**javascript中Object使用详解**

2016年04月20日 15:07:11 阅读数：1797

Object.getPrototypeOf(object):调用对象父类原型上的方法;

制代码

function Person(){

this.method1 = function(){alert(1)}

}

Person.prototype.method2 = function(){alert(2);}

function Man(){

this.m1 = function(){

Object.getPrototypeOf(this).method1();

}

}

Man.prototype = new Person();

Man.prototype.m2 = function(){

Object.getPrototypeOf(this).method2();

}

var man = new Man();

man.m1();

man.m2();

制代码

Object.getOwnPropertyDescriptor(object, propertyname):获取对象中属性的ECMAScript对象;

制代码

var obj = {};

obj.a = "abc";

var descriptor = Object.getOwnPropertyDescriptor(obj, "a");

for(var prop in descriptor){

document.write(prop + ': ' + descriptor[prop]);

document.write("<br />");

}

/\*

configurable: true

enumerable: true

value: abc

writable: true

\*/

制代码

Object.defineProperty(object, propertyname, descriptor):将ECMAScript对象设置为对象中的属性.

制代码

var obj = {};

obj.a = "abc";

var descriptor = Object.getOwnPropertyDescriptor(obj, "a");

descriptor.writable = false;

Object.defineProperty(obj, "a", descriptor);

for(var prop in descriptor){

document.write(prop + ': ' + descriptor[prop]);

document.write("<br />");

}

/\*

configurable: true

enumerable: true

value: abc

writable: false

\*/

制代码

object.defineProperties(object, descriptors):用 ECMAScript对象 设置为object中多个属性的值.

制代码

var obj = {};

obj.a = "abc";

Object.defineProperties(obj,{

a:{

configurable: true,

enumerable: true,

value: 'aaa',

writable: false

}

});

var descriptor = Object.getOwnPropertyDescriptor(obj, "a");

for(var prop in descriptor){

document.write(prop + ': ' + descriptor[prop]);

document.write("<br />");

}

/\*

configurable: true

enumerable: true

value: aaa

writable: false

\*/

制代码

Object.getOwnPropertyNames(object):返回一个由对象属性名组成的数组(包含不可枚举的)

制代码

function a(){

this.a='1';

}

a.prototype.b='2';

var c=new a();

c.c='3';

alert(Object.getOwnPropertyNames(c));//a,c

制代码

Object.create(prototype, descriptors):建立一个原型为[prototype](必需,可为NULL),[descriptors](可选)为ECMAScript对象的对象.

制代码

var a = Object.create({a:1,b:2}, {

c: {

value: "large",

enumerable: true

},

d: {

value: "round",

enumerable: true

}

});

制代码

Object.seal(object):锁定对象,无法修改对象的属性,无法加入新的属性.并把ECMAScript对象的configurable设置为false;

制代码

var obj = {};

obj.a = "abc";

Object.seal(obj);

var descriptor = Object.getOwnPropertyDescriptor(obj, "a");

for(var prop in descriptor){

document.write(prop + ': ' + descriptor[prop]);

document.write("<br />");

}

/\*

configurable: false

enumerable: true

value: abc

writable: true

\*/

制代码

Object.freeze(object):冻结对象,无法修改对象的属性,无法加入新的属性.

(与seal的区别为,freeze会把对象的数据属性的Writable设置为false)

制代码

var obj = {};

obj.a = "abc";

Object.freeze(obj);

var descriptor = Object.getOwnPropertyDescriptor(obj, "a");

for(var prop in descriptor){

document.write(prop + ': ' + descriptor[prop]);

document.write("<br />");

}

/\*

configurable: false

enumerable: true

value: abc

writable: false

\*/

制代码

Object.preventExtensions(object):避免加新属性加入对象(Extensible设置为false);

制代码

var obj = { a: "1"};

Object.preventExtensions(obj);

document.write(Object.isExtensible(obj));//false

obj.newProp = 50;

document.write(obj.newProp);//undefined

制代码

Object.isSealed(object);

Object.isFrozen(object);

Object.isExtensible(object);  
  
判断对象是否为锁定,冻结,不可扩展的.(如果一个对象是冻结的,那其肯定是密封的);

制代码

var obj = { a: "1"};

//Object.seal(obj);

Object.freeze(obj);

//Object.preventExtensions(obj);

alert(Object.isSealed(obj));

//alert(Object.isFrozen(obj));

//alert(Object.isExtensible(obj));

//自己慢慢调吧...

制代码

Object.keys(object):返回一个由对象可枚举的属性组成的数组.

function a(){

this.a='1';

}

var b=new a();

alert(Object.keys(b));//a

function Person(){

this.method1 = function(){alert(1)}

}

Person.prototype.method2 = function(){alert(2);}

function Man(){

this.m1 = function(){

Object.getPrototypeOf(this).method1();

}

}

Man.prototype = new Person();

Man.prototype.m2 = function(){

Object.getPrototypeOf(this).method2();

}

var man = new Man();

man.m1();

man.m2();

undefined

Man

ƒ Man(){

this.m1 = function(){

Object.getPrototypeOf(this).method1();

}

}

man

Man {m1: ƒ}m1: ƒ ()arguments: nullcaller: nulllength: 0name: ""prototype: {constructor: ƒ}\_\_proto\_\_: ƒ ()[[FunctionLocation]]: VM105:7[[Scopes]]: Scopes[1]\_\_proto\_\_: Personm2: ƒ ()arguments: nullcaller: nulllength: 0name: ""prototype: {constructor: ƒ}\_\_proto\_\_: ƒ ()[[FunctionLocation]]: VM105:13[[Scopes]]: Scopes[1]method1: ƒ ()\_\_proto\_\_: Objectmethod2: ƒ ()constructor: ƒ Person()\_\_proto\_\_: Object

var obj = {};

obj.a = "abc";

var descriptor = Object.getOwnPropertyDescriptor(obj, "a");

for(var prop in descriptor){

document.write(prop + ': ' + descriptor[prop]);

document.write("<br />");

}

undefined

obj.a

"abc"

descriptor

{value: "abc", writable: true, enumerable: true, configurable: true}configurable: trueenumerable: truevalue: "abc"writable: true\_\_proto\_\_: constructor: ƒ Object()hasOwnProperty: ƒ hasOwnProperty()isPrototypeOf: ƒ isPrototypeOf()propertyIsEnumerable: ƒ propertyIsEnumerable()toLocaleString: ƒ toLocaleString()toString: ƒ toString()valueOf: ƒ valueOf()\_\_defineGetter\_\_: ƒ \_\_defineGetter\_\_()\_\_defineSetter\_\_: ƒ \_\_defineSetter\_\_()\_\_lookupGetter\_\_: ƒ \_\_lookupGetter\_\_()\_\_lookupSetter\_\_: ƒ \_\_lookupSetter\_\_()get \_\_proto\_\_: ƒ \_\_proto\_\_()set \_\_proto\_\_: ƒ \_\_proto\_\_()

var obj = {};

obj.a = "abc";

var descriptor = Object.getOwnPropertyDescriptor(obj, "a");

descriptor.writable = false;

Object.defineProperty(obj, "a", descriptor);

for(var prop in descriptor){

document.write(prop + ': ' + descriptor[prop]);

document.write("<br />");

}

undefined

descriptor

{value: "abc", writable: false, enumerable: true, configurable: true}configurable: trueenumerable: truevalue: "abc"writable: false\_\_proto\_\_: constructor: ƒ Object()hasOwnProperty: ƒ hasOwnProperty()isPrototypeOf: ƒ isPrototypeOf()propertyIsEnumerable: ƒ propertyIsEnumerable()toLocaleString: ƒ toLocaleString()toString: ƒ toString()valueOf: ƒ valueOf()\_\_defineGetter\_\_: ƒ \_\_defineGetter\_\_()\_\_defineSetter\_\_: ƒ \_\_defineSetter\_\_()\_\_lookupGetter\_\_: ƒ \_\_lookupGetter\_\_()\_\_lookupSetter\_\_: ƒ \_\_lookupSetter\_\_()get \_\_proto\_\_: ƒ \_\_proto\_\_()set \_\_proto\_\_: ƒ \_\_proto\_\_()

var obj = {};

obj.a = "abc";

var descriptor = Object.getOwnPropertyDescriptor(obj, "a");

descriptor.writable = false;

for(var prop in descriptor){

document.write(prop + ': ' + descriptor[prop]);

document.write("<br />");

}

undefined

descriptor

{value: "abc", writable: false, enumerable: true, configurable: true}configurable: trueenumerable: truevalue: "abc"writable: false\_\_proto\_\_: constructor: ƒ Object()hasOwnProperty: ƒ hasOwnProperty()isPrototypeOf: ƒ isPrototypeOf()propertyIsEnumerable: ƒ propertyIsEnumerable()toLocaleString: ƒ toLocaleString()toString: ƒ toString()valueOf: ƒ valueOf()\_\_defineGetter\_\_: ƒ \_\_defineGetter\_\_()\_\_defineSetter\_\_: ƒ \_\_defineSetter\_\_()\_\_lookupGetter\_\_: ƒ \_\_lookupGetter\_\_()\_\_lookupSetter\_\_: ƒ \_\_lookupSetter\_\_()get \_\_proto\_\_: ƒ \_\_proto\_\_()set \_\_proto\_\_: ƒ \_\_proto\_\_()

var obj = {};

obj.a = "abcdd";

var descriptor = Object.getOwnPropertyDescriptor(obj, "a");

descriptor.writable = false;

for(var prop in descriptor){

document.write(prop + ': ' + descriptor[prop]);

document.write("<br />");

}

undefined

descriptor

{value: "abcdd", writable: false, enumerable: true, configurable: true}configurable: trueenumerable: truevalue: "abcdd"writable: false\_\_proto\_\_: constructor: ƒ Object()hasOwnProperty: ƒ hasOwnProperty()isPrototypeOf: ƒ isPrototypeOf()propertyIsEnumerable: ƒ propertyIsEnumerable()toLocaleString: ƒ toLocaleString()toString: ƒ toString()valueOf: ƒ valueOf()\_\_defineGetter\_\_: ƒ \_\_defineGetter\_\_()\_\_defineSetter\_\_: ƒ \_\_defineSetter\_\_()\_\_lookupGetter\_\_: ƒ \_\_lookupGetter\_\_()\_\_lookupSetter\_\_: ƒ \_\_lookupSetter\_\_()get \_\_proto\_\_: ƒ \_\_proto\_\_()set \_\_proto\_\_: ƒ \_\_proto\_\_()

var obj = {};

obj.a = "abc";

Object.defineProperties(obj,{

a:{

configurable: true,

enumerable: true,

value: 'aaa',

writable: false

}

});

var descriptor = Object.getOwnPropertyDescriptor(obj, "a");

for(var prop in descriptor){

document.write(prop + ': ' + descriptor[prop]);

document.write("<br />");

}

undefined

var obj = {};

obj.a = "abc";

var descriptor ={}

descriptor.writable = false;

descriptor= Object.getOwnPropertyDescriptor(obj, "a");

for(var prop in descriptor){

document.write(prop + ': ' + descriptor[prop]);

document.write("<br />");

}

undefined

descriptor

{value: "abc", writable: true, enumerable: true, configurable: true}configurable: trueenumerable: truevalue: "abc"writable: true\_\_proto\_\_: constructor: ƒ Object()hasOwnProperty: ƒ hasOwnProperty()isPrototypeOf: ƒ isPrototypeOf()propertyIsEnumerable: ƒ propertyIsEnumerable()toLocaleString: ƒ toLocaleString()toString: ƒ toString()valueOf: ƒ valueOf()\_\_defineGetter\_\_: ƒ \_\_defineGetter\_\_()\_\_defineSetter\_\_: ƒ \_\_defineSetter\_\_()\_\_lookupGetter\_\_: ƒ \_\_lookupGetter\_\_()\_\_lookupSetter\_\_: ƒ \_\_lookupSetter\_\_()get \_\_proto\_\_: ƒ \_\_proto\_\_()set \_\_proto\_\_: ƒ \_\_proto\_\_()

function a(){

this.a='1';

}

a.prototype.b='2';

var c=new a();

c.c='3';

alert(Object.getOwnPropertyNames(c));//a,c

undefined

var a = Object.create({a:1,b:2}, {

c: {

value: "large",

enumerable: true

},

d: {

value: "round",

enumerable: true

}

});

undefined

a

{c: "large", d: "round"}c: "large"d: "round"\_\_proto\_\_: a: 1b: 2\_\_proto\_\_: Object

var obj = {};

obj.a = "abc";

Object.seal(obj);

var descriptor = Object.getOwnPropertyDescriptor(obj, "a");

for(var prop in descriptor){

document.write(prop + ': ' + descriptor[prop]);

document.write("<br />");

}

undefined

obj

{a: "abc"}a: "abc"\_\_proto\_\_: constructor: ƒ Object()hasOwnProperty: ƒ hasOwnProperty()isPrototypeOf: ƒ isPrototypeOf()propertyIsEnumerable: ƒ propertyIsEnumerable()toLocaleString: ƒ toLocaleString()toString: ƒ toString()valueOf: ƒ valueOf()\_\_defineGetter\_\_: ƒ \_\_defineGetter\_\_()\_\_defineSetter\_\_: ƒ \_\_defineSetter\_\_()\_\_lookupGetter\_\_: ƒ \_\_lookupGetter\_\_()\_\_lookupSetter\_\_: ƒ \_\_lookupSetter\_\_()get \_\_proto\_\_: ƒ \_\_proto\_\_()set \_\_proto\_\_: ƒ \_\_proto\_\_()

descriptor

{value: "abc", writable: true, enumerable: true, configurable: false}configurable: falseenumerable: truevalue: "abc"writable: true\_\_proto\_\_: constructor: ƒ Object()hasOwnProperty: ƒ hasOwnProperty()isPrototypeOf: ƒ isPrototypeOf()propertyIsEnumerable: ƒ propertyIsEnumerable()toLocaleString: ƒ toLocaleString()toString: ƒ toString()valueOf: ƒ valueOf()\_\_defineGetter\_\_: ƒ \_\_defineGetter\_\_()\_\_defineSetter\_\_: ƒ \_\_defineSetter\_\_()\_\_lookupGetter\_\_: ƒ \_\_lookupGetter\_\_()\_\_lookupSetter\_\_: ƒ \_\_lookupSetter\_\_()get \_\_proto\_\_: ƒ \_\_proto\_\_()set \_\_proto\_\_: ƒ \_\_proto\_\_()

var obj = { a: "1"};

Object.preventExtensions(obj);

document.write(Object.isExtensible(obj));//false

obj.newProp = 50;

document.write(obj.newProp);//undefined

undefined

obj

{a: "1"}a: "1"\_\_proto\_\_: constructor: ƒ Object()hasOwnProperty: ƒ hasOwnProperty()isPrototypeOf: ƒ isPrototypeOf()propertyIsEnumerable: ƒ propertyIsEnumerable()toLocaleString: ƒ toLocaleString()toString: ƒ toString()valueOf: ƒ valueOf()\_\_defineGetter\_\_: ƒ \_\_defineGetter\_\_()\_\_defineSetter\_\_: ƒ \_\_defineSetter\_\_()\_\_lookupGetter\_\_: ƒ \_\_lookupGetter\_\_()\_\_lookupSetter\_\_: ƒ \_\_lookupSetter\_\_()get \_\_proto\_\_: ƒ \_\_proto\_\_()set \_\_proto\_\_: ƒ \_\_proto\_\_()

**es6 javascript的\_\_proto\_\_ 属性， Object.setPrototypeOf() ， Object.getPrototypeOf()**

2016年12月01日 17:27:44 阅读数：4425 标签： [es6](http://so.csdn.net/so/search/s.do?q=es6&t=blog)[javascript](http://so.csdn.net/so/search/s.do?q=javascript&t=blog)[\_\_proto\_\_](http://so.csdn.net/so/search/s.do?q=__proto__&t=blog) 更多

个人分类： [es6笔记](https://blog.csdn.net/qq_30100043/article/category/6522874)

（ 1 ）\_\_proto\_\_ 属性

\_\_proto\_\_属性（前后各两个下划线），用来读取或设置当前对象的prototype对象。目前，所有浏览器（包括 IE11 ）都部署了这个属性。

1. *// es6 的写法*
2. var obj = {
3. method: function() { ... }
4. };
5. obj.\_\_proto\_\_ = someOtherObj;
6. *// es5 的写法*
7. var obj = Object.create(someOtherObj);
8. obj.method = function() { ... };

该属性没有写入 ES6 的正文，而是写入了附录，原因是\_\_proto\_\_前后的双下划线，说明它本质上是一个内部属性，而不是一个正式的对外的 API ，只是由于浏览器广泛支持，才被加入了 ES6 。标准明确规定，只有浏览器必须部署这个属性，其他运行环境不一定需要部署，而且新的代码最好认为这个属性是不存在的。因此，无论从语义的角度，还是从兼容性的角度，都不要使用这个属性，而是使用下面的Object.setPrototypeOf()（写操作）、Object.getPrototypeOf()（读操作）、Object.create()（生成操作）代替。

在实现上，\_\_proto\_\_调用的是Object.prototype.\_\_proto\_\_，具体实现如下。

1. Object.defineProperty(Object.prototype, '\_\_proto\_\_', {
2. get() {
3. let \_thisObj = Object(this);
4. return Object.getPrototypeOf(\_thisObj);
5. },
6. set(proto) {
7. if (this === undefined || this === null) {
8. throw new TypeError();
9. }
10. if (!isObject(this)) {
11. return undefined;
12. }
13. if (!isObject(proto)) {
14. return undefined;
15. }
16. let status = Reflect.setPrototypeOf(this, proto);
17. if (!status) {
18. throw new TypeError();
19. }
20. },
21. });
22. function isObject(value) {
23. return Object(value) === value;
24. }

如果一个对象本身部署了\_\_proto\_\_属性，则该属性的值就是对象的原型。

1. Object.getPrototypeOf({ \_\_proto\_\_: null })
2. *// null*

（ 2 ） Object.setPrototypeOf()

Object.setPrototypeOf方法的作用与\_\_proto\_\_相同，用来设置一个对象的prototype对象。它是 ES6 正式推荐的设置原型对象的方法。

1. *// 格式*
2. Object.setPrototypeOf(object, prototype)
3. *// 用法*
4. var o = Object.setPrototypeOf({}, null);
5. *//该方法等同于下面的函数。*
6. function (obj, proto) {
7. obj.\_\_proto\_\_ = proto;
8. return obj;
9. }
10. *//下面是一个例子。*
11. let proto = {};
12. let obj = { x: 10 };
13. Object.setPrototypeOf(obj, proto);
14. proto.y = 20;
15. proto.z = 40;
16. obj.x *// 10*
17. obj.y *// 20*
18. obj.z *// 40*

上面代码将 proto 对象设为 obj 对象的原型，所以从 obj 对象可以读取 proto 对象的属性。

（ 3 ） Object.getPrototypeOf()

该方法与 setPrototypeOf 方法配套，用于读取一个对象的 prototype 对象。

1. Object.getPrototypeOf(obj);
2. *//下面是一个例子。*
3. function Rectangle() {
4. }
5. var rec = new Rectangle();
6. Object.getPrototypeOf(rec) === Rectangle.prototype
7. *// true*
8. Object.setPrototypeOf(rec, Object.prototype);
9. Object.getPrototypeOf(rec) === Rectangle.prototype
10. *// false*

# [JS中isPrototypeOf 和hasOwnProperty 的区别](http://www.cnblogs.com/ass233/p/5880242.html)

**1、isPrototypeOf**   
isPrototypeOf是用来判断指定对象object1是否存在于另一个对象object2的原型链中，是则返回true，否则返回false。   
格式如下：   
object1.isPrototypeOf(object2);   
object1是一个对象的实例；   
object2是另一个将要检查其原型链的对象。   
原型链可以用来在同一个对象类型的不同实例之间共享功能。   
如果 object2 的原型链中包含object1，那么 isPrototypeOf 方法返回 true。   
如果 object2 不是一个对象或者 object1 没有出现在 object2 中的原型链中，isPrototypeOf 方法将返回 false。

**2、hasOwnProperty**hasOwnProperty判断一个对象是否有名称的属性或对象，此方法无法检查该对象的原型链中是否具有该属性，该属性必须是对象本身的一个成员。   
如果该属性或者方法是该 对象自身定义的而不是器原型链中定义的 则返回true;否则返回false;   
格式如下：   
object.hasOwnProperty(proName);   
判断proName的名称是不是object对象的一个属性或对象。

本文转自：http://www.cnblogs.com/ppforever/p/3921481.html

原创文章请随便转载。愿和大家分享，并且一起进步。-- 江 coder

# [js使用in和hasOwnProperty获取对象属性的区别](http://blog.csdn.net/u014291497/article/details/52404205)

in判断的是对象的所有属性，包括对象实例及其原型的属性；   
而hasOwnProperty则是判断对象实例的是否具有某个属性。

示例代码：

[制代码](javascript:void(0);)

<script type="text/javascript">

function Person(){

}

Person.prototype.name = "allen";

var person = new Person();

console.log(person.hasOwnProperty("name")); //false

console.log("name" in person); //true

console.log(person.name); //"allen"

person.name = "justforse";

console.log(person.hasOwnProperty("name")); //true

console.log("name" in person); //true

console.log(person.name); //"justforuse"

delete person.name;

console.log(person.hasOwnProperty("name")); //false

console.log("name" in person); //true

console.log(person.name); //"allen"

</script>

[制代码](javascript:void(0);)

以上代码执行的时候，name属性要么是从实例中获取的，要么是来源于原型，所以使用in 来访问 name属性始终返回true；而hasOwnProperty()只在属性存在与对象实例中时才返回true，当删除了实例中的name属性后，就恢复了原型中name属性的连接，所以返回allen。

此文档的作者：[justforuse](http://www.justforuse.cn/wordpress" \t "_blank)   
Github Pages：[justforuse](http://justforuse.github.io/)

分类: [JAVASCRIPT](https://www.cnblogs.com/jiangxiaobo/category/788871.html)

## call和apply

obj.call(thisObj, arg1, arg2, ...);

obj.apply(thisObj, [arg1, arg2, ...]);

两者作用一致，都是把obj(即this)绑定到thisObj，这时候thisObj具备了obj的属性和方法。或者说thisObj『继承』了obj的属性和方法。绑定后会立即执行函数。

唯一区别是apply接受的是数组参数，call接受的是连续参数。

function add(j, k){

return j+k;

}

function sub(j, k){

return j-k;

}

我们在控制台运行：

add(5,3); //8

add.call(sub, 5, 3); //8

add.apply(sub, [5, 3]); //8

sub(5, 3); //2

sub.call(add, 5, 3); //2

sub.apply(add, [5, 3]); //2

### 调用原生对象的方法

示例：

var a = {0:1, 1:"yjc", length: 2};

a.slice(); //TypeError: a.slice is not a function

Array.prototype.slice.call(a);//[1, "yjc"]

对象a类似array，但不具备array的slice等方法。使用call绑定，这时候就可以调用slice方法。

### 实现继承

通过call和apply，我们可以实现对象继承。示例：

var Parent = function(){

this.name = "yjc";

this.age = 22;

}

var child = {};

console.log(child);//Object {} ,空对象

Parent.call(child);

console.log(child); //Object {name: "yjc", age: 22}

以上实现了对象的继承。

## bind的使用

obj.bind(thisObj, arg1, arg2, ...);

把obj绑定到thisObj，这时候thisObj具备了obj的属性和方法。与call和apply不同的是，bind绑定后不会立即执行。

同样是add()和sub()：

add.bind(sub, 5, 3); //不再返回8

add.bind(sub, 5, 3)(); //8

如果bind的第一个参数是null或者undefined，等于将this绑定到全局对象。

var Parent = function(){ this.name = "yjc"; this.age = 22; } var child = {};

undefined

child

1. *{}*
   1. \_\_proto\_\_:Object

Parent.call(child)

undefined

child

1. *{name: "yjc", age: 22}*
   1. age:22
   2. name:"yjc"
   3. \_\_proto\_\_:Object

在传统的浏览器中，同步的 script 标签是会阻塞 HTML 解析器的，无论是内联的还是外链的

script标签会阻塞html解析器，只有下载和渲染完之后才能继续执行下去

如今的浏览器已经不再这么线性的执行了，在遇到第一个 script 标签后，主线程中的解析器暂停解析，但浏览器会开启一个新的线程去于预解析后面的 HTML 源码，同时预加载遇到的CSS、JS、图片等资源文件，也就是说，在现代浏览器中，上面这个例子中的四个资源文件是会被并行下载的，所以不考虑浏览器的执行耗时的话，渲染出最后那张图片只需要 1 秒钟。

浏览器能做的仅仅是预解析和预加载，脚本的执行和 DOM 树的构建仍然必须是线性的，从而页面的渲染也必须是线性的。脚本必须顺序执行这很好理解，比如 b.js 很可能用到 a.js 里的变量；DOM 树不能提前构建的原因也能想到，a.js 里很可能去查询 DOM 树，在那时执行 querySelectorAll("script").length 必须是 1，img 的话必须是 0。

document.write("<textarea>")

document.write(“<textarea>”)在百度首页不好使，在别的页面没有问题