爱创课堂前端培训

# ECMAScript 5.0

第1天课堂笔记（本课程共2天）

班级：北京前端训练营19期

讲师：彭帅伟

日期：2018年12月22日

彭帅伟老师

QQ: 284337908

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

目录

[ECMAScript 5.0 1](#_Toc3944)

[一、ES5 3](#_Toc20733)

[二、 JSON拓展 4](#_Toc14241)

[2.1 parse 4](#_Toc1789)

[2.2 stringify 4](#_Toc16064)

[三、 数组拓展 5](#_Toc22241)

[3.1 判断数组 5](#_Toc32113)

[3.2 获取成员的索引 6](#_Toc30039)

[3.3 forEach 7](#_Toc24981)

[下午复习： 8](#_Toc7423)

[3.4 map 10](#_Toc13952)

[3.5 fill 10](#_Toc25133)

[3.6 some 11](#_Toc9288)

[3.7 every 11](#_Toc17795)

[3.8 filter 12](#_Toc18012)

[3.9 reduce 13](#_Toc1189)

[3.10 reduceRight 13](#_Toc31738)

[3.11 实现addNum方法 13](#_Toc24924)

[四、函数绑定 14](#_Toc26948)

[五、日期拓展 15](#_Toc6868)

# 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拓展

ES5为了json字符串和js对象互转方便，提供了JSON内置对象，并分别提供了两个方法

## 2.1 parse

作用: 用于将json字符串解析为js对象

使用方式：

JSON.parse(str, fn)

str: 要被处理的字符串

fn: 处理的函数

返回值就是本次处理的结果

有两个参数

第一个参数：属性名

第二个参数：属性值

从外部向内部遍历

举例：

|  |
| --- |
| 1. // 定义json字符串 2. var str = '{"a": 1, "b": "2", "c": {"d": 3}}'; 3. // 将str转为js对象 4. var result = JSON.parse(str, function(key, value) { 5. console.log(arguments); 6. // 我们想要让字符串转为数字 7. if (typeof value === "string") { 8. return +value; 9. } 10. return value; 11. }); |

结果：

|  |
| --- |
|  |

## 2.2 stringify

作用：将js对象转为json字符串

使用方式：

JSON.stringify(obj, fn)

obj: 被处理的对象

fn: 处理的函数

返回值就是本次处理的结果

有两个参数：

第一个参数： 属性名

第二个参数： 属性值

从内部向外部遍历

举例：

|  |
| --- |
| 1. // 定义对象 2. var obj = { 3. a: 1, 4. b: "2", 5. c: { 6. d: 3 7. } 8. } 9. // 将obj转为字符串 10. var result = JSON.stringify(obj, function(key, value) { 11. // console.log(arguments); 12. // 我们想要将所有的字符串转为数字 13. if (typeof value === "string") { 14. return +value; 15. } 16. return value; 17. }); |

结果：

|  |
| --- |
|  |

# 数组拓展

## 3.1 判断数组

第一种方式判断对象的类型

Object.prototype.toString.call(obj)

第二种方式判断是否是实例化对象

obj instanceof Array

第三种方式判断构造函数是否是Array

obj.constructor === Array

第四种方式数组的静态方法

Array.isArray(obj)

举例：

|  |
| --- |
| 1. // 获取类数组对象 2. var obj = document.getElementsByTagName("div"); 3. // 定义数组 4. var arr = []; 5. // 请问有几种方式能确定arr是一个数组？ 6. // 判断实例化对象 7. console.log(obj instanceof Array); 8. console.log(arr instanceof Array); 9. // 构造函数是Object的时候， 也是true 10. console.log(arr instanceof Object); 11. // toString 12. console.log(Object.prototype.toString.call(obj) === "[object Array]"); 13. console.log(Object.prototype.toString.call(arr) === "[object Array]"); 14. // constructor 15. console.log(obj.constructor === Array); 16. console.log(arr.constructor === Array); 17. console.log(arr.constructor === Object); // false 18. // 数组的静态方法 19. console.log(Array.isArray(obj)); 20. console.log(Array.isArray(arr)); |

## 3.2 获取成员的索引

ES5为数组拓展了两个方法用于获取数组中成员的索引: indexOf, lastIndexOf

参数就是要查找的成员

如果找到该成员，会返回成员的索引

如果找不到该成员， 返回-1

在查找的时候是真正的全等查找， 不会做类型转换

举例：

|  |
| --- |
| 1. // 兼容IE 2. if (!Array.prototype.indexOf) { 3. // 拓展方法 4. Array.prototype.indexOf = function(item) { 5. // 遍历数组， 就是在遍历this 6. for (var i = 0; i < this.length; i++) { 7. // 查找成员 8. if (this[i] === item) { 9. // 说明找到该成员了， 返回索引值 10. return i; 11. } 12. } 13. // 循环完毕之后， 没有找到该成员， 返回-1 14. return -1; 15. } 16. } |

## 3.3 forEach

该方法用于代替for循环，并没有移除for循环，只是将for循环封装在了数组迭代器forEach方法的内部

使用方式：

数组.forEach(fn)

fn: 执行的函数

函数中有三个参数：

第一个参数： 成员值

第二个参数： 索引值

第三个参数： 原数组

返回值对forEach的执行结果没有影响

作用域是window

forEach方法的返回值始终 是undefined

jquery中也有一个类似的方法， each方法， jquery中的each方法与forEach方法的区别：

jquery中的each方法中的函数中的参数，第一个是索引值， 第二个是成员值

举例：

|  |
| --- |
| 1. // 兼容IE 2. if (!Array.prototype.forEach) { 3. // 拓展该方法 4. Array.prototype.forEach = function(fn) { 5. // 遍历数组, 就是在遍历this 6. for (var i = 0; i < this.length; i++) { 7. // 执行函数并传递参数 8. // 传递三个参数: 9. // 成员值 this[i], 索引值 i, 原数组 this 10. fn(this[i], i, this); 11. } 12. } 13. } |

## 下午复习：

JSON拓展：

分别提供了两个方法：

parse:

作用：将json字符串解析为js对象

使用方式：

JSON.parse(str, fn)

stringify:

作用: 将js对象转为json字符串

使用方式：

JSON.stirngify(obj, fn)

数组拓展：

判断数组：

第一种方式判断是否是实例化对象

obj instanceof Array

第二种方式判断构造函数是否是Array

obj.constructor === Array

第三种方式判断对象的类型

Object.prototype.toString.call(obj)

第四种方式数组的静态方法

Array.isArray(obj)

获取成员的索引：

ES5提供了两个方法，用于获取数组中成员的索引值， indexOf, lastIndexOf

indexOf: 从前向开始查找

lastIndexOf: 从后向前开始查找

参数就是要查找的成员

如果没有找到该成员返回-1

forEach:

接受一个参数： 函数

有三个参数：

第一个参数： 成员值

第二个参数： 索引值

第三个参数： 原数组

作用域： window

forEach方法的返回值始终是undefined

## 3.4 map

该方法用于遍历数组并映射结果， 与forEach方法类似，只是map方法的返回值是有意义的

使用方式：

数组.map(fn)

fn: 执行的函数

有三个参数：

第一个参数： 成员值

第二个参数： 索引值

第三个参数： 原数组

作用域： window

返回值就是执行结果组成的成员

map方法的返回值是一个新的数组， 数组中的成员都是每一个函数执行结果组成的成员

举例：

|  |
| --- |
| 1. // 兼容IE 2. if (!Array.prototype.map) { 3. // 拓展该方法 4. Array.prototype.map = function(fn) { 5. // 定义结果容器 6. var result = []; 7. // 遍历数组, 就是在遍历this 8. for (var i = 0; i < this.length; i++) { 9. // 执行函数并传递三个参数 10. // 成员值 this[i], 索引值 i, 原数组 this 11. result.push(fn(this[i], i, this)); 12. } 13. // 返回新的数组 14. return result; 15. } 16. } |

## 3.5 fill

该方法用于填充数组

我们通过Array(len),或者是 new Array(len) , 得到的只有长度没有成员， 所以我们不能使用数组的迭代器方法（forEach, map）进行遍历， 这个时候就需要往数组中填充成员

参数就是要填充的成员， 即使传递的是函数也不会执行

举例：

|  |
| --- |
| 1. // 兼容IE 2. if (!Array.prototype.fill) { 3. // 添加该方法 4. Array.prototype.fill = function(item) { 5. // 遍历数组，就是在遍历this 6. for (var i = 0; i < this.length; i++) { 7. // 填充数组 8. this[i] = item; 9. } 10. // 返回数组 11. return this; 12. } 13. } |

断言方法

## 3.6 some

判断数组中是否有成员满足条件

使用方式与forEach方法类似

接受一个参数就是要执行的函数

有三个参数： 成员值、 索引值、 原数组

返回值：判断的依据

some方法的返回值：

true: 至少有一个成员满足条件

false: 都不满足条件

some方法对true比较敏感， 一旦遇到满足条件的成员就立即停止遍历

举例：

|  |
| --- |
| 1. /\*\* 2. \* some 实现some方法 3. \* @arr 要判断的数组 4. \* @fn 要执行的函数 5. \* return bool 判断数组中是否有成员满足条件 6. \*/ 7. function some(arr, fn) { 8. // 遍历数组 9. for (var i = 0; i < arr.length; i++) { 10. // 执行函数并判断结果 11. // 函数中要传递三个参数: 成员值 arr[i], 索引值 i, 原数组 arr 12. if (fn(arr[i], i, arr)) { 13. // 遇到满足条件的成员，停止遍历并返回true 14. return true; 15. } 16. } 17. // 循环完毕后， 没有找到符合条件的成员返回false 18. return false; 19. } |

## 3.7 every

该方法用于判断数组中成员是否都满足条件

使用方式与forEach方法类似

接收一个参数就是要执行的函数

函数中有三个参数： 成员值、 索引值、 原数组

返回值就是判断的依据

every方法的返回值：

true: 都满足条件

false: 至少有一个是不满足条件

every对false敏感， 一旦遇到一个不满足条件的成员的时候立即停止遍历

举例：

|  |
| --- |
| 1. /\*\* 2. \* every 实现every方法 3. \* @arr 要判断的数组 4. \* @fn 要执行的函数 5. \* return bool 判断数组中是否都满足条件 6. \*/ 7. function every(arr, fn) { 8. // 遍历数组 9. for (var i = 0; i < arr.length; i++) { 10. // 执行函数并判断结果 11. // 传递三个参数： 成员值 arr[i], 索引值 i, 原数组 arr 12. if (!fn(arr[i], i, arr)) { 13. // 遇到不满足条件的成员就停止遍历并返回false 14. return false; 15. } 16. } 17. // 遍历完成， 都满足条件就返回true 18. return true; 19. } |

## 3.8 filter

实现对数组的过滤

使用方式和forEach方法类似

参数就是要执行的函数

有三个参数：成员值、 索引值、 原数组

作用域是window

返回值就是过滤的条件

filter方法返回值是满足过滤条件组成的新的成员

举例：

|  |
| --- |
| 1. // 兼容IE 2. if (!Array.prototype.filter) { 3. // 拓展该方法 4. Array.prototype.filter = function(fn) { 5. // 创建新的数组 6. var result = []; 7. // 遍历数组, 就是在遍历this 8. for (var i = 0; i < this.length; i++) { 9. // 执行函数传递参数并判定结果 10. // 传递三个参数: 成员值 this[i]， i, 原数组 this 11. if (fn(this[i], i, this)) { 12. // 满足条件的成员要放入新的数组中 13. result.push(this[i]); 14. } 15. } 16. // 返回新的数组 17. return result; 18. } 19. } |

## 3.9 reduce

## 3.10 reduceRight

这两个方法是累加的方法， reduce是从前向后遍历， reduceRight是从后向前开始遍历

会对每一个成员逐一处理，并将结果返回

参数就是要执行的函数

有四个参数：上一次的累积结果，当前成员值， 当前索引值， 原数组

返回值就是当次累积的结果， 将会在下一次遍历的时候作为第一个参数传递

reduce是从第二个成员开始遍历，第一个成员会在第一次遍历的时候作为第一个参数传递

reduceRight是从倒数第二个成员开始遍历，倒数第一个成员在第一次遍历的时候作为第一个参数传递

举例：

|  |
| --- |
| 1. // 兼容IE reduceRight 2. if (!Array.prototype.reduceRight) { 3. // 拓展该方法 4. Array.prototype.reduceRight = function(fn) { 5. // 由于是从倒数第二个成员开始遍历，当次累积的结果是倒数第一个成员 6. var result = this[this.length - 1]; 7. // 循环， 从倒数第二个成员开始遍历 8. for (var i = this.length - 2; i >= 0; i--) { 9. // 执行函数传递参数 10. // 有四个参数: 上一次的累积结果， 当前成员值 this[i], 当前索引值i, 原数组 this 11. result = fn(result, this[i], i, this) 12. } 13. // 返回结果 14. return result; 15. } 16. } |

## 3.11 实现addNum方法

addNum(num1, num2)接受两个参数分别是，两个整数， 求两个整数之和

比如：

addNum(1, 100)

一种是包含两个参数， 另一种是不包含两个参数

我们统一包含两个参数

不要使用for循环

|  |
| --- |
| 1. // 定义一个函数 2. function addNum(num1, num2) { 3. // 确定最大值和最小值 4. var max = Math.max(num1, num2); 5. var min = Math.min(num1, num2); 6. // console.log(max); 7. // console.log(min); 8. // (5, 10) 5, 6, 7, 8, 9, 10 10 - 5 + 1 9. // (5, 10) 6, 7, 8, 9 10 - 5 - 1 10. // 创建数组 11. return Array(max - min + 1) 12. // 填充数组 13. .fill(1) 14. // 构建由最小值到最大值之间的数组 15. .map(function(value, index, arr) { 16. // index是递增的， 由min加上index 即可得到从最小值到最大值之间的数字 17. return min + index; 18. }) 19. // 求相加的结果 20. .reduce(function(pre, value, index, arr) { 21. return pre + value; 22. }) 23. } |

# 四、函数绑定

ES5对函数拓展了bind方法

作用：为函数绑定作用域（当函数执行的时候，改变函数的作用域，并传递参数）

call与apply的区别

他们都是改变函数作用域的方法，都是在调用该方法的时候，执行函数并改变作用域的，第一个参数都是改变的作用域

call 从第二个参数开始，表示传递给函数的参数

apply 第二个参数是数组，每一个成员表示传递给函数的参数

bind跟call类似

第一个参数表示改变的作用域对象

从第二个参数开始，表示传递的参数

区别：

call | apply 调用即执行

bind调用不执行，但是得到一个新的方法，可以执行

# 五、日期拓展

toJSON 将日期转化成json格式，（标准化格式）

|  |
| --- |
| 1. var date = new Date(); 2. console.log(date) 3. console.log(date.toJSON()) |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |