JavaScript 基础

简介

JavaScript (JS) 最初作为运行在浏览器上的脚本语言，随着 Web 的快速发展而火遍全球。现在 JS 早已不再单单作为 Web 脚本。随着 Node.js 的出现，JS 开始作为后端开发语言；electron 的出现，又让JS 有走进桌面应用的世界，而且这些领域，JS 都有很高的可用性。在后端领域，结合谷歌的 V8 引擎，让 JS 代码拥有了堪比传统编译型语言的性能；在桌面应用领域，也有基于 electron 的 Atom、Slack 这样的成功应用。总之，给人的感觉就是：JS 已经无所不能！

所以，学好 JS 是很有必要的。本节内容介绍 JS 的基本语法和一些核心概念。文中的大部分代码基于 chrome 浏览器的 console。

知识点

JavaScript 数据类型

JavaScript 控制结构

JavaScript 函数与匿名函数

JavaScript 对象

JavaScript 原型

JavaScript 数组

JavaScript 类

简单数据类型

JS 的简单数据类型包括数字、字符串，布尔值，null 和 undefined。

JS 只有一种数字类型，它在内部被表示为一个 64 位的浮点数，和 Java 中的 double 类型一样。有俩个特殊的表示运算结果的数值：NaN 表示不能产生正常结果的运算结果，Infinity 表示一个无穷大的数值。

字符串是被单引号或者双引号包起来的值：

> '' // 空字符串

< ""

> "shiyanlou"

< "shiyanlou"

字符串有一个属性 length 和一些特殊方法：

> 'shiyanlou'.length

< 9

> 'shiyanlou'.toUpperCase()

< "SHIYANLOU"

> 'shiyanlou'.charAt(3)

< "y"

字符串是不可变的，也就是说，一旦一个字符串被创建了，那就无法在修改它。可以通过 ‘+’ 运算符连接字符串创建一个新的字符串：

> 'Hello' + ' ' + 'shiyanlou'

< "Hello shiyanlou"

Null 类型 只有一个 null 值，表示不存在的对象，Undefined 类型也只有 undefine 值，当变量未被初始化时，那它的默认值就是 undefined。想知道它 undefined 的区别可以看这篇文章：undefined与null的区别

变量和常量

JS 中 用 var 或者 let 声明一个变量，用 const 声明常量：

> var num = 42;

> let name = "shiyanlou";

> const PI = 3.14;

使用 var 声明的变量，如果是在函数内部，那么它只在函数内有效，如果在函数外部，那么它是全局有效的。let 声明的变量，只在这个语句所在的代码块内有效，也就是 { } 内。

控制结构

JS 的结构语句基本上和 C 语言一样。

if/else 语句

语法格式是这样的：

if (条件 1) {

当条件 1 为 true 时执行的代码;

} else if (条件 2) {

当条件 2 为 true 时执行的代码;

} else {

当条件 1 和 条件 2 都不为 true 时执行的代码;

}

switch/case 语句

在做大量的选择判断的时候，如果依然使用 if/else 结构，那么代码有可能会变得很凌乱，于是我们采用 switch/case 结构：

switch(k) {

case k1:

执行代码块 1 ;

break;

case k2:

执行代码块 2 ;

break;

default:

默认执行（k 值没有在 case 中找到匹配时）;

}

循环

for 循环是最常用到的结构，格式是这样的：

for(变量 = 初始值; 循环条件; 变量累加方法) {

循环语句;

}

举例说明更清楚，比如循环打印出 0～7 的数字：

<html>

<head></head>

<body>

<script>

for(var i=0;i<8;i++) {

document.write("number is "+i+"<br>");

}

</script>

</body>

</html>

在浏览器中的效果：

JavaScript for 循环操作视频：

您还没有安装flash播放器，请点击这里安装

安装后重启浏览器即可播放视频

Play Video

while 循环

区别于 for 循环的另一种循环方式：

while (条件) {

需要执行的代码;

}

此外，while 循环还有一种变体，称作 do/while 循环：

do {

需要执行的代码;

}

while (条件);

而这两者的区别是，do/while 循环在检测条件之前就会执行，也就是说，即使条件为 false，do/while 也会执行一次循环代码。

break 和 continue 语句

有时候在循环体内，需要立即跳出循环或跳过循环体内其余代码而进行下一次循环，这便是 break 和 continue 的作用。

break 本语句放在循环体内，作用是立即跳出循环。

continue 本语句放在循环体内，作用是中止本次循环，并执行下一次循环。如果循环的条件已经不符合，就跳出循环。

比如：

for (i = 1; i < 10; i++) {

if (i == 5 || i == 6) continue;

if (i == 8) break;

document.write(i);

}

输出为“12347”，便是跳过了 5 和 6，然后在 i==8 的时候跳出了循环。

函数

在 JavaScript 中，函数由关键词 function 定义，函数可以有多个参数。基本格式为：

function 函数名 (参数1，参数2) {

函数体;

return 返回值;

}

比如add 函数：

function add(x, y) {

return x + y;

}

定义函数时，可以不指定函数名，这样函数可以作为值赋给一个变量：

let add = function (x, y) {

return x + y;

}

匿名函数

匿名函数常用在需要函数做参数的地方，JS 的匿名函数也被叫做箭头函数，因为它的基本格式是这样的：

参数 => 语句；

比如说 JS 的数组的 map 方法，可以将传入的函数作用在数组的每个元素上，下面是一个将数组的中数字平方的例子：

> [2, 3, 4, 5].map(x => x\*\*2);

< [4, 9, 16, 25]

匿名函数可以有多个参数或者多个语句，格式如下：

(参数1, 参数2) => {

语句1;

语句2;

};

对象

对象是 JS 中最重要的概念。JS 的对象是可变的键值对集合。对象的的键可以是任意字符串，包括空字符串，值可以是除了 undefined 之外的任意值。下面是一个简单的对象例子，包含一个 value 属性和一个 incrment 方法：

> var myObject = {

'value': 0,

'incrment': function(count) {

this.value += typeof count === 'number' ? count : 1;

}

};

> myObject.incrment()

> myObject.value

< 1

> myObject.incrment(2)

> myObject.value

< 3

在对象中，如果键名是一个合法的 JS 标识符号并且不是 JS 的保留字，那么就可以省略引号。上面的对象可以这样写：

> var myObject = {

value: 0,

incrment: function(count) {

this.value += typeof count === 'number' ? count : 1;

}

};

一般使用 . 访问对象的属性和方法，也可以使用 [] ，但是并不推荐。对于对象中不存在的属性或方法，返回 undefine。

原型

每个对象都会连接到一个原型对象，并且从它那继承属性，所有对象字面量创建的对象都会连接到 Object.prototype，它是 JS 中所有对象的祖先对象，这一点类似 Python 的 object 对象。展开上面 shiyanlou 对象的三角符号，就能看到 shiyanlou 对象从 Object.prototype 继承了哪些东西：

\_\_proto\_\_:

constructor: ƒ Object()

hasOwnProperty: ƒ hasOwnProperty()

isPrototypeOf: ƒ isPrototypeOf()

propertyIsEnumerable: ƒ propertyIsEnumerable()

toLocaleString: ƒ toLocaleString()

toString: ƒ toString()

valueOf: ƒ valueOf()

\_\_defineGetter\_\_: ƒ \_\_defineGetter\_\_()

\_\_defineSetter\_\_: ƒ \_\_defineSetter\_\_()

\_\_lookupGetter\_\_: ƒ \_\_lookupGetter\_\_()

\_\_lookupSetter\_\_: ƒ \_\_lookupSetter\_\_()

get \_\_proto\_\_: ƒ \_\_proto\_\_()

set \_\_proto\_\_: ƒ \_\_proto\_\_()

数组

数组通常是由多个相同类型的值构成的一个集合，每个值都是这个数组的元素。JS 中数组的基本使用方法如下：

> var fruits = ['apple', 'orange', 'banana']

// 获取数组长度

> fruits.length

< 3

// 通过下标访问数组元素

> fruits[1]

< "orange"

// 调用数组方法

> fruits.join(' ');

< "apple orange banana"

类

JS 是基于原型继承的，也就是说对象可以直接从其它对象继承属性。在 ES6 之前，JS 中的类是基于原型的，要声明类，首先要声明它的构造器：

var Dog = function(name) {

this.name = name;

}

如果一个函数前面带上 new 来调用，那么背后会创建一个连接到该函数的 prototype 成员对象，同时 this 会被绑定到这个对象上，所以要为这个类添加方法，需要将它绑定到 prototype 上：

Dog.prototype.bark = function() {

console.log('汪汪汪')

}

现在就可以这样创建对象和调用方法：

> dog = new Dog('旺财');

> dog.bark();

< '汪汪汪'

这种方式和传统的面向对象语言定义类还是有很大差别的，使用起来也不是很方便。在 ES6 中终于引进了 class 关键字来定义类，使用 ES6 语法改写上面的例子：

class Dog1 {

// 构造函数

constructor(name) {

// 属性

this.name ＝ name

}

bark() {

console.log('汪汪汪')

}

}

// 实例化一个对象

> dog = new Dog1('旺财');

> dog.bark();

< '汪汪汪'

JavaScript 类示例操作视频：

您还没有安装flash播放器，请点击这里安装

安装后重启浏览器即可播放视频

Play Video

总结

本节实验学习了 JavaScript 的基本知识，介绍了数据类型、控制结构、函数等编程语言的必备要素，简单的示例学习如何编写 JavaScript 程序。本节的知识点总结如下：

JavaScript 数据类型

JavaScript 控制结构

JavaScript 函数与匿名函数

JavaScript 对象

JavaScript 原型

JavaScript 数组

JavaScript 类

JavaScript 目前不只应用在前端 Web 开发领域，在后端 NodeJS 的应用也非常广泛。学习 JavaScript 入门很简单，但深入很难，必须在项目中学习 JavaScript 一些高级用法，以及 jQuery 这类常用的 JavaScript 库。JavaScript 的内容非常多，仔细学习至少要几个月的时间，而我们本节实验的内容只能是粗浅的入门，在项目中应用到的时候我们会继续介绍

来源: 实验楼

链接: https://www.shiyanlou.com/courses/996

本课程内容，由作者授权实验楼发布，未经允许，禁止转载、下载及非法传播