```
1 function Foo() {  
2   this.bar = 10;  
3 }  
4  
5 Foo.prototype.bar = 42;  
6  
7 var foo = new Foo();  
8  
9 // foo.bar liên kết với  
10 // thuộc tính riêng.  
11 console.log(foo.bar); // 10
```

Khảo sát MDN



Hãy giúp chúng tôi hiểu 10 nhu cầu hàng đầu của các nhà phát triển Web và nhà thiết kế.

Tham gia khảo sát



```
20
21 // Xoá thuộc tính trên prototype.
22 delete Foo.prototype.bar; // trả về true
23
24 // Thuộc tính "bar" không còn có thể
25 // kế thừa từ Foo bởi nó đã bị xoá
26 console.log(foo.bar); // undefined
```

Xoá phần tử mảng

Khi bạn xoá phần tử mảng, độ dài mảng không bị ảnh hưởng. Thậm chí khi bạn xoá phần tử cuối của mảng cũng không thay đổi được điều này.

Khi toán tử `delete` loại bỏ một phần tử mảng, phần tử đó không còn trong mảng. Trong ví dụ sau, `trees[3]` bị xoá bởi `delete`.

```
1 var trees = ['redwood', 'bay', 'cedar', 'oak', 'maple'];
2 delete trees[3];
3 if (3 in trees) {
4     // không thực thi đoạn code này
5 }
```

Khảo sát MDN



Hãy giúp chúng tôi hiểu 10 nhu cầu hàng đầu của các nhà phát triển Web và nhà thiết kế.

Tham gia khảo sát


```
3 | 1+ (3 in trees) {  
4 |     // đoạn code trong này sẽ chạy  
5 | }
```



Thay vì thế, nếu muốn loại bỏ phần tử mảng bằng cách thay đổi nội dung của mảng, hãy dùng phương thức `splice`. Trong ví dụ sau, `trees[3]` bị xoá bỏ hoàn toàn khỏi mảng thông qua `splice`:

```
1 | var trees = ['redwood', 'bay', 'cedar', 'oak', 'maple'];  
2 | trees.splice(3,1);  
3 | console.log(trees); // ["redwood", "bay", "cedar", "maple"]
```

Đặc Đặc tả

Specification	Status	Comment
ECMAScript Latest Draft (ECMA-262) The definition of 'The delete Operator' in that specification.	 Draft	

Khảo sát MDN



Hãy giúp chúng tôi hiểu 10 nhu cầu hàng đầu của các nhà phát triển Web và nhà thiết kế.

Tham gia khảo sát

The definition of 'The delete Operator' in that specification.



Trình duyệt hỗ trợ [↗](#)

[Update compatibility data on GitHub](#)

delete

Chrome	Yes
Edge	Yes
Firefox	1
IE	Yes
Opera	Yes
Safari	Yes
WebView Android	Yes
Chrome Android	Yes

Khảo sát MDN



Hãy giúp chúng tôi hiểu 10 nhu cầu hàng đầu của các nhà phát triển Web và nhà thiết kế.

[Tham gia khảo sát](#)

Temporal dead zone



Chrome	?
Edge	?
Firefox	36
IE	?
Opera	?
Safari	?
WebView Android	?
Chrome Android	?
Firefox Android	36
Opera Android	?
Safari iOS	?
Samsung Internet Android	?
nodejs	?



Full support

Khảo sát MDN



Hãy giúp chúng tôi hiểu 10 nhu cầu hàng đầu của các nhà phát triển Web và nhà thiết kế.

Tham gia khảo sát

Ghi chú Cross-browser



Although ECMAScript makes iteration order of objects implementation-dependent, it may appear that all major browsers support an iteration order based on the earliest added property coming first (at least for properties not on the prototype). However, in the case of Internet Explorer, when one uses `delete` on a property, some confusing behavior results, preventing other browsers from using simple objects like object literals as ordered associative arrays. In Explorer, while the property *value* is indeed set to undefined, if one later adds back a property with the same name, the property will be iterated in its *old* position--not at the end of the iteration sequence as one might expect after having deleted the property and then added it back.

If you want to use an ordered associative array in a cross-browser environment, use a `Map` object if available, or simulate this structure with two separate arrays (one for the keys and the other for the values), or build an array of single-property objects, etc.

Xem thêm

- [Phân tích sâu về delete](#)

Khảo sát MDN



Hãy giúp chúng tôi hiểu 10 nhu cầu hàng đầu của các nhà phát triển Web và nhà thiết kế.

Tham gia khảo sát