JavaScript深入浅出

语句、严格模式

@Bosn

语句 Statement

JavaScript程序由语句组成，语句遵守特定的语法规则。

例如：if语句, while语句, with语句等等。

语句

block
break
continue
empty
if...else
switch
try catch

var
function
return
do...while
for
for...in
while

debugger
label
with

块 block

块语句常用于组合0 ~ 多个语句。块语句用一对花括号定义。

语法：

```
{
    语句1;
    语句2;
    ...
    语句n;
}
```

```
{
    var str = "hi";
    console.log(str);
}
```

```
if (true) {
    console.log('hi');
}
```

```
{a : 1, b : 2} // SyntaxError: Unexpected token :
var o = {a : 1, b : 2}; //ok
```

请注意：没有块级作用域

块 block

请注意：没有块级作用域

```
for (var i = 0; i < 10; i++) {
    var str = "hi";
    console.log(str);
}
```

=

```
var i = 0
for (; i < 10; i++) {
    var str = "hi";
    console.log(str);
}
```

```
{
    var x = 1;
    // .....
}
```

=

```
var x = 1;
{
    // .....
}
```

块 block

请注意：没有块级作用域

```
function foo() {
    var a = 1;
    console.log(a); // 1
}
foo();
console.log(typeof a); // undefined
```

var

```
var a = 1;
```

```
var a = b = 1;
```

```
var a = 1, b = 1;
```

```
function foo() {
    var a = b = 1;
}
foo();

console.log(typeof a);  //    'undefined'
console.log(typeof b);  //    'number'
```

try catch

```
try {
    throw "test";
} catch (ex) {
    console.log(ex); // test
} finally {
    console.log('finally');
}
```

try catch

```
try {
    // do sth.
} finally {
    console.log('finally');
}
```

try catch

```
try {
    try {
        throw new Error("oops");
    }
    finally {
        console.log("finally");
    }
}
catch (ex) {
    console.error("outer", ex.message);
}
```

```
"finally"
"outer" "oops"
```

try catch

```
try {
  try {
    throw new Error("oops");
  }
  catch (ex) {
    console.error("inner", ex.message);
  }
  finally {
    console.log("finally");
  }
}
catch (ex) {
  console.error("outer", ex.message);
}
```

```
"inner" "oops"
"finally"
```

try catch

```
try {
  try {
    throw new Error("oops");
  }
  catch (ex) {
    console.error("inner", ex.message);
    throw ex;
  }
  finally {
    console.log("finally");
  }
}
catch (ex) {
  console.error("outer", ex.message);
}
```

```
"inner" "oops"
"finally"
"outer" "oops"
```

function

```
fd(); // true

function fd() {
    // do sth.
    return true;
}
```

```
fe(); // TypeError

var fe = function() {
    // do sth.
};
```

for...in

```
var p;
var obj = {x : 1, y: 2}

for (p in obj) {
}
```

1. 顺序不确定
2. enumerable为false时不会出现
3. for in对象属性时受原型链影响

switch

```
var val = 2;
```

```
switch(val) {
    case 1:
        console.log(1);
        break;
    case 2:
        console.log(2);
        break;
    default:
        console.log(0);
        break;
}
```

2

```
switch(val) {
    case 1:
        console.log(1);
    case 2:
        console.log(2);
    default:
        console.log(0);
}
```

2,0

```
switch(val) {
    case 1:
    case 2:
    case 3:
        console.log(123);
        break;
    case 4:
    case 5:
        console.log(45);
        break;
    default:
        console.log(0);
}
```

123

循环

```
while(isTrue) {
    // do sth.
}
```

```
do {
    // do sth.
} while (isTrue)
```

```
var i;
for (i = 0; i < n; i++) {
    // do sth.
}
```

with

```
with ({x : 1}) {
    console.log(x);
}
```

```
with (document.forms[0]) {
    console.log(name.value);
}
```

```
var form = document.forms[0];
console.log(form.name.value);
```

- 让JS引擎优化更难
- 可读性差
- 可被变量定义代替
- 严格模式下被禁用

严格模式

严格模式是一种特殊的执行模式，
它修复了部分语言上的不足，
提供更强的错误检查，并增强安全性。

```
'use strict';
function func() {

}
```

```
function func() {
    'use strict';
}
```

```
func
```

不允许用with

```
!function() {
    with({x : 1}) {
        console.log(x);
    }
}();
```

1

```
!function() {
    'use strict';
    with({x : 1}) {
        console.log(x);
    }
}();
```

SyntaxError

不允许未声明的变量被赋值

```
!function() {
    x = 1;
    console.log(window.x);
}();
```

1

```
!function() {
    'use strict';
    x = 1;
    console.log(window.x);
}();
```

ReferenceError

arguments变为参数的静态副本

```
!function(a) {
    arguments[0] = 100;
    console.log(a);
}(1);
```

100

1=>100
不传 => undefined

```
!function(a) {
    'use strict';
    arguments[0] = 100;
    console.log(a);
}(1);
```

1

```
!function(a) {
    'use strict';
    arguments[0].x = 100;
    console.log(a.x);
}({x:1});
```

100

delete参数、函数名报错

```
!function(a) {
    console.log(delete a);
}(1);
```

false

```
!function(a) {
    'use strict';
    delete a;
}(1);
```

SyntaxError

delete不可配置的属性报错

```
!function(a) {
    var obj = {};
    Object.defineProperty(obj,
      'a', {configurable : false});
    console.log(delete obj.a);
}(1);
```

false

```
!function(a) {
    'use strict';
    var obj = {};
    Object.defineProperty(obj,
      'a', {configurable : false});
    delete obj.a;
}(1);
```

TypeError

对象字面量重复属性名报错

```
!function() {
    var obj = {x : 1, x : 2};
    console.log(obj.x);
}();
```

2

```
!function() {
    'use strict';
    var obj = {x : 1, x : 2};
}();
```

SyntaxError

禁止八进制字面量

```
!function() {
    console.log(0123);
}();
```

83

```
!function() {
    'use strict';
    console.log(0123);
}();
```

SyntaxError

严格模式

eval, arguments变为关键字，不能作为变量、函数名

```
!function() {
    function eval(){}
    console.log(eval);
}();
```

function eval(){}

```
!function() {
    'use strict';
    function eval(){}
}();
```

SyntaxError

eval独立作用域

```
!function() {
    eval('var evalVal = 2;');
    console.log(typeof evalVal);
}();
```

number

```
!function() {
    'use strict';
    eval('var evalVal = 2;');
    console.log(typeof evalVal);
}();
```

undefined

严格模式

不允许用with

所有变量必须声明, 赋值给为声明的变量报错，而不是隐式创建全局变量。

eval中的代码不能创建eval所在作用域下的变量、函数。而是为eval单独创建一个作用域，并在eval返回时丢弃。

函数中得特殊对象arguments是静态副本，而不像非严格模式那样，修改arguments或修改参数变量会相互影响。

删除configurable=false的属性时报错，而不是忽略

禁止八进制字面量，如010 (八进制的8)

eval, arguments变为关键字，不可作为变量名、函数名等

一般函数调用时(不是对象的方法调用，也不使用apply/call/bind等修改this)this指向null，而不是全局对象。

若使用apply/call，当传入null或undefined时，this将指向null或undefined，而不是全局对象。

试图修改不可写属性(writable=false)，在不可扩展的对象上添加属性时报TypeError，而不是忽略。

arguments.caller, arguments.callee被禁用

严格模式

严格模式是一种特殊的运行模式，
它修复了部分语言上的不足，
提供更强的错误检查，并增强安全性。

谢谢