The arguments object

`arguments` là object giống `Array` khả truy cập bên trong hàm, chứa các giá trị của đối số truyền vào trong hàm đó.

> **Ghi chú:** Nếu bạn viết mã tương thích cho ES6, thì bạn nên tham khảo rest parameters.

> **Ghi chú:** "Giống Array" tức là `arguments` có thuộc tính `length` và đánh chỉ mục từ không (0), nhưng nó không có phương thức dựng sẵn của `Array` như `forEach()` và `map()`. Xem §Mô tả để biết thêm chi tiết.

JavaScript Demo: Functions Arguments

```
function func1(a, b, c) {
  console.log(arguments[0]);
  // expected output: 1

  console.log(arguments[1]);
  // expected output: 2

  console.log(arguments[2]);
  // expected output: 3
}

func1(1, 2, 3);

```

Run ›

Reset

Cú pháp

```
arguments
```

Mô tả

Object `arguments` là biến cục bộ có sẵn cho mọi hàm không phải hàm mũi tên. Bạn có thể tham chiếu tới đối số của một hàm bên trong hàm đó bằng cách sử dụng object `arguments`. Nó chứa từng đối số mà hàm đó được gọi cùng, với chỉ mục bắt đầu từ 0.

Chẳng hạn, nếu một hàm được truyền vào 3 đối số, bạn có thể truy cập nó theo cách sau:

```
arguments[0] // đối số thứ nhất
arguments[1] // đối số thứ hai
arguments[2] // đối số thứ ba
```

Mỗi đối số đều có thể thiết lập hoặc gán lại:

```
arguments[1] = 'new value';
```

Object `arguments` không phải là `Array`. Nó tương tự nhưng không có thuộc tính của `Array` ngoài `length`. Chẳng hạn, nó không có phương thức `pop()`. Tuy nhiên, nó có thể ép kiểu về `Array`:

```
var args = Array.prototype.slice.call(arguments);
// Using an array literal is shorter than above but allocates an empty array
var args = [].slice.call(arguments);
```

Như có thể làm với mọi object giống-Array, bạn có thể dùng phương thức `Array.from()` của ES2015 hoặc cú pháp spread để ép kiểu `arguments` thành Array:

```
var args = Array.from(arguments);
var args = [...arguments];
```

Object `arguments` có ích đối với hàm được truyền nhiều đối số hơn so với khởi tạo ban đầu. Kỹ thuật này có ích với hàm cần được truyền nhiều biến, như là `Math.min()`. Hàm ví dụ dưới đây chấp nhật mọi xâu ký tự và trả về xâu dài nhất:

```
function longestString() {
  var longest = '';
  for (var i=0; i < arguments.length; i++) {
    if (arguments[i].length > longest.length) {
      longest = arguments[i];
    }
  }
  return longest;
}
```

Bạn có thể dùng `arguments.length` để đếm số lượng đối số được truyền vào khi hàm được gọi. Thay vì thế, nếu bạn muốn đếm số lượng tham số chính quy mà hàm chấp nhận khi khởi tạo, hãy tham khảo thuộc tính `length` của hàm.

Sử dụng `typeof` với Arguments

Toán tử `typeof` trả về `'object'` khi dùng với `arguments`

```
1 | console.log(typeof arguments); // 'object'
```

Kiểu của từng đối số có thể xác định lần lượt bằng cách chỉ đích danh trong object `arguments`:

```
1 | console.log(typeof arguments[0]); // trả về kiểu của đối số thứ nhất
```

Thuộc tính

`arguments.callee`
Tham chiếu tới hàm đang được thực thi sở hữu arguments.

`arguments.caller`
Reference to the function that invoked the currently executing function.

`arguments.length`
Số lượng đối số được truyền vào hàm.

`arguments[@@iterator]`
Returns a new `Array` iterator object that contains the values for each index in the `arguments`.

Ví dụ

Định nghĩa hàm nối xâu ký tự

Ví dụ sau đây định nghĩa một hàm nối các xâu ký tự với nhau. Tham số chính quy mà hàm nhận là một xâu chứa các ký tự ngăn cách các xâu với nhau sau khi được nối.

```
function myConcat(separator) {
  var args = Array.prototype.slice.call(arguments, 1);
  return args.join(separator);
}
```

Bạn có thể truyền vào bao nhiêu xâu ký tự tuỳ ý. Giá trị trả về là một xâu chứa tất cả các đối số được truyền vào:

```
// trả về "red, orange, blue"
myConcat(', ', 'red', 'orange', 'blue');

// trả về "elephant; giraffe; lion; cheetah"
myConcat('; ', 'elephant', 'giraffe', 'lion', 'cheetah');

// trả về "sage. basil. oregano. pepper. parsley"
myConcat('. ', 'sage', 'basil', 'oregano', 'pepper', 'parsley');
```

Định nghĩa hàm sinh danh sách HTML

Ví dụ sau đây định nghĩa một hàm sinh ra một xâu ký tự chứa các thẻ HTML để tạo thành một danh sách. Tham số chính quy duy nhất mà hàm nhận là một ký tự như "u" nếu danh sách không có thứ tự (đánh dấu chấm), hay "o" nếu danh sách có thứ tự (đánh số). Hàm đó được định nghĩa như sau:

```
function list(type) {
  var html = '<' + type + 'l><li>';
  var args = Array.prototype.slice.call(arguments, 1);
  html += args.join('</li><li>');
  html += '</li></' + type + 'l>'; // end list

  return html;
}

```

Bạn có thể truyền bao nhiêu đối số tuỳ ý, và nó sẽ thêm từng đối số vào danh sách có kiểu xác định trước. Chẳng hạn:

```
var listHTML = list('u', 'One', 'Two', 'Three');

/* listHTML is:
"<ul><li>One</li><li>Two</li><li>Three</li></ul>"
*/
```

Tham số rest, default và destructured

Object `arguments` có thể dùng cùng lúc với các tham số như rest, default, và destructured.

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

Tuy trong strict-mode, tham số rest, default, hoặc destructured parameters không can thiệp vào hành vi của object `arguments`, nhưng trong non-strict mode vẫn có không ít khác biệt.

Nếu một hàm non-strict **không** chứa tham số rest, default, hay destructured, thì giá trị trong object `arguments` **có** thay đổi đồng bộ với giá trị của tham số truyền vào. Hãy xem đoạn mã dưới đây:

```
function func(a) {
  arguments[0] = 99; // cập nhật arguments[0] cũng cập nhật a
  console.log(a);
}
func(10); // 99
```

và

```
function func(a) {
  a = 99; // cập nhật a cũng cập nhật arguments[0]
  console.log(arguments[0]);
}
func(10); // 99
```

Khi hàm non-strict **có** chứa tham số rest, default hoặc destructured, thì giá trị trong object `arguments` **không** theo dõi giá trị của đối số. Thay vì vậy, chúng ánh xạ đến đối số truyền vào khi hàm được gọi:

```
function func(a = 55) {
  arguments[0] = 99; // updating arguments[0] does not also update a
  console.log(a);
}
func(10); // 10
```

và

```
function func(a = 55) {
  a = 99; // updating a does not also update arguments[0]
  console.log(arguments[0]);
}
func(10); // 10
```

và

```
// An untracked default parameter
function func(a = 55) {
  console.log(arguments[0]);
}
func(); // undefined
```

Đặc tả kĩ thuật

Đặc tả	Trạng thái	Ghi chú
ECMAScript 1st Edition (ECMA-262)	ST Standard	Initial definition. Implemented in JavaScript 1.1
ECMAScript 5.1 (ECMA-262) The definition of 'Arguments Object' in that specification.	ST Standard	
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'Arguments Exotic Objects' in that specification.	ST Standard	
ECMAScript Latest Draft (ECMA-262) The definition of 'Arguments Exotic Objects' in that specification.	D Draft	

Tương thích trình duyệt

Update compatibility data on GitHub

`arguments`

Chrome	Yes
Edge	Yes
Firefox	1
IE	Yes
Opera	Yes

Safari	Yes
WebView Android	Yes
Chrome Android	Yes
Firefox Android	4
Opera Android	Yes
Safari iOS	Yes
Samsung Internet Android	Yes
nodejs	Yes

`callee`

Chrome	Yes
Edge	Yes
Firefox	1
IE	6
Opera	Yes
Safari	Yes
WebView Android	Yes
Chrome Android	Yes
Firefox Android	4
Opera Android	Yes
Safari iOS	Yes

Samsung Internet Android	Yes
nodejs	Yes

`caller`

Chrome	No
Edge	No
Firefox	No
IE	? — 9
Opera	No
Safari	No
WebView Android	No
Chrome Android	No
Firefox Android	No
Opera Android	No
Safari iOS	No
Samsung Internet Android	No
nodejs	No

`length`

Chrome	Yes
Edge	Yes
	1

IE	Yes
Opera	Yes
Safari	Yes
WebView Android	Yes
Chrome Android	Yes
Firefox Android	4
Opera Android	Yes
Safari iOS	Yes
Samsung Internet Android	Yes
nodejs	Yes

`@@iterator`

Chrome	52
Edge	?
Firefox	46
IE	No
Opera	Yes
Safari	9
WebView Android	52
Chrome Android	52
Firefox Android	46

Opera Android	Yes
Safari iOS	9
Samsung Internet Android	6.0
nodejs	Yes

Flag as incorrect

Full support

No support

Compatibility unknown

Non-standard. Expect poor cross-browser support.

Deprecated. Not for use in new websites.

Xem thêm

- `Function`
- Rest parameters