爱创课堂前端培训

ES5

第1天课堂笔记（本课程共2天）

班级：北京前端训练营27期

讲师：蔡久明

日期：2019年10月17日

爱创课堂官网 ：[www.icketang.com](http://www.icketang.com)

目录

**[一、ES5 2](#_Toc2766)**

**[二、 JSON扩展 2](#_Toc26991)**

[2.1 parse 3](#_Toc20130)

[2.2 stringify 4](#_Toc8247)

[三、 数组 5](#_Toc23201)

[3.1 判断数组和对象 5](#_Toc17878)

[3.2 获得数组中值的索引 5](#_Toc32011)

[3.3 forEach 6](#_Toc21601)

[3.4 map 7](#_Toc24875)

[3.5 fill 8](#_Toc9561)

[3.6 some 8](#_Toc28765)

[3.7 every 9](#_Toc798)

[3.8 filter 10](#_Toc37)

[3.9 reduce和reduceRight 11](#_Toc30494)

[3.10 addNum 12](#_Toc24117)

[3.11 bind 13](#_Toc16989)

[3.12 toJSON 14](#_Toc4426)

# 一、ES5

javascript最早是由网景公司推出的，极其简单，被很多开发者接受，逐渐流行起来，后来IE为了抢占市场微软，将IE浏览器内置在windows系统中，所以IE的市场占有率相当的高。IE脚本语言是Jscript（vbscript）

网景公司为了推广js，与sun公司合作，为了让js更流行，借助当时极其流行的语法java，将js更名为javascript，所以java与javascript关系就像雷锋和雷峰塔。网景公司做了一件好事，将js的语言规范提交给ECMA组织，所以我们学习ECMAScript规范就是在学习javascript规范，所以ECMAScript是js规范的未来。微软很有个性，自己非要研制一套规范，研制的非常不好用，后来自己内部工程师都不干了，非要重新研制新的浏览器，所以微软决定放弃xp系统（放弃IE6,7）。重新研制了IE9浏览器，完全遵守ECMAScript语言规范，所以IE9是微软的第一代高级浏览器（是所有高级浏览器中，最差的一款）。

在国内，我们还要维护IE6,7,8，原因是国内一些企业决定维护xp系统，所以IE6,7就无法淘汰，所以就苦了国内的工程师了，还要维护IE6,7,8

好消息是移动端基本都是webkit内核，因此我们可以放心的使用html5，css3，ES5规范等等

在pc端，由于高级浏览器都实现了html5，css3，ES5规范等等，所以我们可以直接用高级浏览器测试它们

ES规范版本 ES1， ES2， ES3， ES4， ES3.1， ES5， ES6， ES2016， ES2017， ES2018

# JSON扩展

网络中的信息的传输，传输的都是字符串，XML 用来传输数据，现在大多数web开发都是用json数据结构。

前端 ----string---》后端  
 将对象转为字符串:stringify

前端《----string--- 后端

将字符串转为对象:parse

执行代码

|  |
| --- |
| 1. var jsonstr = '{"name":"久哥","age":18,"sex":"男"}'; 2. // console.log(typeof jsonstr); 3. var obj = { 4. name: 'zhanglei', 5. age: 18, 6. sex: '男' 7. } 8. // 将对象转为json字符串 9. var str1 = JSON.stringify(obj); // '{"name":"zhanglei","age":18,"sex":"男"}' 10. console.log(str1); 11. console.log('i love you'); // 'i love you' 12. // 将json字符串转为对象 13. var obj1 = JSON.parse(jsonstr); 14. console.log(obj1); |

结果：

|  |
| --- |
|  |

## 2.1 parse

JSON.parse(str, fn)

str:要解析的json字符串

fn:回调函数，该函数自动执行，可以对要生成的对象进行遍历，如果需要对数据进行操作则使用该方法

function(key, value) {

key:每次遍历元素的属性名

value:每次遍历元素的属性值

}

执行代码：

|  |
| --- |
| 1. var str = '{"name":"久哥","age":18,"sex":"男"}'; 2. var obj = { 3. name: 'zhanglei', 4. age: 18, 5. sex: '男' 6. } 7. /\* 8. JSON.parse(str, fn) 9. \*/ 10. var obj = JSON.parse(str, function(key, value) { 11. // 该种方式会对生成的对象进行遍历，遍历的次数为元素个数 + 1 12. // console.log(this, arguments); 13. // console.log(key, value); 14. if (key) { 15. // 键非空---遍历的过程中 16. return '27期' + value; 17. } else { 18. // 键为空---遍历的最后一次 19. return value; 20. } 21. }); 22. /\* 23. 获得数据，对数据进行处理，如果你感觉这种不好 24. var obj = JSON.parse(str) 25. 遍历对象进行操作 26. for (var i in obj) { 27. obj[i] = '27期' + obj[i]; 28. } 29. \*/ 30. console.log(obj); |

## 2.2 stringify

JSON.stringify(obj, fn)

str:要解析的json字符串

fn:回调函数，该函数自动执行，可以对要生成的对象进行遍历，如果需要对数据进行操作则使用该方法

function(key, value) {

key:每次遍历元素的属性名

value:每次遍历元素的属性值

}

遍历过程和JSON.parse一样的，先遍历每一个元素，

遍历的时候是有前提的，每一次都要求有返回值，如果没有返回值，则遍历立即结束

返回值作为对象当次遍历元素的值，

最后一次，键为空，值为要返回的对象

执行代码：

|  |
| --- |
| 1. var jsonstr = JSON.stringify(obj, function(key, value) { 2. /\* 遍历过程和JSON.parse一样的，先遍历每一个元素， 3. 遍历的时候是由前提的，每一次都要求有返回值， 4. 返回值作为对象当次遍历元素的值， 5. 最后一次，键为空，值为要返回的对象 \*/ 6. // console.log(this, arguments); 7. // console.log(key, value); 8. if (key) { 9. // 键非空---遍历的过程中 10. return '27期' + value; 11. } else { 12. // 键为空---遍历的最后一次 13. return value; 14. } 15. }); 16. console.log(jsonstr); 17. /\* 18. 上面的等价方式： 19. 1、先对对象的数据进行处理 20. for (var i in obj) { 21. obj[i] = '27期' + obj[i]; 22. } 23. 2、将对象转为字符串 24. var jsonstr = JSON.stringify(obj); 25. \*/ |

## 数组

## 3.1 判断数组和对象

四种

isArray

instanceof

constructor

Object.prototype.toString.call() === ‘[Object Array]’

执行代码：

|  |
| --- |
| 1. var obj = document.getElementsByTagName('div'); 2. var arr = [1, 2, 3]; 3. console.log(obj, arr); 4. // ES5 isArray 是Array 5. console.log(Array.isArray(obj)); // false 6. console.log(Array.isArray(arr)); // true 7. // instanceof 8. console.log(obj instanceof Array); // false 9. console.log(arr instanceof Array); // true 10. // constructor 11. console.log(obj.\_\_proto\_\_.constructor === Array); // false 12. console.log(arr.\_\_proto\_\_.constructor === Array); // true 13. // toString 14. console.log(Object.prototype.toString.call(obj) === '[object Array]');// false 15. console.log(Object.prototype.toString.call(arr) === '[object Array]');// true |

结果：

|  |
| --- |
|  |

## 3.2 获得数组中值的索引

indexOf:正序查找

lastIndexOf:倒叙查找

注意：无论是正序还是倒叙，索引不会改变，只是查找的方向不同，如果数组中有多个要查找的值，则查找到第一个就停止查找。查找不到返回 -1

执行代码:

|  |
| --- |
| 1. var arr = [1, 2, 3, 2, 4]; 2. // 获得值的索引 3. // indexOf() 正序查找 4. console.log(arr.indexOf(2)); // 1 5. console.log(arr.indexOf(-2)); // -1 6. // lastIndexOf 倒序查找 7. console.log(arr.lastIndexOf(2)); // 3 8. console.log(arr.lastIndexOf(-2)); // -1 |

结果：

|  |
| --- |
|  |

兼容处理：

|  |
| --- |
| 1. // 进行能力检测 2. if (!Array.prototype.indexOf) { 3. // 该版本的浏览器不支持indexOf 4. Array.prototype.indexOf = function(value) { 5. for (var i = 0; i < this.length; i++) { 6. if (this[i] === value) { 7. return i; 8. } 9. } 10. // 没有找到 11. return -1; 12. } 13. } 14. // 进行能力检测 15. if (!Array.prototype.lastIndexOf) { 16. // 该版本的浏览器不支持lastIndexOf 17. Array.prototype.lastIndexOf = function(value) { 18. for (var i = this.length -1; i >= 0; i--) { 19. if (this[i] === value) { 20. return i; 21. } 22. } 23. // 没有找到 24. return -1; 25. } 26. } |

## 3.3 forEach

遍历数组

arr.forEach(function(value, index, arr) {

value:遍历元素的值

index:遍历元素的索引

arr:被遍历的数组

})

注意：

1. 回调函数内，会忽略return的返回值
2. 整体没有返回值
3. 原数组没有发生变化

执行代码：

|  |
| --- |
| 1. var arr = ['杨浩', '张发', '秋文', '白杰']; 2. var res = arr.forEach(function(value, index, arr) { 3. // console.log(arguments); 4. console.log(value, index, arr); 5. return 111; // 会被忽略的 6. }); 7. console.log(res); 8. console.log(arr); |

结果：

|  |
| --- |
|  |

## 3.4 map

map遍历数组，但是存在映射关系，会产生一个新数组，数组中的元素值为遍历中每一次的返回值。不会改变原数组

var res = arr.map(function(value, index, arr) {

res:最终生成的新的数组

value:元素的值

index：元素的索引

arr：原数组

return:为每一次添加到新生成的数组中的值

});

执行代码：

|  |
| --- |
| 1. var res = arr.map(function(value, index, arr) { 2. // console.log(arguments); 3. console.log(value, index, arr); 4. return '27期' + value; 5. }); 6. console.log(res); 7. console.log(arr); |

结果：

|  |
| --- |
|  |

兼容处理：

|  |
| --- |
| 1. // 能力检测 2. if (!Array.prototype.map) { 3. Array.prototype.map = function(fn) { 4. // 声明一个数组 5. var arr = []; 6. for (var i = 0; i < this.length; i++) { 7. arr.push(fn(this[i], i, this)); 8. } 9. // 返回数组 10. return arr; 11. } 12. } |

## 3.5 fill

给数组填充数据，配合 Array()

执行代码:

|  |
| --- |
| 1. var arr = new Array(5); 2. console.log(arr); 3. // 进行能力检测 4. if (!Array.prototype.fill) { 5. Array.prototype.fill = function(value) { 6. for (var i = 0; i < this.length; i++) { 7. this[i] = value; 8. } 9. } 10. } 11. arr.fill('久哥最帅'); 12. console.log(arr); |

结果：

|  |
| --- |
|  |

## 3.6 some

var res = arr.some(fn);

fn: function(value， index， arr) {

res:最终生成的新的数组

value:元素的值

index：元素的索引

arr：原数组

}

some用于检测数组中的元素，如果有符合条件的，则整体的值为true，后续代码不执行。如果一个符合条件的都没有，则最终结果为false

执行代码;

|  |
| --- |
| 1. var arr = ['杨浩', '张发', '秋文', '白杰']; 2. var res = arr.some(function(value, index, arr) { 3. // console.log(arguments); 4. console.log(value, index, arr); 5. if (value === '秋文') { 6. return true; 7. } 8. }); 9. console.log(res); |

结果：

|  |
| --- |
|  |

## 3.7 every

和some很类似，只不过every要求所有的元素都符合条件，否则就返回false

执行代码：

|  |
| --- |
| 1. var res1 = arr.every(function(value, index, arr) { 2. // console.log(arguments); 3. console.log(value, index, arr); 4. if (value[0] == '1') { 5. return true; 6. } 7. }); 8. console.log(res1); |

结果：

|  |
| --- |
|  |

总结:

some：类似于逻辑或，一真则真

every:类似于逻辑与，一假则假

兼容处理:

|  |
| --- |
| 1. // 能力检测 2. if (!Array.prototype.some) { 3. Array.prototype.some = function(fn) { 4. // 循环 5. for (var i = 0; i < this.length; i++) { 6. if (fn(this[i], i, this)) { 7. return true; 8. } 9. } 10. } 11. } 12. // 能力检测 13. if (!Array.prototype.every) { 14. Array.prototype.every = function(fn) { 15. // 循环 16. for (var i = 0; i < this.length; i++) { 17. if (!fn(this[i], i, this)) { 18. return false; 19. } 20. } 21. } 22. } |

## 3.8 filter

过滤数组

var res = arr.filter(function(value, index, arr) {

// console.log(arguments);

if (value % 2 == 1) {

return true;

}

});

执行代码：

|  |
| --- |
| 1. var arr = [1,2,3,4,5]; 2. var res = arr.filter(function(value, index, arr) { 3. // console.log(arguments); 4. if (value % 2 == 1) { 5. return true; 6. } 7. }); 8. console.log(res); 9. console.log(arr); |

结果：

|  |
| --- |
|  |

兼容写法：

|  |
| --- |
| 1. // 能力检测 2. if (!Array.prototype.filter) { 3. Array.prototype.filter = function(fn) { 4. // 定义一个新数组 5. var arr = []; 6. for (var i = 0; i < this.length; i++) { 7. if (fn(this[i], i, this)) { 8. arr.push(this[i]); 9. } 10. } 11. // 返回数组 12. return arr; 13. } 14. } |

## 3.9 reduce和reduceRight

reduce：fn 回调函数

数组的遍历，遍历的时候从第二个元素开始的

fn:

参数： prev， value， index，arr

prev:第一次为第一个元素的值，之后为上一次回调函数的返回值

value:元素的值

index：元素的索引

arr：数组

reduce返回值：

回调函数最后一次的返回值

执行代码：

|  |
| --- |
| 1. var arr = [1, 2, 3, 4, 5, 6, 7]; 2. /\* 3. reduce：fn 回调函数 4. 数组的遍历，遍历的时候从第二个元素开始的 5. fn: 6. 参数： prev， value， index，arr 7. prev:第一次为第一个元素的值，之后为上一次回调函数的返回值 8. value:元素的值 9. index：元素的索引 10. arr：数组 11. 返回值： 12. 回调函数最后一次的返回值 13. \*/ 14. var res = arr.reduce(function(prev, value, index, arr) { 15. // console.log(arguments); 16. console.log(prev, value, index, arr); 17. // 数组值的累加 18. return prev + value; 19. // return prev \* value; 20. // return 111; 21. }); 22. console.log(res); 23. console.log(arr); |

结果：

|  |
| --- |
|  |

reduceRight和reduce一样，只是倒序的过程

兼容写法：

|  |
| --- |
| 1. if (!Array.prototype.reduce) { 2. Array.prototype.reduce = function(fn) { 3. var res = this[0]; 4. for (var i = 1; i < this.length; i++) { 5. res = fn(res, this[i], i, this); 6. } 7. return res; 8. } 9. } 10. if (!Array.prototype.reduceRight) { 11. Array.prototype.reduceRight = function(fn) { 12. var res = this[this.length - 1]; 13. for (var i = this.length - 2; i >= 0; i--) { 14. res = fn(res, this[i], i, this); 15. } 16. return res; 17. } 18. } |

## 3.10 addNum

addNum(num1, num2)

需求：

1. 求num1~num2累加的和
2. 分类
   1. 算边界
   2. 不计算边界

实现带边界的

addNum(1, 5) === 1 + 2+3+4+5 = 15

addNum(5, 1) === 1 + 2+3+4+5 = 15

要求：不能使用for循环

执行代码;

|  |
| --- |
| 1. function addNum(num1, num2) { 2. var max = Math.max(num1, num2); 3. var min = Math.min(num1, num2); 4. // return (min + max) \* (max - min + 1) / 2; 5. // 创建数组 6. var arr = new Array(max - min + 1); 7. console.log(arr); 8. // 填充数组 9. arr.fill(min); 10. console.log(arr); 11. // 更改数组为目标数组 12. arr = arr.map(function(value, index) { 13. return value + index; 14. }); 15. console.log(arr); 16. // 累加 17. var res = arr.reduce(function(prev, value) { 18. return prev + value; 19. }); 20. // 返回最终的结果 21. return res; 22. } 23. var res = addNum(100, 1); 24. console.log(res); // 55 |

## 3.11 bind

call和apply直接执行，而bind返回一个函数，不会立即执行

执行这个返回函数的时候，如果添加实参，实参会和原来绑定时候传入的参数进行合并，作为所有实参传给函数进行调用

执行代码：

|  |
| --- |
| 1. function demo(a,b) { 2. console.log(this, arguments); 3. } 4. /\* 5. call和apply直接执行，而bind返回一个函数，不会立即执行 6. \*/ 7. demo.call(document, 1, 2, 3); 8. demo.apply(document, [1, 2, 3]); 9. var res = demo.bind(document, 1, 2, 3); 10. console.log(res); 11. res(4, 5, 6); |

结果：

|  |
| --- |
|  |

自己实现：

|  |
| --- |
| 1. // 1、让原来的函数，失效 2. Function.prototype.bind = null; 3. // 2、自定义实现 4. Function.prototype.bind = function(target){ 5. /\* 6. 获得调用该方法的函数---this,因为函数不是立即执行， 7. 但是执行的代码必须为this函数实例的函数体 8. 参数的问题： 9. 1、绑定时传入的参数 10. 2、调用时传入的参数 11. 函数执行时的参数 = 绑定时传入的参数 + 调用时传入的参数 12. \*/ 13. // console.log(arguments); 14. // 类数组对象调用数组方法 15. var arg1 = [].slice.call(arguments, 1); 16. // console.log(arg1); 18. // console.log(Array.isArray(arguments)); 19. var me = this; 20. // 自定义一个函数 21. var fun = function() { 22. // console.log(arguments); 23. // 获得调用函数时传入的参数 24. var arg2 = [].slice.call(arguments); 25. // console.log(arg2); 26. // 进行数组的合并 27. var arg = arg1.concat(arg2); 28. // console.log(arg); 29. me.apply(target, arg); 30. } 31. // 返回一个函数 32. return fun; 33. } 34. var res = demo.bind(document, 1, 2, 3,888); 35. console.log(res); 36. res(4, 5, 6,333); |

## 3.12 toJSON

格式化时间

执行代码;

|  |
| --- |
| 1. var date = new Date(); 2. console.log(date); 3. // toJSON格式化时间 4. console.log(date.toJSON()); |

结果：

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |