链滴

# 面试开发常用的 JavaScript 知识点总结

作者：Vanessa
原文链接：https://ld246.com/article/1470722230732
来源网站：链滴

# No1.语法和类型

## 1.声明定义

变量类型：var，定义变量；let，定义块域(scope)本地变量；const，定义只读常量。

变量格式：以字母、下划线“_”或者$符号开头，大小写敏感。

变量赋值：声明但未赋值的变量在使用时值为undefined，未声明的变量直接使用会抛异常。

未赋值变量作计算：结果为NaN。例如：

```
var x, y = 1;
console.log(x + y); //结果为NaN，因为x没有赋值。
```

## 2.作用域

变量作用域：在ES6之前没有块声明域，变量作用于函数块或者全局。如下面的代码输入的x为5。

```
if (true) {
    var x = 5;
}
console.log(x); // 5
```

ES6变量作用域：ES6支持块作用域，但需要使用let声明变量。下面的代码输出结果抛出异常。

```
if (true) {
    let y = 5;
}
console.log(y); // ReferenceError: y is not defined1234
```

变量上浮：在一个方法或者全局代码中，我们在生命变量之前使用变量时并没有抛异常，而是返undefined。这是因为javascript自动把变量的声明上浮到函数或者全局的最前面。如下面的代码:

```
/**
* 全局变量上浮
*/
console.log(x === undefined); // logs "true"
var x = 3;

/**
* 方法变量上浮
*/
var myvar = "my value";
// 打印变量myvar结果为：undefined
(function() {
    console.log(myvar); // undefined
    var myvar = "local value";
})();
```

上面代码和下面代码是等价的：

```
/**
* 全局变量上浮
*/
var x;
console.log(x === undefined); // logs "true"
x = 3;

/**
* 方法变量上浮
*/
var myvar = "my value";
(function() {
    var myvar;
    console.log(myvar); // undefined
    myvar = "local value";
})();
```

全局变量：在页面中，全局对象是window，所以我们访问全局变量可通过window.variable。如：

```
version = "1.0.0";
console.log(window.version); //输出1.0.0
```

# No2.数据结构和类型

## 1.数据类型

6个基础类型：Boolean（true或者false）、null（js大小写敏感，和Null、NULL是有区别的）、ndefined、Number、String、Symbol（标记唯一和不可变）

一个对象类型：object。

object和function：对象作为值的容器，函数作为应用程序的过程。

## 2.数据转换

函数：字符串转换为数字可使用parseInt和parseFloat方法。

parseInt：函数签名为parseInt(string, radix)，radix是2-36的数字表示数字基数，例如十进制者十六进制。返回结果为integer或者NaN，例如下面输出结果都为15。

```
parseInt("0xF", 16);
parseInt("F", 16);
parseInt("17", 8);
parseInt(021, 8);
parseInt("015", 10);
parseInt(15.99, 10);
arseInt("15,123", 10);
```

```
parseInt("FXX123", 16);
parseInt("1111", 2);
parseInt("15*3", 10);
parseInt("15e2", 10);
parseInt("15px", 10);
```

parseFloat：函数签名为parseFloat(string)，返回结果为数字或者NaN。例如：

```
parseFloat("3.14"); //返回数字
parseFloat("314e-2"); //返回数字
parseFloat("more non-digit characters"); //返回NaN
```

# 3.数据类型文本化

文本化类型：Array、Boolean、Floating-point 、integers、Object、RegExp、String。

Array中额外的逗号情况：["Lion", , "Angel"]，长度为3，[1]的值为undefiend。['home', , 'school', ]，最后一个逗号省略所以长度为3。[ , 'home', , 'school']，长度为4。['home', , 'school', , ]，长为4。

integer整数：整数可以表达为十进制、八进制、十六进制、二进制。例如：

```
0, 117 and -345 //十进制
015, 0001 and -0o77 //八进制
0x1123, 0x00111 and -0xF1A7 //十六进制
0b11, 0b0011 and -0b11 1234 //二进制
```

浮点数：[(+|-)][digits][.digits][(E|e)[(+|-)]digits]。例如：

```
3.1415926，-.123456789，-3.1E+12（3100000000000），.1e-23（1e-24）
```

对象：对象的属性获取值可通过“.属性”或者“[属性名]”获取。例如：

```
var car = { manyCars: {a: "Saab", "b": "Jeep"}, 7: "Mazda" };
console.log(car.manyCars.b); // Jeep
console.log(car[7]); // Mazda
```

对象属性：属性名可以是任意字符串或者空字符串，无效的名字可通过引号包含起来。复杂的名不能通过.获取，但可以通过[]获取。例如：

```
var unusualPropertyNames = {
    "": "An empty string",
    "!": "Bang!"
}
console.log(unusualPropertyNames.""); // SyntaxError: Unexpected string
console.log(unusualPropertyNames[""]); // An empty string
console.log(unusualPropertyNames.!); // SyntaxError: Unexpected token !
console.log(unusualPropertyNames["!"]); // Bang!
```

转意字符：下面的字符串输出结果包含了双引号，因为使用了转意符号“"

。

```
var quote = "He read \"The Cremation of Sam McGee\" by R.W. Service.";
console.log(quote);
//输出：He read "The Cremation of Sam McGee" by R.W. Service.1。
```

字符串换行法：直接在字符串行结束时添加“\”，如下代码所示：

```
var str = "this string \
is broken \
across multiple\
lines."
console.log(str); // this string is broken across multiplelines.
```

# No3.控制流和错误处理

## 1.块表达式

作用：块表达式一般用于控制流，像if、for、while。下面的代码中{x++;}就是一个块声明。

```
while (x < 10) {
   x++;
}
```

ES6之前没有块域范围：在ES6之前，在block中定义的变量实际是包含在方法或者全局中，变量影响超出了块作用域的范围。例如下面的代码最终执行结果为2，因为block中声明的变量作用于方法。

```
var x = 1;
{
   var x = 2;
}
console.log(x); // outputs 2
```

ES6之后有块域范围：在ES6中，我们可以把块域声明var改成let，让变量只作用域block范围。

## 2.逻辑判断

判断为false的特殊值：false、undefined、null、0、NaN、""。

简单boolean和对象Boolean类型：简单boolean类型的false和true与对象Boolean类型的false和rue是有区别，两者是不相等的。如下面的例子：

```
var b = new Boolean(false);
if (b) // 返回true
if (b == true) // 返回false
```

# No4.异常处理

# 1.异常类型

抛出异常语法：抛异常可以是任意类型。如下所示。

```
throw "Error2"; // 字符串类型
throw 42; // 数字类型
throw true; // 布尔类型
throw {toString: function() { return "I'm an object!"; } }; //对象类型
```

自定义异常：

```
// 创建一个对象类型UserException
function UserException(message) {
   this.message = message;
   this.name = "UserException";
}

//重写toString方法，在抛出异常时能直接获取有用信息
UserException.prototype.toString = function() {
   return this.name + ': "' + this.message + '"';
}

// 创建一个对象实体并抛出它
throw new UserException("Value too high");
```

# 2.语法

关键字：使用try{}catch(e){}finally{}语法，和C#语法相似。

finally返回值：如果finaly添加了return 语句，则不管整个try.catch返回什么，返回值都是finall的return。如下所示：

```
function f() {
   try {
      console.log(0);
      throw "bogus";
   } catch(e) {
      console.log(1);
      return true; // 返回语句被暂停，直到finally执行完成
      console.log(2); // 不会执行的代码
   } finally {
      console.log(3);
      return false; //覆盖try.catch的返回
      console.log(4); //不会执行的代码
   }
   // "return false" is executed now
   console.log(5); // not reachable
}
f(); // 输出 0, 1, 3; 返回 false
```

finally吞并异常：如果finally有return并且catch中有throw异常。throw的异常不会被捕获，因

已经被finally的return覆盖了。如下代码所示：

```
function f() {
    try {
        throw "bogus";
    } catch(e) {
        console.log('caught inner "bogus"');
        throw e; // throw语句被暂停，直到finally执行完成
    } finally {
        return false; // 覆盖try.catch中的throw语句
    }
    // 已经执行了"return false"
}

try {
    f();
} catch(e) {
    //这里不会被执行，因为catch中的throw已经被finally中的return语句覆盖了
    console.log('caught outer "bogus"');
}
// 输出
// caught inner "bogus"
```

系统Error对象：我们可以直接使用Error{name, message}对象，例如：throw (new Error('The essage'));

转自 WEB开发者