爱创课堂前端培训

# JavaScript进阶

第4天课堂笔记（本课程共8天）

班级：北京前端训练营19期

讲师：张容铭

日期：2018年12月17日

张容铭老师

微博：@张容铭\_YYQH

QQ : 286031482

E-mail : yuye\_qinghe@qq.com

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

目录

[JavaScript进阶 1](#_Toc26444)

[复习 3](#_Toc7446)

[一、滚轮事件 4](#_Toc21960)

[1.1事件对象 5](#_Toc11247)

[下午复习 7](#_Toc15730)

[二、键盘事件 8](#_Toc20209)

[2.1触发键盘事件的元素 8](#_Toc651)

[2.2 tabindex 8](#_Toc16620)

[2.3事件对象 8](#_Toc1517)

[三、面向对象 9](#_Toc21361)

[3.1构造函数 9](#_Toc32380)

[3.2构造函数执行的四个步骤 10](#_Toc18184)

# 复习

dom中

clientWith, clientHeight: content+padding

offsetWidth, offsetHeight: content+padding+border

clientLeft, clientTop: 边框的宽高

jquery:

$(dom).width() content

$(dom).innerWidth() content + padding

$(dom).outerWidth() content + padding + border

$(dom).outerWidth(true) content + padding + border + margin

定位元素

offsetParent 获取定位父元素

offsetLeft 距离左侧距离

offetTop 距离上边的距离

$(dom).position() left, top

滚动事件 onscroll

获取顶部距离

document.documentElement.offsetTop

document.body.offsetTop

检测浏览器

navigator.userAgent

节流 设置一个开关，让程序执行，程序执行完毕，打开开关。

# 一、滚轮事件

我们常常在浏览器中，滚动鼠标滚动，为了监听这一操作，浏览器为我们提供了滚轮事件

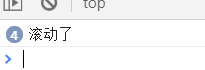
事件名称: mousewheel

兼容性：在火狐浏览器下，滚动事件叫 DOMMouseScroll

在火狐浏览器中，我们要使用DOM2.0订阅事件的方法来订阅

注意：滚轮事件也是一个高频事件

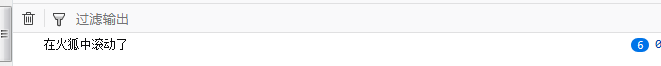
chrome



IE浏览器



在火狐中



|  |
| --- |
| 1. // 监听滚轮事件 2. window.onmousewheel = function() { 3. // 输出内容 4. console.log('滚动了') 5. } 6. // 在火狐中，监听事件 7. window.addEventListener('DOMMouseScroll', function() { 8. console.log('在火狐中滚动了') 9. }) 10. // 小于等于ie8,我们要通过为document绑定 11. document.onmousewheel = function(e) { 12. console.log(123, e, window.event) 13. } 14. // 兼容浏览器 15. bindEvent(window, 'mousewheel', function(e) { 16. // 兼容 17. e = e || window.event; 18. console.log('滚动了', e) 19. }) |

## 1.1事件对象

在点击事件中，我们比较关心：点击的元素，事件触发的横纵坐标等等

在滚轮事件中，我们比较关心滚轮的方向

滚轮方向的信息也在事件对象中

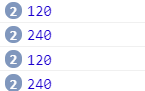
在标准事件中，通过wheelDelta表示滚动方向信息

在chrome中

向下滑动，每次滑动的是-120的倍数



向上滑动，每次滑动的是120的倍数



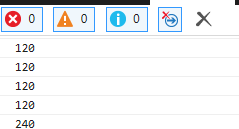
注意：新版本中，每次滚动的数字是150的倍数

在IE浏览器中

向下滑动，每次滑动的是-120的倍数

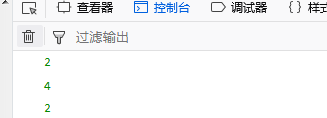


向上滑动，每次滑动的是120的倍数



在firefox浏览器中

向下滑动，每次滑动的是2的倍数



向上滑动，每次滑动的是-2的倍数



注意：在之前的版本中，滑动是3的倍数。

|  |
| --- |
| 1. // 绑定事件 2. bindEvent(window, 'mousewheel', function(e) { 3. // 兼容事件对象 4. e = e || window.event; 5. // 标准浏览器下，滚轮滚动信息 6. // console.log(e.wheelDelta) 7. // 火狐，滚轮滚动信息 8. console.log(e.detail) 9. }) |

## 下午复习

滚轮事件

mousewheel事件，火狐：DOMMouseScroll

navigator.userAgent Firefox

在ie低版本中，document MSIE 8

事件对象 e = e || window.event

ff e.detail 2|3

其他 e.wheelDelta 120|150

# 二、键盘事件

当我们敲击键盘的时候，也会触发事件，为了捕获这类交互，提供了键盘事件

键盘事件有三个：

keydown 按下键盘

keyup 键盘弹起

keypress 按下字符（有输入的情况下）（辅助键，如ctrl，shift不能触发）

我们可以为document绑定键盘事件，监听键盘的输入

执行顺序：按键按下，输入字符，按键弹起 => keydown, keypress, keyup

该事件在所有浏览器下表现是一致的。

## 2.1触发键盘事件的元素

有三类：

全局元素绑定，如document

表单元素可以绑定：如input

可以获取焦点的元素（设置tabindex属性的元素）

## 2.2 tabindex

是w3c标准属性，

作用：

让元素可以获取焦点

让元素的焦点可以通过tab键切换

tab键：按顺序触发

shift+tab键：逆向触发

该属性的属性值表示tab键触发焦点的顺序

## 2.3事件对象

在键盘事件中，我们更关心我们按下的键盘，因此提供了两个属性

key 表示按键的名称

keyCode 表示按键的编码

注意：key区分大小写，区分数字和符号，keyCode始终是该键的编码，因此工作中，我们判断编码更精确。

|  |
| --- |
| 1. // 监听键盘事件 2. // 按下按键 3. document.onkeydown = function() { 4. console.log('按键按下了') 5. } 6. // 按键弹起 7. document.onkeyup = function() { 8. console.log('按键弹起来了') 9. } 10. // 有内容输入 11. document.onkeypress = function() { 12. console.log('输入字符了') 13. } |

# 三、面向对象

面向对象编程是与面向过程编程相对的，我们在目前开发中，使用的就是面向过程开发

面向过程开发：通过将数据存储在变量中，通过方法封装功能

面向对象开发：通过将数据存储在对象中，通过属性方法封装功能

面向对象开发是一种开发方式。

就是一种新的书写代码的方式

特点：通过对象存储数据（属性），通过方法实现功能

在方法中，通过this访问数据，无法使用this，要缓存this

|  |
| --- |
| 1. // 函数定义式 2. function Person(name, sex, height) { 3. // 存储这些数据 4. this.name = name; 5. this.sex = sex; 6. this.height = height; 7. } 8. // 函数表达式 9. var Person = function(name, sex, height) { 10. console.log(111, this) 11. // 存储这些数据 12. this.name = name; 13. this.sex = sex; 14. this.height = height; 15. // 第四步是说，返回创建的对象 16. // return this; 17. // 返回值类型的数据 18. // return 100 19. // return 'hello' 20. // return true 21. // return undefined 22. // return null 23. // 引用类型的 24. // return [] 25. // return { color: 'red' } 26. // return function() {} 27. } 28. // 使用构造函数，就是在创建该类型的对象 29. var xiaobai = new Person('小白', '男', 190) 30. console.log(123, xiaobai) |

## 3.1构造函数

我们可以通过执行一个函数，得到一个对象，这是一种简单工厂设计模式。（关于工厂模式后面会讨论）

这种模式创建的对象有个问题，我们无法区分该对象的类型

我们定义的对象，没有类别，为了区分类别，我们就要定义类

通过构造函数定义类的具体实现。

构造函数的定义跟普通的函数的定义没有区别。都是一个函数

但是我们常常将构造函数的首字母大写，人为要求的。

构造函数与普通函数的使用是有区别

普通函数可以直接执行来使用

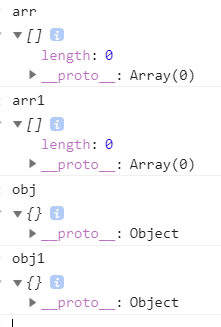
构造函数通过new关键字来使用

构造函数一旦省略new关键字，就当做普通函数处理了

常见的构造函数：Array， Object...

我们在创建数组，创建对象的时候，也是使用new关键字

注意：在工作中，千万不要直接执行构造函数，一定要使用new关键字



## 3.2构造函数执行的四个步骤

构造函数执行有四步

第一步 在函数中，开辟一个空间

第二步 更改this指向（将this指向了我们要创建的对象）

第三步 执行构造函数，为this赋值

第四步 返回创建的实例化对象

所以在构造函数中，return this是没有任何问题的。所以工作中，构造函数的后台已经将实例化对象返回了，所以在构造函数中，返回this就没有必要了，所以工作中，我们也常常省略return this语句

但是如果返回其他的数据，就有问题了

如果返回值类型的数据（例如，数字，字符串，布尔值，null，undefined），没有影响

如果返回引用类型的数据（例如，数字，对象等）,我们将无法得到实例化对象

注意：构造函数跟普通函数是一样的

普通函数可以通过：函数表达式或者函数定义式创建

构造函数也可以通过：函数表达式或者函数定义式创建

定义式，类可以声明前置

表达式，类不可以声明前置

定义类是为了区分对象的，构造函数就是类的实现，定义构造函数就是定义类，执行（创建）构造函数，就是在创建该类型的对象

构造函数执行完毕，创建的对象，就是类的实例化对象。

|  |
| --- |
| 1. // 定义气球类 2. function Ball(src) { 3. // 存储数据 4. this.src = src; 5. // 创建图片 6. this.img = document.createElement('img') 7. // 气球元素 8. this.dom = document.createElement('div') 9. // 元素的宽高 10. this.width = 0; 11. this.height = 0; 12. // 元素的位置 13. this.top = 0; 14. this.left = 0; 15. // 方法定义功能 16. this.init = function() { 17. // 备份this 18. var me = this; 19. // 监听图片加载完成 20. this.img.onload = function() { 21. // 根据图片的宽高，计算元素的宽高 22. me.width = me.img.width / 4; 23. me.height = me.img.height / 3; 24. // 绘制气球 25. me.createBall(); 26. // 上树 27. me.upTree(); 28. // 绘制动画 29. me.animate() 30. console.log('加载完成', me, me.img.width, me.img.height) 31. } 32. // 加载图片 33. this.img.src = this.src 34. } 35. // 定义其他功能 36. // 绘制气球 37. this.createBall = function() { 38. // 设置样式 39. this.dom.style.width = this.width + 'px'; 40. this.dom.style.height = this.height + 'px'; 41. // 定位 42. this.dom.style.position = 'absolute'; 43. // 横纵坐标 44. this.dom.style.top = this.top + 'px'; 45. this.dom.style.left = this.left + 'px'; 46. // 背景 47. this.dom.style.backgroundImage = 'url(' + this.src + ')'; 48. // 位置 49. this.dom.style.backgroundPositionX = -this.width \* parseInt(Math.random() \* 4) + 'px'; 50. this.dom.style.backgroundPositionY = -this.height \* parseInt(Math.random() \* 3) + 'px'; 51. } 52. // 气球元素上树 53. this.upTree = function() { 54. // 上树 55. document.body.appendChild(this.dom) 56. } 57. // 移动气球 58. this.move = function() { 59. // 横向移动 60. this.left += 1; 61. // 更改样式 62. this.dom.style.left = this.left + 'px'; 63. } 64. // 循环移动 65. this.animate = function() { 66. // 缓存this 67. var me = this; 68. // 每30毫秒移动一次 69. setInterval(function() { 70. // console.log(this) 71. // 移动图片 72. me.move(); 73. }, 30) 74. } 75. } 76. // 实例化 77. var b = new Ball('images/balloon.jpg'); 78. // 执行 79. b.init(); |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |