Une fonction

Chaque fonction JavaScript est en fait un `Function` objet. Cela peut être vu avec le code `(function(){}).constructor === Function`, qui renvoie true.

Constructeur

`Function()`
Crée un nouvel `Function` objet. Appeler directement le constructeur peut créer des fonctions dynamiquement, mais souffre de problèmes de sécurité et de performances similaires (mais beaucoup moins importants) à `eval()`. Cependant, contrairement à `eval()`, le `Function` constructeur crée des fonctions qui s'exécutent uniquement dans la portée globale.

Propriétés d'occurrence

`Function.prototype.arguments`
Un tableau correspondant aux arguments passés à une fonction. Ceci est obsolète en tant que propriété de **`Function`**. Utilisez `arguments` plutôt l'objet (disponible dans la fonction).

`Function.prototype.caller`
Spécifie la fonction qui a appelé la fonction en cours d'exécution. Cette propriété est obsolète et n'est fonctionnelle que pour certaines fonctions non strictes.

`Function.prototype.displayName`
Nom d'affichage de la fonction.

`Function.prototype.length`
Spécifie le nombre d'arguments attendus par la fonction.

`Function.prototype.name`
Le nom de la fonction.

Méthodes d'instance

`Function.prototype.apply(thisArg [, argsArray])`
Appelle une fonction et la définit `this` sur le fourni `thisArg`. Les arguments peuvent

être passés en tant `Array` qu'objet.

`Function.prototype.bind(thisArg[, arg1[, arg2[, ...argN]]])`
Crée une nouvelle fonction qui, lorsqu'elle est appelée, est `this` définie sur le fourni `thisArg`. Facultativement, une séquence donnée d'arguments sera ajoutée aux arguments à condition que la fonction nouvellement liée soit appelée.

`Function.prototype.call(thisArg[, arg1, arg2, ...argN])`
Appelle une fonction et la définit `this` sur la valeur fournie. Les arguments peuvent être passés tels quels.

`Function.prototype.toString()`
Renvoie une chaîne représentant le code source de la fonction. Remplace la `Object.prototype.toString` méthode.

Exemples

Différence entre le constructeur de fonction et la déclaration de fonction

Les fonctions créées avec le `Function` constructeur ne créent pas de fermetures à leurs contextes de création ; ils sont toujours créés dans la portée globale. Lors de leur exécution, ils ne pourront accéder qu'à leurs propres variables locales et globales, pas à celles de la portée dans laquelle le `Function` constructeur a été créé. Ceci est différent de l'utilisation `eval()` avec du code pour une expression de fonction.

```
var x = 10;

function createFunction1() {
    var x = 20;
    return new Function('return x;'); // this |x| refers global |x|
}

function createFunction2() {
    var x = 20;
    function f() {
        return x; // this |x| refers local |x| above
    }
    return f;
}

var f1 = createFunction1();
console.log(f1());          // 10
var f2 = createFunction2();
```

```
console.log(f2());          // 20
```

Bien que ce code fonctionne dans les navigateurs Web, `f1()` il produira un `ReferenceError` dans Node.js, car `x` il ne sera pas trouvé. En effet, la portée de niveau supérieur dans Node n'est pas la portée globale et `x` sera locale au module.

Caractéristiques

spécification
Spécification du langage ECMAScript # sec-function-objects

Compatibilité du navigateur

Signaler les problèmes avec ces données de compatibilité sur GitHub

`Function`	
Chrome	1
Bord	12
Firefox	1
Internet Explorer	4
Opéra	3
Safari	1
WebAfficher Android	1
ChromeAndroid	18
Firefox pour Androïd	4
Opéra Android	10.1
Safari sur iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0
`Function()` constructeur	
Chrome	1

Edge	12
Firefox	1
Internet Explorer	4
Opera	3
Safari	1
WebView Android	1
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`apply`	
Chrome	1
Edge	12
Firefox	1
Internet Explorer	5.5
Opera	4
Safari	1
WebView Android	1
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

ES 5.1 : objet de type tableau générique en tant que `arguments`

Chrome	17
Edge	12
Firefox	4
Internet Explorer	9
Opera	5
Safari	6
WebView Android	37
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	6
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`arguments`

Chrome	1
Edge	12
Firefox	1
Internet Explorer	4
Opera	3
Safari	1
WebView Android	1
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`bind`	
Chrome	7
Edge	12
Firefox	4
Internet Explorer	9
Opera	11.6
Safari	5.1
WebView Android	4
Chrome Android	18
Firefox for Android	4
Opera Android	12
Safari on iOS	6
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`call`	
Chrome	1
Edge	12
Firefox	1
Internet Explorer	5.5
Opera	4
Safari	1
WebView Android	1
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0

Node.js	0.10.0

`caller`

Chrome	1
Edge	12
Firefox	1
Internet Explorer	8
Opera	9.6
Safari	3
WebView Android	1
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`displayName`

Chrome	**Non**
Edge	**Non**
Firefox	13
Internet Explorer	**Non**
Opera	**Non**
Safari	**Non**
WebView Android	**Non**
Chrome Android	**Non**
Firefox for Android	14
Opera Android	**Non**
Safari on iOS	**Non**
Samsung Internet	**Non**

Deno	**Non**
Node.js	**Non**
`length`	
Chrome	1
Edge	12
Firefox	1
Internet Explorer	4
Opera	3
Safari	1
WebView Android	1
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0
Configurable : vrai	
Chrome	43
Edge	12
Firefox	37
Internet Explorer	**Non**
Opera	30
Safari	dix
WebView Android	43
Chrome Android	43
Firefox for Android	37
Opera Android	30
Safari on iOS	dix

Samsung Internet	4.0
Deno	1.0
Node.js	4.0.0

`name`

Chrome	15
Edge	14
Firefox	1
Internet Explorer	**Non**
Opera	10.5
Safari	6
WebView Android	1
Chrome Android	18
Firefox for Android	4
Opera Android	11
Safari on iOS	6
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

Configurable : vrai

Chrome	43
Edge	14
Firefox	38
Internet Explorer	**Non**
Opera	30
Safari	dix
WebView Android	43
Chrome Android	43
Firefox for Android	38
Opera Android	30

Safari on iOS	dix
Samsung Internet	4.0
Deno	1.0
Node.js	4.0.0
Noms inférés sur les fonctions anonymes	
Chrome	51
Edge	14
Firefox	53
Internet Explorer	**Non**
Opera	38
Safari	dix
WebView Android	51
Chrome Android	51
Firefox for Android	53
Opera Android	41
Safari on iOS	dix
Samsung Internet	5.0
Deno	1.0
Node.js	6.5.0
`toSource`	
Chrome	**Non**
Edge	**Non**
Firefox	**1 – 74**
Internet Explorer	**Non**
Opera	**Non**
Safari	**Non**
WebView Android	**Non**
Chrome Android	**Non**
Firefox for Android	4

Opera Android	**Non**
Safari on iOS	**Non**
Samsung Internet	**Non**
Deno	**Non**
Node.js	**Non**

`toString`	
Chrome	1
Edge	12
Firefox	1
Internet Explorer	5
Opera	3
Safari	1
WebView Android	1
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`Function.prototype.toString` **Révision des outils**	
Chrome	**Non**
Edge	**Non**
Firefox	54
Internet Explorer	**Non**
Opera	**Non**
Safari	**Non**
WebView Android	**Non**
Chrome Android	**Non**

Firefox for Android	54
Opera Android	**Non**
Safari on iOS	**Non**
Samsung Internet	**Non**
Deno	**Non**
Node.js	**Non**

 Plein soutien

 Prise en charge partielle

 Pas de support

Non standard. Vérifiez la prise en charge de plusieurs navigateurs avant de l'utiliser.

Obsolète. Ne pas utiliser dans les nouveaux sites Web.

Voir les notes d'implémentation.

L'utilisateur doit explicitement activer cette fonctionnalité.

Voir également

- Fonctions et étendue des fonctions
- `function` déclaration
- `function` expression
- `function*` déclaration
- `function*` expression

- `function*` expression
- `AsyncFunction`
- `GeneratorFunction`

Dernière modification : 23 janvier 2022, par les contributeurs MDN