

# JS 基础

黄小桐

## 由来

由 Brendan Eich 于 1995 年创造

# ECMAScript 为其语言标准

ES5 于 2009 年发布

ES6 (es2015) 于 2015 年发布

# JavaScript 引擎

解释和执行 JS 代码的虚拟机

V8 by Google used in Chrome、Node.js

Chakra by Microsoft in Edge

SpiderMonkey by Mozilla used in Firefox

JavaScriptCore by Apple used in Safari

# 运行 JavaScript

1. 通过 script 标签链接一个 js 文文件 (通常情况下在页面底部)
2. 直接在 script 标签里写 js 代码



```
<script type="text/javascript" src="main.js">
</script>

<script type="text/javascript">
  var a = 1;
  alert( a );
</script>
```

# 数据类型

Null

Undefined

Bool

Number

String

Symbol

BigInt

Array

Function

Date

RegExp

Error

# 数据类型

## Null

- 值: null

## Undefined

- 值: undefined
- 未赋值的变量是 undefined

## Boolean

- 值: true 和 false

## Number

- 不区分整数值和浮点数值, 所有的数均为浮点数表示
- $0.1 + 0.2 == 0.3$  ?

# 数据类型

## Boolean 转换规则

0 " null undefined NaN false 为 false

其他都是 true

# 数据类型

## Number 内置对象 Math

`Math.sin(3.5)`

`const d = Math.PI * 3`

**NaN: not a number**

- `parseInt('a') => NaN`
- `NaN !== NaN`



# 数据类型

`parseInt(string, radix)` 将字符串转化为整形

逐个解析字符串中的字符，直到遇到一个无法被解析为数字的字符

Radix 默认是 10 进制，调用这个函数时，最好写上具体的 radix 值

不传 radix 时：

Radix = 16 when the string starts with 0x or 0X

Radix = 8 or 10 when the string starts with 0 Else radix is 10

# 数据类型



```
parseInt(" 0xF", 16);  
parseInt(" F", 16);  
parseInt("015", 10);  
parseInt(15.99, 10);  
parseInt("FXX123", 16);  
parseInt("15e2", 10);  
parseInt("Hello", 8);
```

# 数据类型

## 获取数据类型

`typeof`

`Object.prototype.toString.call`

有什么区别？

# 语法



```
/* this is comment */
```

```
let a = true
```

```
if ( a ) {
```

```
    a = false
```

```
}
```

```
while ( !a ) {
```

```
    a = true
```

```
}
```

## 标识符 (变量名)

区分大小写

规则

Start with \$ \_ letter

Followed by digits(0-9) \$ \_ letter

Unicode letters

Unicode escape sequences

## 标识符 (变量名)

### 命名规则

驼峰风格

常量名全大写

类名大写开始, 其他的都是小写开始



```
usedMoney  
setState  
isDirty  
function Editor(){  
    this.id = utils.getId()  
    this.document = document  
}
```

代码是给人(别人或者1个星期后的 自己)阅读和修改的

給变量和函数一个简单易懂的名字

可维护性++



**const** 常量, **let** 变量

**var** 两种场景均可使用 (不推荐)



```
let a = true;  
let b;  
let c,d,e,  
    f = 1024;
```

## 尽量避免全局变量

不易维护

牵一发而动全身

易被覆盖,修改,而你还不知道

循环控制

while

do-while

for

for-in

分支控制

if

switch

## 循环控制 while & do-while

```
while ( expression ) {  
    }  
do {  
    statement;  
} while ( expression )
```

## 循环控制 for & for-in

```
for( initialize; test; update ) {  
    statement;  
}  
  
for (property in object) {  
    statement;  
}
```

if branch

```
if( expression ) {  
    statement;  
} else if (expression) {  
    statement;  
}
```

switch branch

```
switch ( expression ) {  
    case expression:  
        statement;  
    case expression:  
        statements  
        break  
    default  
        statements
```

```
}
```



```
// 函数定义
function add(x, y) {
    return x + y
}

// 函数表达式
var add = function(x, y){
    return x + y + 1
}
```

函数声明和函数表达式




一个 JavaScript 函数可以包含 0 个或多个已命名的变量

`return` 语句返回一个值并结束函数

如果没有使用 `return` 语句, `return` 语句没返回值, 函数会返回 `undefined`

如果调用函数时没有提供足够的参数,缺少的参数会被 `undefined` 替代

可以传入多于函数本身需要参数个数的参数,可以通过 `arguments` 访问



```
function add() {  
    var sum = 0;  
    for (var i = 0, j = arguments.length; i < j; i++) {  
        sum += arguments[i];  
    }  
    return sum;  
}  
add()  
add(2, 3, 4)
```

# 变量作用域

var 声明的变量会成为所在函数或者全局作用域的变量

ES6 引入了语句块作用域




```
for (let i = 0; i < 10; i++) {  
  console.log(i)  
}  
console.log(i)  
  
for (var j = 0; j < 10; j++) {  
  console.log(j)  
}  
console.log(j)
```

结果是什么？

# 变量提升

var 声明的变量提升，被赋值为 undefined



```
function test() {  
  console.info(a === undefined)  
  var a = 1  
}
```

# 函数提升

函数声明的提升

函数表达式呢？

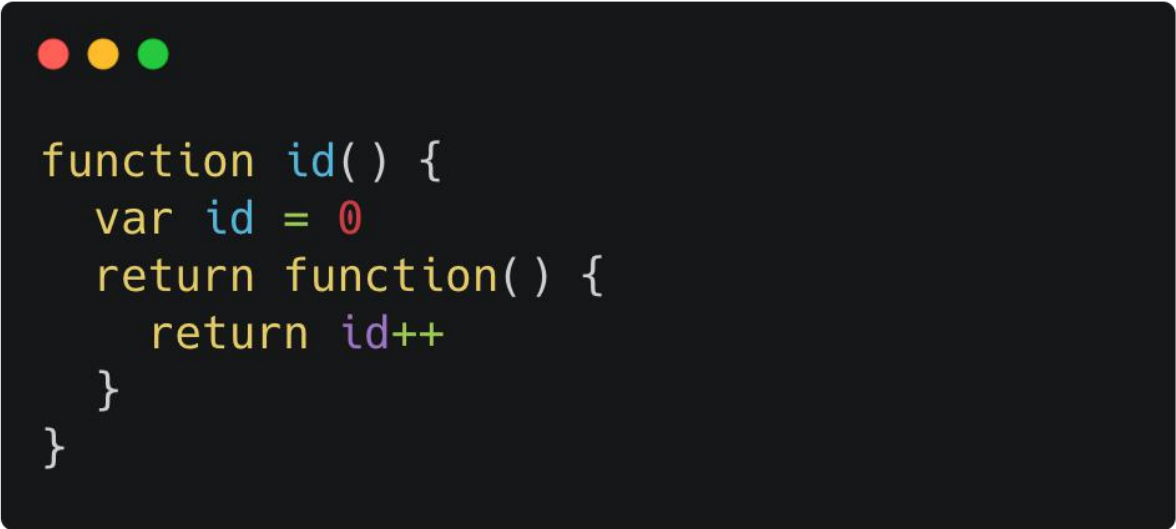


```
test( )
```

```
function test() {  
  console.info(a === undefined)  
  var a = 1  
}
```

# 闭包

通俗的理解：函数内部可以访问到函数外部的变量的机制



```
function id() {  
  var id = 0  
  return function() {  
    return id++  
  }  
}
```

## 自动分号插入

当不清楚写不写分号时，写上

```
var tester = function() {  
  
}  
  
(function() {  
    console.log(tester)  
})();  
  
var a = 123  
[123].conct(123)
```



# 额外要了解的

ES6 class

Promise

Async await

CommonJS / AMD / UMD / ES modules

functional javascript

TypeScript

# 参考

**javascript 兼容性**

<https://kangax.github.io/compat-table/es6/>

**自动分号插入**

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical\\_grammar#automatic\\_semicolon\\_insertion](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar#automatic_semicolon_insertion)

谢谢