目录

[javascript进阶 1](#_Toc27012)

[复习： 3](#_Toc24436)

[一、 构造函数 4](#_Toc20673)

[1.1 构造函数执行的四步 4](#_Toc7537)

[二、原型 4](#_Toc18379)

[2.1 hasOwnProperty 5](#_Toc234)

[三、安全类 6](#_Toc12469)

[四、继承 7](#_Toc17612)

[4.1 类式继承 7](#_Toc16211)

[4.2 构造函数式继承 7](#_Toc1993)

[4.3 组合式继承 7](#_Toc12533)

[4.4 寄生式继承 7](#_Toc28084)

[4.5 寄生组合式继承 8](#_Toc3808)

[4.6 instanceof 8](#_Toc25746)

[五、 内置构造函数 9](#_Toc14734)

[5.1 内置构造函数的分类 9](#_Toc29524)

[5.2 Function 9](#_Toc11011)

[5.3 RegExp 9](#_Toc26464)

[5.4 String 10](#_Toc4282)

[5.5 Boolean 10](#_Toc8011)

[5.6 Number 11](#_Toc8686)

# 复习：

鼠标滚轮事件：

事件名称： onmousewheel

火狐中不支持onmousewheel事件，支持DOMMouseScroll事件

事件对象及属性：

在非火狐中获取鼠标滚轮方向的属性： e.wheelDelta

如果是向下滚动：-120的倍数

如果是向上滚动：120的倍数

在火狐中获取鼠标滚轮方向的属性：e.detail

如果是向下滚动：3的倍数

如果是向上滚动： -3的倍数

e.key: 获取输入的字符

e.keyCode: 获取到的是编码

常用的编码：

回车 32 空格 13 左箭头37 上箭头38 右箭头39 下箭头 40

键盘事件：

onkeydown: 当键盘按下的时候触发

onkeyup: 当键盘抬起的时候触发

onkeypress: 有字符输入的时候触发

tabIndex属性：

该属性是html的标准属性，页面中的元素有很多，当用户失去鼠标之后， 可以通过按下键盘上的tab键来获取元素的焦点，给元素添加tabIndex属性， 它的值就是决定了按下tab键的时候，获取焦点的顺序

# 构造函数

构造函数与普通函数在定义方式上没有区别，只不过构造函数的首字母要大写

举例：

|  |
| --- |
| 1. // 普通函数 2. function fn() { 3. } 4. // 构造函数 5. function People() { 6. } 7. // 构造函数 8. function Dog() { |

调用方式： 要通过new 关键字

举例：

|  |
| --- |
| 1. var p = new People(); 2. console.log(p); |

结果：

|  |
| --- |
|  |

## 1.1 构造函数执行的四步

|  |
| --- |
| 构造函数执行时候的四个步骤:  -在函数内部，开辟一个新的空间  -改变this指向 (指向创建的对象)  -执行函数中的代码， 为this赋值  -返回创建的实例化对象 |

因此， 我们可以通过this.xx进行赋值

# 二、原型

每一个函数都有一个属性是prototype, 它的值是一个对象

对于普通函数来说， 作用不大

但是对于构造函数来说， 作用就是实例共享方法

|  |
| --- |
| 1. // 定义构造函数 2. function People(name, age, sex) { 3. this.name = name; 4. this.age = age; 5. this.sex = sex; 6. } 7. People.prototype.sayHello = function() { 8. console.log("大家好, 我的名字叫做" + this.name + ",我的年龄是" + this.age + ",我是一个" + this.sex + "孩子"); 9. } 10. People.prototype.sayNihao = function() { 11. console.log("大家好, 我的名字叫做" + this.name); 12. } |

执行代码：

|  |
| --- |
| 1. // 实例化对象 2. var xiaoming = new People("小明", 12, "男"); 3. var xiaohong = new People("小红", 11, "女"); 4. xiaoming.sayHello(); 5. xiaohong.sayHello(); 6. xiaoming.sayNihao(); 7. xiaohong.sayNihao(); |

结果：

|  |
| --- |
|  |

## hasOwnProperty

该方法是用于检测对象中是否含有指定的属性

举例：

|  |
| --- |
| 1. // 定义一个构造函数 2. function People(name, age, sex) { 3. this.name = name; 4. this.age = age; 5. this.sex = sex; 6. } 7. // 方法写在原型中 8. People.prototype.sayHello = function() { 9. console.log("大家好, 我的名字叫做" + this.name + "我的年龄是" + this.age + "我是一个" + this.sex + "孩子"); 10. } 11. People.prototype.sayNihao = function() { 12. console.log("你好"); 13. } 14. // 添加属性 15. People.prototype.title = "nihao"; 16. // 实例化对象 17. var xiaoming = new People("小明", 12, "男"); |

执行代码：

|  |
| --- |
| 1. // hasOwnProperty 2. // 该方法是用于检测对象中是否含有指定的属性 3. console.log(xiaoming.hasOwnProperty("title")); |
|  |

结果：

|  |
| --- |
|  |

# 三、安全类

无论外部如何调用类， 都会返回该类的实例化对象

举例：

|  |
| --- |
| 1. function People(name, age, sex) { 2. // 判断this指向，从而决定代码如何执行 3. if (this === window) { 4. // 说明没有使用new 来调用， 是当做普通函数调用， 而普通函数想要返回内容的话需要使用return 5. return new People(name, age, sex); 6. } else { 7. // 说明使用new关键字, 一切正常 8. this.name = name; 9. this.age = age; 10. this.sex = sex; 11. } 12. } |

执行代码：

|  |
| --- |
| 1. // 测试 2. var p = new People("小明", 12, "男"); 3. // 不使用new 来调用 4. var p1 = People("小红", 12, "女"); |

结果：

|  |
| --- |
|  |

# 四、继承

继承： 指的是子类继承父类的属性和方法。

继承方式： 1 类式继承 2 构造函数式继承 3 组合式继承 4 寄生式继承 5 寄生式组合继承

## 4.1 类式继承

|  |
| --- |
| 1. // 定义父类 2. function People() { 3. } 4. People.prototype.sayHello = function(){} 5. // 定义子类 6. function Student() { 7. } 8. // 继承 9. Student.prototype = new People(); 10. // 注意补回constructor属性 11. Student.prototype.constructor = Student; |

## 4.2 构造函数式继承

注：其实跟继承没关系

|  |
| --- |
| 1. // 定义父类 2. function People(name, age, sex) { 3. this.name = name; 4. this.age = age; 5. This.sex = sex; 6. } 7. People.prototype.sayHello = function(){} 8. // 定义子类 9. function Student(name, age, sex, grade) { 10. People.apply(this, arguments); 11. // 新属性的继承代码一定要放在下方 12. this.grade = grade; 13. } |

## 4.3 组合式继承

3.1 + 3.2

## 4.4 寄生式继承

|  |
| --- |
| 1. function jisheng(father, child) { 2. // 由于使用类式继承， 子类的原型就是父类的实例了，那么会在子类的原型中多出很多个无用的属性 3. // 定义一个构造函数 4. var F = function() { 5. } 6. // 将该构造函数的原型指向父类的原型 7. F.prototype = father.prototype; 8. // 将该构造函数的实例交给子类的原型 9. child.prototype = new F(); 10. // 手工补回子类的constructor属性 11. child.prototype.constructor = child; 12. // 返回child 13. return child; 14. } |

## 4.5 寄生组合式继承

|  |
| --- |
| 1. // 寄生组合式继承: 在定义类的时候， 使用构造函数式继承， 这样就叫做寄生组合式继承 2. function People(name, age, sex) { 3. this.name = name; 4. this.age = age; 5. this.sex = sex; 6. } 7. // 将方法写在原型中 8. People.prototype.intro = function() { 9. console.log("你好"); 10. } 11. // 定义子类 12. function Student(name, age, sex, grade) { 13. People.apply(this, arguments); 14. this.grade = grade; 15. } 16. // 寄生式继承： 定义一个寄生函数，接收两个参数， 第一个参数是父类， 第二个参数是子类 17. function jisheng(father, child) { 18. // 由于使用类式继承， 子类的原型就是父类的实例了，那么会在子类的原型中多出很多个无用的属性 19. // 定义一个构造函数 20. var F = function() { 21. } 22. // 将该构造函数的原型指向父类的原型 23. F.prototype = father.prototype; 24. // 将该构造函数的实例交给子类的原型 25. child.prototype = new F(); 26. // 手工补回子类的constructor属性 27. child.prototype.constructor = child; 28. // 返回child 29. return child; 30. } |

## 4.6 instanceof

该关键字用于判定某一个对象是否是某一个构造函数的实例

使用方式：

|  |
| --- |
| 1. 对象 instanceof 构造函数 |

# 内置构造函数

## 5.1 内置构造函数的分类

ECMAScript核心语法支持的内置构造函数:

Object、Array、Function、String、Number、Boolean、Date、RegExp、 Error

## 5.2 Function

使用方式：

除了最后一个参数是函数体，其它的都是形参

举例：

|  |
| --- |
| 1. var fun = new Function("a", "b", "return a + b"); 2. var result = fun(1, 2); |

结果：

|  |
| --- |
|  |
|  |

等价方式1：

|  |
| --- |
| 1. var fun = function(a, b) { 2. return a + b; 3. } |

等价方式2：

|  |
| --- |
| 1. function fun(a, b) { 2. return a + b; 3. } |

特点: 当创建函数的时候使用的是new Function得到的函数名称是anonymous

而通过函数表达式或者是函数声明式通过函数.name得到的是函数名

额外：

|  |
| --- |
| 1. 函数有一个length属性，是形参的个数 2. 函数中有一个arguments.length, 函数执行的时候实参的个数 |

## 5.3 RegExp

该构造函数用于定义正则表达式

使用方式：

接收两个参数

第一个参数：正则表达式的表达体

第二个参数：正则表达式的修饰符 i, g, m

举例：

|  |
| --- |
| 1. // RegExp 2. // 该构造函数用于定义正则表达式 3. var reg = /^\s+$/g; 4. var reg1 = new RegExp("^\\s+$", "g"); |

结果：

|  |
| --- |
|  |

注意: 由于字符串中也有转义字符，所以要多转义一次

## 5.4 String

这是一个构造函数，是string值类型的对应包装类型

举例：

|  |
| --- |
| 1. var str = "abcdefg"; 2. var str\_obj = new String(str); |

结果：

|  |
| --- |
|  |

## 5.5 Boolean

这是一个构造函数，是bool值类型的对应包装类型

举例：

|  |
| --- |
| 1. var f = false; 2. var f\_obj = new Boolean(f); |

结果：

|  |
| --- |
|  |

## 5.6 Number

这是一个构造函数， 是number值类型的对应包装类型

举例：

|  |
| --- |
| 1. // Number 2. var num = "3a"; 3. // var num\_obj = new Number(num); 4. // Number本身不是一个安全类，当Number自执行的时候，是作为类型转换函数来使用了 5. var num1 = Number(num); |

结果：

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |