# 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

## 2.1 parse

该方法用于将json字符串解析为js对象

使用方式：

JSON.parse(str, fn)

str: 要处理的字符串

fn: 处理的函数

有两个参数

第一个参数： 属性名

第二个参数： 属性值

返回值就是本次处理的结果

举例：

|  |
| --- |
| 1. // 定义一个json字符串 2. var str = '{"a": 1, "b": "2", "c": {"d": 3}}'; 3. // 我们想要将str解析为js对象 4. var result = JSON.parse(str, function(key, value) { 5. // console.log(arguments); 6. // 我们想要让value都作为数字保留 7. if (typeof value == "string") { 8. return +value; 9. } 10. return value; 11. }); |

结果：

|  |
| --- |
|  |

## 2.2 stringify

该方法用于将js对象解析为json字符串

使用方式：

JSON.stringify(obj, fn)

obj: 要处理的对象

fn: 要处理的函数

函数中有两个参数

第一个参数： 属性名

第二个参数: 属性值

返回值就是本次处理的结果

举例:

|  |
| --- |
| 1. // 定义一个对象 2. var obj = { 3. a: 1, 4. b: "2", 5. c: { 6. d: 3 7. } 8. } 9. // 我们想要将obj转为json字符串 10. var result = JSON.stringify(obj, function(key, value) { 11. console.log(arguments); 12. // 我们想要value都作为数字保留 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)

第二种方式判断实例化对象

obj instanceof Array

第三种方式判断构造函数

obj.construcotr === Array

第四种数组的静态方法

Array.isArray(obj)

举例：

|  |
| --- |
| 1. // 获取类数组对象 2. var obj = document.getElementsByTagName("div"); 3. // // 定义一个数组 4. var arr = []; 5. // 请问有几种方式能够确定一个数组？ 6. // 第一种方式 intanceof 7. console.log(obj instanceof Array); 8. console.log(arr instanceof Array); 9. // 当obj的构造函数是Object的时候，返回的也是true 10. console.log(obj instanceof Object); 11. // // constructor 12. console.log(obj.constructor === Array); 13. console.log(arr.constructor === Array); 14. // 这里返回false 15. console.log(arr.constructor === Object); 16. // // 判断对象的类型 17. console.log(Object.prototype.toString.call(obj) === "[object Array]"); 18. console.log(Object.prototype.toString.call(arr) === "[object Array]"); 19. // Array的静态方法 20. console.log(Array.isArray(obj)); 21. console.log(Array.isArray(arr)); |

## 3.2 获取数组成员的索引

在ES5中为数组拓展了两个方法用于获取成员的索引: indexOf、lastIndexOf

参数就是要查找的成员

如果找到成员就返回索引

如果没有找到该成员就返回-1

在查找的过程中是真正的全等查找，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，就是遍历this 6. for (var i = 0; i < this.length; i++) { 7. // 查找并判断成员 8. if (this[i] === item) { 9. // 找到该成员返回索引值 10. return i; 11. } 12. } 13. // 遍历完成没有找到成员 返回-1 14. return -1; 15. } 16. } 17. // var pos = arr.indexOf(3); 18. // alert(pos); 19. // 兼容lastIndexOf方法 20. if (!Array.prototype.lastIndexOf) { 21. // 拓展该方法 22. Array.prototype.lastIndexOf = function(item) { 23. // 遍历数组, 就是遍历this 24. for (var i = this.length - 1; i >= 0; i--) { 25. // 查找并判断 26. if (this[i] === item) { 27. // 找到成员 返回索引值 28. return i; 29. } 30. } 31. // 遍历完成没有找到成员 返回-1 32. return -1; 33. } 34. } |

## 3.3 forEach方法

该方法是数组的迭代器方法，用于替代for循环，但是并没有将for循环移除， 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：要处理的函数

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

作用域是window

函数中的返回值对forEach方法的执行结果没有影响

forEach方法的返回值始终是undefined

在jQuery中也有一个类似的方法，叫做each，它与forEach方法的区别是：

在jQuery中的each方法有两个参数：

第一个参数：索引值

第二个参数： 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，遍历this 6. for (var i = 0; i < this.length; i++) { 7. // 执行fn并且传递三个参数 8. // 成员值: this[i] 索引值: i 原数组: this 9. fn(this[i], i, this); 10. } 11. } 12. } |

## 下午复习：

JSON拓展：

ES5中为了json字符串与js对象互转方便提供了JSON内置对象

分别提供了两个方法：

该方法用于将json字符串转为js对象

parse(str, fn)：

str: 要处理字符串

fn: 要执行的函数

用于将js对象转为json字符串

stringify(obj, fn)

obj: 要处理的对象

fn: 处理的函数

数组拓展：

获取数组中成员的索引值：

提供了两个方法：

indexOf: 从前向后开始查找

lastindexOf: 从后向前开始查找

参数就是要查找的成员

如果找到成员就返回索引值

如果没有找到就返回-1

判断数组：

第一种判断对象的类型

Object.prototype.toString.call(obj)

第二种判断实例化对象

obj instanceof Array

第三种判断构造函数

obj.constructor === Array

第四种数组的静态方法

Array.isArray(obj)

forEach方法：

用于替代for循环，但是并没有将for循环移除， 只不过将for循环封装在了该方法的内部

接收一个参数是执行的函数

函数中有三个参数： 成员值、 索引值、 原数组

## 3.4 map方法

该方法用于遍历数组并映射结果， 它与forEach方法类似，只不过它的返回值是有意义的

使用方式：

[].map(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

返回值就是执行结果的成员

map方法的返回值是一个新的数组，数组中的每一项都是每一次执行结果得到的新成员

举例：

|  |
| --- |
| 1. // 兼容IE 2. if (!Array.prototype.map) { 3. // 添加该方法 4. Array.prototype.map = function(fn) { 5. // 定义结果容器数组 6. var result = []; 7. // 遍历数组，就是遍历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. // 遍历完成之后， 返回新的数组 14. return result; 15. } 16. } |

## 3.5 fill方法

该方法用于填充数组

通过Array(len)或者是new Array(len)得到的是一个空数组，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， 所以我们要填充数组

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. // 拓展该方法 4. Array.prototype.fill = function(item) { 5. // 遍历数组， 就是遍历this 6. for (var i = 0; i < this.length; i++) { 7. // 填充数组成员 8. this[i] = item; 9. } 10. // 返回数组 11. return this; 12. } 13. } |

断言方法

## 3.6 some

some方法用于判断数组中是否有成员满足条件

使用方式与forEach方法类似

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

作用域是window

返回值就是判断的依据

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，一旦遇到一个满足条件的成员就立即停止遍历返回结果

举例：

|  |
| --- |
| 1. // 兼容IE 2. if (!Array.prototype.some) { 3. // 拓展该方法 4. Array.prototype.some = function(fn) { 5. // 遍历数组 6. for (var i = 0; i < this.length; i++) { 7. // 执行fn并判断结果 8. if (fn(this[i], i, this)) { 9. // 找到该成员， 停止遍历返回true 10. return true; 11. } 12. } 13. // 遍历完成没有找到， 返回false 14. return false; 15. } 16. } |

## 3.7 every

用于判断数组中是否都满足条件

使用方式与forEach方法类似

参数要执行的函数

有三个参数： 成员值、索引值、原数组

作用域是window

返回值就是判断的依据

every方法的返回值：

true: 都满足条件

false: 至少有一个成员是不满足条件的

every方法对false比较敏感，一旦遇到一个不满足条件的立即停止并返回结果

|  |
| --- |
| 1. // 兼容IE 2. if (!Array.prototype.every) { 3. // 拓展该方法 4. Array.prototype.every = function(fn) { 5. // 遍历数组就是遍历this 6. for (var i = 0; i < this.length; i++) { 7. // 执行fn并判断结果 8. // 传递三个参数: 成员值: this[i]、索引值: i、 原数组 this 9. if (!fn(this[i], i, this)) { 10. // 当有成员不满足条件的时候停止遍历并返回false 11. return false; 12. } 13. } 14. // 都满足条件返回true 15. return true; 16. } 17. } |

## 3.8 filter

实现对数组的过滤

使用方式与forEach方法类似

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

作用域：window

返回值就是过滤的条件

filter方法的返回值是每一次满足过滤条件组成的新的成员数组

|  |
| --- |
| 1. // 兼容IE 2. if (!Array.prototype.filter) { 3. // 拓展该方法 4. Array.prototype.filter = function(fn) { 5. // 创建结果数组容器 6. var result = []; 7. // 遍历数组， 就是遍历this 8. for (var i = 0; i < this.length; i++) { 9. // 执行函数并判断结果 10. if (fn(this[i], i, this)) { 11. // 满足条件的成员放入到数组结果容器中 12. result.push(this[i]); 13. } 14. } 15. // 返回新数组 16. return result; 17. } 18. } |

累加方法

## 3.9 reduce

## 3.10 reduceRight

这两个是累加的方法，reduce是从前向后开始累加，reduceRight是从后向前开始累加

会对所有的成员逐一处理，并将结果返回

参数就是要执行的函数

有四个参数：上一次累积的结果， 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，倒数第一个成员将会在第一次遍历的时候作为第一个参数传递

|  |
| --- |
| 1. /\*\* 2. \* 实现reduceRight方法 3. \* @arr 要计算的数组 4. \* @fn 要执行的函数 5. \* return 累积的结果 6. \*\*/ 7. function reduceRight(arr, fn) { 8. // 定义当次累积的结果 9. // 由于reduceRight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 10. var result = arr[arr.length - 1]; 11. // 遍历数组 12. for (var i = arr.length - 2; i >= 0; i--) { 13. // 执行fn传递四个参数: 14. // 上一次的累积结果: result, 当前成员值 arr[i], 当前索引值i, 原数组 arr 15. result = fn(result, arr[i], i, arr) 16. } 17. // 返回累积结果 18. return result; 19. } |

## 3.11 实现addNum方法

addNum(num1, num2)接收两个参数， 分别都是整数， 求两个正整数之间所有的整数之和

比如：

addNum(1, 10)

第一种是包含两个参数

第二种是不包含两个参数

我们统一实现的是包含两个参数的

要求: 不要使用for循环

|  |
| --- |
| 1. // 定义函数 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(min, max); 7. // (5, 10) 5, 6, 7, 8, 9, 10 10 - 5 + 1 8. // (5, 10) 6, 7, 8, 9 10 - 5 - 1 9. // 创建数组确定长度 10. var arr = new Array(max - min + 1); 11. // console.log(arr); 12. // 为了遍历数组，填充数组 13. arr.fill(); 14. // console.log(arr); 15. // 调用map方法可以得到一个新的数组 16. var result = arr.map(function(value, index, arr) { 17. // 我们想要得到一个从最小值到最大值之间的数组 18. // 因为index每次递增，所以我们应该用min + index 即可得到一个从最小值到最大值之间的数组 19. // console.log(value, index); 20. return min + index; 21. }) 22. // console.log(result); 23. // 调用累加方法 24. return result.reduceRight(function(pre, value, index, arr) { 25. return pre + value; 26. }) 27. } |

# 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()) |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |