

Function

Function 构造函数 创建一个新的Function对象。在 JavaScript 中, 每个函数实际上都是一个Function对象。

句法

```
new Function ([arg1[, arg2[, ...argN]],] functionBody)
```

参数

arg1, arg2, ... argN

被函数使用的参数的名称必须是合法命名的。参数名称是一个有效的JavaScript标识符的字符串, 或者一个用逗号分隔的有效字符串的列表;例如“x”, “theValue”, 或“A, B”。


functionBody

一个含有包括函数定义的JavaScript语句的字符串。

描述

使用Function构造器生成的 **Function**对象是在函数创建时解析的。这比你使用[函数声明](#)或者[函数表达式\(function\)](#)并在你的代码中调用更为低效, 因为使用后者创建的函数是跟其他代码一起解析的。

所有被传递到构造函数中的参数, 都将被视为将被创建的函数的参数, 并且是相同的标示符名称和传递顺序。

 **注意:** 使用Function构造器生成的函数, 并不会在创建它们的上下文中创建闭包; 它们一般在全局作用域中被创建。当运行这些函数的时候, 它们只能访问自己的本地变量和全局变量, 不能访问Function构造器被调用生成的上下文的作用域。这和使用带有函数表达式代码的 `eval` 不同。

以调用函数的方式调用Function的构造函数 (不是用new关键字) 跟以构造函数来调用是一样的。

属性和方法

全局的Function对象没有自己的属性和方法, 但是, 因为它本身也是函数, 所以它也会通过原型链从 `Function.prototype` 上继承部分属性和方法。

原型对象

属性

`Function.arguments`

以数组形式获取传入函数的所有参数。此属性已被 `arguments` 替代。

`Function.arity`

用于指定的函数的参数的个数, 但已被删除。使用 `length` 属性代替。

`Function.caller`

获取调用函数的具体对象。

`Function.length`

获取函数的接收参数个数。

`Function.name`

获取函数的名称。

`Function.displayName`

获取函数的display name。

`Function.prototype.constructor`

声明函数的原型构造方法, 详细请参考 `Object.constructor` 。

方法

`Function.prototype.apply()`

在一个对象的上下文中应用另一个对象的方法; 参数能够以数组形式传入。

`Function.prototype.bind()`

`bind()` 方法会创建一个新函数, 称为绑定函数。当调用这个绑定函数时, 绑定函数会以创建它时传入 `bind()` 方法的第一个参数作为 `this`, 传入 `bind()` 方法的第二个以及以后的参数加上绑定函数运行时本身的参数按照顺序作为原函数的参数来调用原函数。

`Function.prototype.call()`

在一个对象的上下文中应用另一个对象的方法; 参数能够以列表形式传入。

`Function.prototype.isGenerator()`

若函数对象为 `generator`, 返回 `true`, 反之返回 `false` 。

Function.prototype.toSource() ⚠

获取函数的实现源码的字符串。覆盖了 `Object.prototype.toSource` 方法。

Function.prototype.toString()

获取函数的实现源码的字符串。覆盖了 `Object.prototype.toString` 方法。

实例

`Function` 实例从 `Function.prototype` 继承了一些属性和方法。同其他构造函数一样，你可以改变构造函数的原型从而使得所有的`Function`实例的属性和方法发生改变。

示例

传入参数调用Function构造函数

下面的代码会创建一个需要两个参数的`Function`对象

```
1 // 可以直接运行在 JavaScript 控制的代码例子
2
3 // 创建了一个能返回两个参数和的函数
4 const adder = new Function("a", "b", "return a + b");
5
6 // 调用函数
7 adder(2, 6);
8 // 8
```

参数"a"和"b"是参数的名字，在函数体中被使用，`"return a + b"`。

大量修改DOM元素的递归快捷方式

使用 `Function` 构造器创建函数是从一个函数中动态地创建一些不确定数量的有可执行代码的在全局范围里可用的新对象的方式之一。在下面的例子中，如果你不想使用闭包，那么每创建一个新的查询函数都不可避免地要调用 `Function` 构造器。

```
1 <!doctype html>
2 <html>
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
5 <title>MDN Example - a recursive shortcut to massively modify the DOM</title>
6 <script type="text/javascript">
7   var domQuery = (function() {
```

```
8   var aDOMFunc = [
9       Element.prototype.removeAttribute,
10      Element.prototype.setAttribute,
11      CSSStyleDeclaration.prototype.removeProperty,
12      CSSStyleDeclaration.prototype.setProperty
13  ];
14
15  function setSomething (bStyle, sProp, sVal) {
16      var bSet = Boolean(sVal), fAction = aDOMFunc[bSet | bStyle << 1],
17          aArgs = Array.prototype.slice.call(arguments, 1, bSet ? 3 : 2),
18          aNodeList = bStyle ? this.cssNodes : this.nodes;
19
20      if (bSet && bStyle) { aArgs.push(""); }
21      for (
22          var nItem = 0, nLen = this.nodes.length;
23          nItem < nLen;
24          fAction.apply(aNodeList[nItem++], aArgs)
25      );
26      this.follow = setSomething.caller;
27      return this;
28  }
29
30  function setStyles (sProp, sVal) { return setSomething.call(this, true, sProp, sVal); }
31  function setAttribs (sProp, sVal) { return setSomething.call(this, false, sProp, sVal); }
32  function getSelectors () { return this.selectors; }
33  function getNodes () { return this.nodes; }
34
35  return (function (sSelectors) {
36      var oQuery = new Function('return arguments.callee.follow.apply(arguments, arguments);');
37      oQuery.selectors = sSelectors;
38      oQuery.nodes = document.querySelectorAll(sSelectors);
39      oQuery.cssNodes = Array.prototype.map.call(oQuery.nodes, function (oNode) {
40          oNode.attributes = setAttribs;
41          oNode.inlineStyle = setStyles;
42          oNode.follow = getNodes;
43          oNode.toString = getSelectors;
44          oNode.valueOf = getNodes;
45          return oNode;
46      });
47  })();
48 </script>
49 </head>
50
```

2017/10/13Function - JavaScript | MDN

```
51 <body>
52
53 <div class="testClass">Lorem ipsum</div>
54 <p>Some text</p>
55 <div class="testClass">dolor sit amet</div>
56
57 <script type="text/javascript">
58     domQuery('.testClass')
59         .attributes('lang', 'en')('title', 'Risus abundat in ore stultorum')
60         .inlineStyle('background-color', 'black')('color', 'white')('width',
61 </script>
62 </body>
63
64 </html>
```

规范

Specification	Status	Comment
ECMAScript 1st Edition (ECMA-262)	<div>ST</div> Standard	Initial definition. Implemented in JavaScript 1.0.
ECMAScript 5.1 (ECMA-262) Function	<div>ST</div> Standard	
ECMAScript 2015 (6th Edition, ECMA-262) Function	<div>ST</div> Standard	

浏览器支持

	Desktop	Mobile			
Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Basic support	(Yes)	(Yes)	(Yes)	(Yes)	(Yes)

相关链接

- Functions and function scope
- Function
- function statement

- `function expression`
- `function* statement`
- `function* expression`
- `GeneratorFunction`
- CSP: <https://developers.google.com/web/fundamentals/security/csp/#eval>
- CSP: <https://gist.github.com/xgqfrms-GitHub/ecf7733d066d56723b00de41a849037a>

