严格模式

严格模式是ES5引入

严格模式主要有以下限制:

1. 变量必须声明后再使用
2. 函数的参数不能有同名属性，否则报错
3. 不能使用with语句
4. 不能对只读属性赋值，否则报错
5. 不能使用前缀0表示八进制数，否则报错
6. 不能删除不可删除的属性，否则报错
7. 不能删除变量delete prop，会报错，只能删除属性delete global[prop]
8. eval不会在它的外层作用域引入变量
9. eval和arguments不能被重新赋值
10. arguments不会自动反映函数参数的变化
11. 不能使用arguments.callee
12. 不能使用arguments.caller
13. 禁止this指向全局对象
14. 不能使用fn.caller和fn.arguments获取函数调用的堆栈
15. 增加了保留字（比如protected、static和interface）

"use strict" 严格模式

> "use strict" 是ES5 中的新指令（ECMAScript version 5）。
>
> "严格模式"体现了Javascript更合理、更安全、更严谨的发展方向，包括IE 10在内的主流浏览器，都已经支持它，许多大项目已经开始全面拥抱它。

开启严格模式

针对脚本

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

针对函数

```
function strict(){
  "use strict";
  return "这是严格模式。";
}
```

优化形式

```
//因为第一种调用方法不利于文件合并，所以更好的做法是，借用第二种方法，将整个脚本文件放在一个立即执行的匿名函数之中。
(function (){

"use strict";

// some code here

})();
```

全局变量显式声明

在正常模式中，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。

```
"use strict";
v = 1; // 报错，v未声明

for(i = 0; i < 2; i++) { // 报错，i未声明
}
```

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，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。

（1）禁止使用with语句

> 因为with语句无法在编译时就确定，属性到底归属哪个对象。

```
"use strict";

var v = 1;

with (o){ // 语法错误
  v = 2;
}
```

（2）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，还是处于函数作用域。严格模式下，eval语句本身就是一个作用域，不再能够生成全局变量了，它所生成的变量只能用于eval内部。

```
"use strict";

var x = 2;

console.info(eval("var x = 5; x")); // 5

console.info(x); // 2
```

增强的安全措施

（1）禁止this关键字指向全局对象

```
function f(){
        return !this;
  }
  // 返回false，因为"this"指向全局对象，"!this"就是false

  function f(){
        "use strict";
        return !this;
  }
  // 返回true，因为严格模式下，this的值为undefined，所以"!this"为true。
```

因此，使用构造函数时，如果忘了加new，this不再指向全局对象，而是报错。

```
function f(){
  return !this;
}
// 返回false，因为"this"指向全局对象，"!this"就是false

function f(){
  "use strict";
  return !this;
}
// 返回true，因为严格模式下，this的值为undefined，所以"!this"为true。
```

（2）禁止在函数内部遍历调用栈

```
function f1(){

  "use strict";

  f1.caller; // 报错

  f1.arguments; // 报错

}

f1();
```

禁止删除变量

> 严格模式下无法删除变量。只有configurable设置为true的对象属性，才能被删除。

```
"use strict";

var x;

delete x; // 语法错误

var o = Object.create(null, {'x': {
  value: 1,
  configurable: true
}});

delete o.x; // 删除成功
```

显式报错

> 正常模式下，对一个对象的只读属性进行赋值，不会报错，只会默默地失败。严格模式下，将报错。

```
"use strict";

var o = {};

Object.defineProperty(o, "v", { value: 1, writable: false });

o.v = 2; // 报错
```

严格模式下，对一个使用getter方法读取的属性进行赋值，会报错。

```
"use strict";

var o = {

    get v() { return 1; }

};

o.v = 2; // 报错
```

严格模式下，对禁止扩展的对象添加新属性，会报错。

```
"use strict";

var o = {};

Object.preventExtensions(o);

o.v = 1; // 报错
```

严格模式下，删除一个不可删除的属性，会报错。

```
"use strict";

delete Object.prototype; // 报错
```

重名错误

严格模式新增了一些语法错误。

（1）对象不能有重名的属性

正常模式下，如果对象有多个重名属性，最后赋值的那个属性会覆盖前面的值。严格模式下，这属于语法错误。

```
"use strict";

  var o = {
      p: 1,
      p: 2
  }; // 语法错误
```

（2）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。

```
"use strict";

  function f(a, a, b) { // 语法错误

      return ;

  }
```

禁止八进制表示法

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。

```
  "use strict";

  var n = 0100; // 语法错误
```

arguments对象的限制

arguments是函数的参数对象，严格模式对它的使用做了限制。

（1）不允许对arguments赋值

```
"use strict";

arguments++; // 语法错误

var obj = { set p(arguments) { } }; // 语法错误

try { } catch (arguments) { } // 语法错误

function arguments() { } // 语法错误

var f = new Function("arguments", "'use strict'; return 17;"); // 语法错误
```

（2）arguments不再追踪参数的变化

```
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]
```

（3）禁止使用arguments.callee

这意味着，你无法在匿名函数内部调用自身了。

```
"use strict";

var f = function() { return arguments.callee; };

f(); // 报错
```

函数必须声明在顶层

将来Javascript的新版本会引入"块级作用域"。为了与新版本接轨，严格模式只允许在全局作用域或函数作用域的顶层声明函数。也就是说，不允许在非函数的代码块内声明函数。

```
"use strict";

  if (true) {

      function f() { } // 语法错误

  }

  for (var i = 0; i < 5; i++) {

      function f2() { } // 语法错误

  }
```

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。

使用这些词作为变量名将会报错。

```
  function package(protected) { // 语法错误

      "use strict";

      var implements; // 语法错误

  }
```

此外，ECMAscript第五版本身还规定了另一些保留字（class, enum, export, extends, import, super），以及各大浏览器自行增加的const保留字，也是不能作为变量名的。