

严格模式

除了正常的运行模式，JavaScript 还有第二种运行模式：严格模式（strict mode）。顾名思义，这种模式采用更加严格的 JavaScript 语法。

同样的代码，在正常模式和严格模式中，可能会有不一样的运行结果。一些在正常模式下可以运行的语句，在严格模式下将不能运行。

设计目的

早期的 JavaScript 语言有很多设计不合理的地方，但是为了兼容以前的代码，又不能改变老的语法，只能不断添加新的语法，引导程序员使用新语法。

严格模式是从 ES5 进入标准的，主要目的有以下几个。

- 明确禁止一些不合理、不严谨的语法，减少 JavaScript 语言的一些怪异行为。
- 增加更多报错的场合，消除代码运行的一些不安全之处，保证代码运行的安全。
- 提高编译器效率，增加运行速度。
- 为未来新版本的 JavaScript 语法做好铺垫。

总之，严格模式体现了 JavaScript 更合理、更安全、更严谨的发展方向。

启用方法

进入严格模式的标志，是一行字符串`use strict`。

```
```javascript
'use strict';
```
```

老版本的引擎会把它当作一行普通字符串，加以忽略。新版本的引擎就会进入严格模式。

严格模式可以用于整个脚本，也可以只用于单个函数。

（1）整个脚本文件

`use strict`放在脚本文件的第一行，整个脚本都将以严格模式运行。如果这行语句不在第一行就无效，整个脚本会以正常模式运行。（严格地说，只要前面不是产生实际运行结果的语句，`use strict`可以不在第一行，比如直接跟在一个空的分号后面，或者跟在注释后面。）

```
```html
<script>
 'use strict';
 console.log('这是严格模式');
</script>
```
```

```

<script>
  console.log('这是正常模式');
</script>
'''

```

上面代码中，一个网页文件依次有两段 JavaScript 代码。前一个`<script>`标签是严格模式，后一个不是。

如果`use strict`写成下面这样，则不起作用，严格模式必须从代码一开始就生效。

```

'''html
<script>
  console.log('这是正常模式');
  'use strict';
</script>
'''

```

**** (2) 单个函数****

`use strict`放在函数体的第一行，则整个函数以严格模式运行。

```

'''javascript
function strict() {
  'use strict';
  return '这是严格模式';
}

function strict2() {
  'use strict';
  function f() {
    return '这也是严格模式';
  }
  return f();
}

function notStrict() {
  return '这是正常模式';
}
'''

```

有时，需要把不同的脚本合并在一个文件里面。如果一个脚本是严格模式，另一个脚本不是，它们的合并就可能出错。严格模式的脚本在前，则合并后的脚本都是严格模式；如果正常模式的脚本在前，则合并后的脚本都是正常模式。这两种情况下，合并后的结果都是不正确的。这时可以考虑把整个脚本文件放在一个立即执行的匿名函数之中。

```

'''javascript
(function () {
  'use strict';

```

```
// some code here
})();
```

显式报错

严格模式使得 JavaScript 的语法变得更严格，更多的操作会显式报错。其中有些操作，在正常模式下只会默默地失败，不会报错。

只读属性不可写

严格模式下，设置字符串的`length`属性，会报错。

```
``javascript
'use strict';
'abc'.length = 5;
// TypeError: Cannot assign to read only property 'length' of string 'abc'
``
```

上面代码报错，因为`length`是只读属性，严格模式下不可写。正常模式下，改变`length`属性是无效的，但不会报错。

严格模式下，对只读属性赋值，或者删除不可配置（non-configurable）属性都会报错。

```
``javascript
// 对只读属性赋值会报错
'use strict';
Object.defineProperty({}, 'a', {
  value: 37,
  writable: false
});
obj.a = 123;
// TypeError: Cannot assign to read only property 'a' of object #<Object>

// 删除不可配置的属性会报错
'use strict';
var obj = Object.defineProperty({}, 'p', {
  value: 1,
  configurable: false
});
delete obj.p
// TypeError: Cannot delete property 'p' of #<Object>
``
```

只设置了取值器的属性不可写

严格模式下，对一个只有取值器（getter）、没有存值器（setter）的属性赋值，会报错。

```
``javascript
```

```
'use strict';
var obj = {
  get v() { return 1; }
};
obj.v = 2;
// Uncaught TypeError: Cannot set property v of #<Object> which has only a getter
""
```

上面代码中，`obj.v`只有取值器，没有存值器，对它进行赋值就会报错。

禁止扩展的对象不可扩展

严格模式下，对禁止扩展的对象添加新属性，会报错。

```
""javascript
'use strict';
var obj = {};
Object.preventExtensions(obj);
obj.v = 1;
// Uncaught TypeError: Cannot add property v, object is not extensible
""
```

上面代码中，`obj`对象禁止扩展，添加属性就会报错。

eval、arguments 不可用作标识名

严格模式下，使用`eval`或者`arguments`作为标识名，将会报错。下面的语句都会报错。

```
""javascript
'use strict';
var eval = 17;
var arguments = 17;
var obj = { set p(arguments) { } };
try { } catch (arguments) { }
function x(eval) { }
function arguments() { }
var y = function eval() { };
var f = new Function('arguments', "'use strict'; return 17;");
// SyntaxError: Unexpected eval or arguments in strict mode
""
```

函数不能有重名的参数

正常模式下，如果函数有多个重名的参数，可以用`arguments[i]`读取。严格模式下，这属于语法错误。

```
""javascript
function f(a, a, b) {
  'use strict';
  return a + b;
}
```

```
}  
// Uncaught SyntaxError: Duplicate parameter name not allowed in this context  
````
```

### ### 禁止八进制的前缀0表示法

正常模式下，整数的第一位如果是`0`，表示这是八进制数，比如`0100`等于十进制的64。严格模式禁止这种表示法，整数第一位为`0`，将报错。

```
````javascript  
'use strict';  
var n = 0100;  
// Uncaught SyntaxError: Octal literals are not allowed in strict mode.  
````
```

### ## 增强的安全措施

严格模式增强了安全保护，从语法上防止了一些不小心会出现的错误。

### ### 全局变量显式声明

正常模式中，如果一个变量没有声明就赋值，默认是全局变量。严格模式禁止这种用法，全局变量必须显式声明。

```
````javascript  
'use strict';  
  
v = 1; // 报错，v未声明  
  
for (i = 0; i < 2; i++) { // 报错，i 未声明  
    // ...  
}  
  
function f() {  
    x = 123;  
}  
f() // 报错，未声明就创建一个全局变量  
````
```

因此，严格模式下，变量都必须先声明，然后再使用。

### ### 禁止 this 关键字指向全局对象

正常模式下，函数内部的`this`可能会指向全局对象，严格模式禁止这种用法，避免无意间创造全局变量。

```
````javascript  
// 正常模式
```

```
function f() {
  console.log(this === window);
}
f() // true

// 严格模式
function f() {
  'use strict';
  console.log(this === undefined);
}
f() // true
```

```

上面代码中，严格模式的函数体内部`this`是`undefined`。

这种限制对于构造函数尤其有用。使用构造函数时，有时忘了加`new`，这时`this`不再指向全局对象，而是报错。

```
```javascript
function f() {
  'use strict';
  this.a = 1;
};

f();// 报错，this 未定义
```
```

严格模式下，函数直接调用时（不使用`new`调用），函数内部的`this`表示`undefined`（未定义），因此可以用`call`、`apply`和`bind`方法，将任意值绑定在`this`上面。正常模式下，`this`指向全局对象，如果绑定的值是非对象，将被自动转为对象再绑定上去，而`null`和`undefined`这两个无法转成对象的值，将被忽略。

```
```javascript
// 正常模式
function fun() {
  return this;
}

fun() // window
fun.call(2) // Number {2}
fun.call(true) // Boolean {true}
fun.call(null) // window
fun.call(undefined) // window

// 严格模式
'use strict';
function fun() {
  return this;
}
```
```

```
fun() //undefined
fun.call(2) // 2
fun.call(true) // true
fun.call(null) // null
fun.call(undefined) // undefined
```
```

上面代码中，可以把任意类型的值，绑定在`this`上面。

禁止使用 `fn.callee`、`fn.caller`

函数内部不得使用`fn.caller`、`fn.arguments`，否则会报错。这意味着不能在函数内部得到调用栈了。

```
```javascript
function f1() {
 'use strict';
 f1.caller; // 报错
 f1.arguments; // 报错
}

f1();
```
```

禁止使用 `arguments.callee`、`arguments.caller`

`arguments.callee`和`arguments.caller`是两个历史遗留的变量，从来没有标准化过，现在已经取消了。正常模式下调用它们没有什么作用，但是不会报错。严格模式明确规定，函数内部使用`arguments.callee`、`arguments.caller`将会报错。

```
```javascript
'use strict';
var f = function () {
 return arguments.callee;
};

f(); // 报错
```
```

禁止删除变量

严格模式下无法删除变量，如果使用`delete`命令删除一个变量，会报错。只有对象的属性，且属性的描述对象的`configurable`属性设置为`true`，才能被`delete`命令删除。

```
```javascript
'use strict';
var x;
delete x; // 语法错误
```
```

```
var obj = Object.create(null, {
  x: {
    value: 1,
    configurable: true
  }
});
delete obj.x; // 删除成功
``
```

静态绑定

JavaScript 语言的一个特点，就是允许“动态绑定”，即某些属性和方法到底属于哪一个对象，不是在编译时确定的，而是在运行时（runtime）确定的。

严格模式对动态绑定做了一些限制。某些情况下，只允许静态绑定。也就是说，属性和方法到底归属哪个对象，必须在编译阶段就确定。这样做有利于编译效率的提高，也使得代码更容易阅读，更少出现意外。

具体来说，涉及以下几个方面。

禁止使用 with 语句

严格模式下，使用`with`语句将报错。因为`with`语句无法在编译时就确定，某个属性到底归属哪个对象，从而影响了编译效果。

```
``javascript
'use strict';
var v = 1;
var obj = {};

with (obj) {
  v = 2;
}
// Uncaught SyntaxError: Strict mode code may not include a with statement
``
```

创设 eval 作用域

正常模式下，JavaScript 语言有两种变量作用域（scope）：全局作用域和函数作用域。严格模式创设了第三种作用域：`eval`作用域。

正常模式下，`eval`语句的作用域，取决于它处于全局作用域，还是函数作用域。严格模式下，`eval`语句本身就是一个作用域，不再能够在其所运行的作用域创设新的变量了，也就是说，`eval`所生成的变量只能用于`eval`内部。

```
``javascript
(function () {
```



```

'use strict';
var x = 2;
console.log(eval('var x = 5; x')) // 5
console.log(x) // 2
})0

```

上面代码中，由于`eval`语句内部是一个独立作用域，所以内部的变量`x`不会泄露到外部。

注意，如果希望`eval`语句也使用严格模式，有两种方式。

```

```javascript
// 方式一
function f1(str){
 'use strict';
 return eval(str);
}
f1('undeclared_variable = 1'); // 报错

// 方式二
function f2(str){
 return eval(str);
}
f2("'use strict';undeclared_variable = 1') // 报错
```

```

上面两种写法，`eval`内部使用的都是严格模式。

arguments 不再追踪参数的变化

变量`arguments`代表函数的参数。严格模式下，函数内部改变参数与`arguments`的联系被切断了，两者不再存在联动关系。

```

```javascript
function f(a) {
 a = 2;
 return [a, arguments[0]];
}
f(1); // 正常模式为[2, 2]

function f(a) {
 'use strict';
 a = 2;
 return [a, arguments[0]];
}
f(1); // 严格模式为[2, 1]
```

```

上面代码中，改变函数的参数，不会反应到`arguments`对象上来。

向下一个版本的 JavaScript 过渡

JavaScript 语言的下一个版本是 ECMAScript 6，为了平稳过渡，严格模式引入了一些 ES6 语法。

非函数代码块不得声明函数

ES6 会引入块级作用域。为了与新版本接轨，ES5 的严格模式只允许在全局作用域或函数作用域声明函数。也就是说，不允许在非函数的代码块内声明函数。

```
``javascript
'use strict';
if (true) {
  function f1() {} // 语法错误
}

for (var i = 0; i < 5; i++) {
  function f2() {} // 语法错误
}
``
```

上面代码在`if`代码块和`for`代码块中声明了函数，ES5 环境会报错。

注意，如果是 ES6 环境，上面的代码不会报错，因为 ES6 允许在代码块之中声明函数。

保留字

为了向将来 JavaScript 的新版本过渡，严格模式新增了一些保留字（implements、interface、let、package、private、protected、public、static、yield等）。使用这些词作为变量名将会报错。

```
``javascript
function package(protected) { // 语法错误
  'use strict';
  var implements; // 语法错误
}
``
```

参考链接

- MDN, [Strict mode](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Functions_and_function_scope/Strict_mode)
- Dr. Axel Rauschmayer, [JavaScript: Why the hatred for strict mode?](http://www.2ality.com/2011/10/strict-mode-hatred.html)
- Dr. Axel Rauschmayer, [JavaScript's strict mode: a summary](http://www.2ality.com/2011/01/javascripts-strict-mode-summary.html)

- Douglas Crockford, [Strict Mode Is Coming To Town](<http://www.yuiblog.com/blog/2010/12/14/strict-mode-is-coming-to-town/>)
- [JavaScript Strict Mode Support](http://java-script.limewebs.com/strictMode/test_hosted.html)