## JS引用

1. Script标签一旦用于引入外部文件，就不能再编写代码了。需再创建一个新的script标签用于编写。
2. 可放在a标签和事件中



## 变量

1. 在js中，创建一个变量，需要var 声明

var a;

1. 为变量赋值

a=123;

或 var b=456;

## 标识符

在js中所有由我们自主命名的都可以称为标识符

例如：变量名，函数名，属性名

命名规则：

1. 标识符中可以含有字母、数字、\_ 、$
2. 标识符不能以数字开头
3. 标识符不能使用关键字和保留字（如：if、var）
4. 标识符一般采用驼峰命名法

var a\_1\_$;

## string

1. 使用字符串时引号不能嵌套：一对单引号中不能出现单引号，双引号同理。若需使用，则要用\转义字符。如： str=”我说:\”今天天气真好\””;

\n表示换行

\t 制表符（tab）

\\ 表示 \

1. 可以使用typeof来检查一个变量的类型

语法：typeof 变量

1. JS中可以表示的数字最大值：Number.MAX\_VALUE。如果使用number表示的数字超过了最大值，则会返回Infinity,表示正无穷，Infinity也属number。
2. 若使用js进行浮点数运算，可能得到一个不精确的结果。如：c = 0.1+0.2
3. 强制类型转换：
4. a = string(a);
5. a = a.toString(); //在转换null和 undefined会报错

## number

1. 使用Number（）函数
2. 如果是纯数字的字符串，则直接转换为数字
3. 如果字符串中有非数字的内容，则转换为NaN
4. 如果字符串是一个空串或者是一个全是空格的字符串，则转换为0
5. 使用parseInt()、parsefloat()

## 其它进制的数字

1. 在js中，如果需要表示16进制的数字，则需要以0x开头(并不是所有浏览器都支持)

1. 如果需要表示8进制的数字，则需要以0开头



1. 如果需要表示2进制的数字，则需要以0b开头



## 转换为boolean

1. 数字：除了0和NaN，都是true



1. 字符串：除了空串，都是true
2. null 和undefined都是false

## 运算符

1. 任何值和NaN做运算都得NaN
2. 任何值（包括NaN）和字符串（包括空串）相加都会被转换为字符串



1. 任何值做- \* /运算时都会自动转换为Number
2. 快速转换为number



## 自增和自减

1. a++和++a都会立即使原变量的值加一。但a++的值等于原变量的值（即等于a），++a等于自增后的值（即a+1）
2. 同上，a--的值等于原变量的值（即等于a），--a等于自增后的值（即a-1）

## 逻辑运算符

1. && （与）

找false，从左至右运行，当存在false，后面不再运行



1. || （或）

找true，从左至右运行，当存在true，后面不再运行



1. true && true ，当两个值都为true，则返回后边的

false && false ，当两个值都为false，则返回前面的

（若第一个值为true，则必然返回第二个值；若第一个值为false，则必然返回第一个值）





1. true || true ，当两个值都为true，则直接返回前面的

false || false ，当两个值都为false，则直接返回后面的

（若第一个值为true，则直接返回第一个值；若第一个值为false，则必然返回第一个值）

## 关系运算符

1. 若两个字符串相比较，则只会取第一位进行字符编码比较





## Unicode编码

1. 在字符串中使用转义字符\u 可以输出Unicode编码



1. 在HTML中，使用&#可输出Unicode编码

（Unicode编码默认为16进制，在html中使用需先转换为10进制）



## 相等运算符

1. 当使用==来比较两个值时，如果两个值的类型不同，则会自动进行类型转换，将其转换为相同类型，然后再比较
2. NaN不和任何值相等
3. != 判断两个值是否不相等
4. === 判断两个值是否全等



1. !== 判断两个值是否不全等



## 条件运算符（三元运算符）

1. 语法：

条件表达式?语句1：语句2;

1. 执行流程：

首先对条件表达式进行求值

如果该值为true，则执行语句1，并返回执行结果

如果该值为flase，则执行语句2，并返回执行结果



1. 如果条件表达式为非布尔值，则会先将其转换为布尔值

## prompt（）

该函数可以弹出一个提示框，并带有一个文本框，可输入一段内容

## 条件表达式

1. 语法：

switch(条件表达式){

case 表达式1：

语句1 。。。

break;

case 表达式2：

语句2 。。。

break;

case 表达式3：

语句3 。。。

break;

default:

语句。。。

break;

}



## for循环



## break和continue

1. break可结束离得最近的整个循环
2. continue只会结束当次循环



当 i=3 时跳过

1. 可以为循环语句创建一个label，来标识循环语句，break后面跟label可结束指定循环



## console.time(“str”)

可开启一个计时器，console.timeEnd("str");用来结束一个计时器



## 对象的分类

1. 内建对象

由ES标准中定义的对象：Math String Number Boolean Function Object…

1. 宿主对象

由JS运行环境提供的对象，主要由浏览器提供：Bom Dom

1. 自定义对象

由开发人员自己创建的对象



## 对象的属性名和属性值

1. 如果要使用特殊的属性名，则需要使用特殊方式：对象[“属性名”] = 属性值



1. 变量



1. 属性值可以是任意数据类型，以及可以是一个对象
2. in可以检查对象中是否含有某个属性



## 基本数据类型和引用数据类型

1. 基本数据类型的值直接在栈内存中存储，值与值之间相互独立



1. 引用数据类型（对象）是保存到堆内存中，变量保存的是对象的内存地址



1. b修改了对象名，两者断开联系



1. 不同对象，即使属性值相同，也不全等（因内存地址不同）



## 对象字面量

1. 可使用{}来直接创建对象



1. 可直接指定对象中的属性（如果使用特殊名，必须加引号）



## 函数

1. 函数生成与调用



1. 可以在（）中指定多个形参



1. 实参可以是任意的数据类型，也可以是一个对象

当参数过多时，可以将参数封装到一个对象中，然后通过对象传递



1. 实参也可以是一个函数



## 定义一个函数，可以根据半径计算一个圆的面积，并返回计算结果



## 立即执行函数

1. 无需调用，立即执行，但也无法再次调用



## 枚举对象中的属性

* + 1. 语法：

for (var 变量 in 对象){

}



## 作用域

1. 全局作用域

- 直接编写在script中的JS代码，都在全局作用域

- 全局作用域中有一个全局对象window，它代表的是浏览器的一个窗口

- 在全局作用域中创建的变量都会作为window对象的属性保存

1. 函数作用域

- 调用函数时创建函数作用域，函数执行完毕后，作用域销毁

- 每调用一次就会创建一个新的作用域，他们之间相互独立

- 在函数作用域中，如若像访问与函数作用域中相同名字的全局作用域变量，需调用window.

## 变量的声明提前

* + 1. 使用var声明的变量，会在所有的代码执行之前被声明（但是不会赋值）



## 方法

函数也可以称为对象的属性，如果一个函数作为对象的属性保存，那么称这个函数是这个对象的方法，调用函数就说调用对象的方法

## this

解析器每次在调用函数时都会向函数内部传递一个隐含参数

这个隐含参数就是this，this指向的是一个对象

这个对象外面称为函数执行的上下文对象

根据函数的调用方式的不同，this也会指向不同的对象

## 使用工厂方法创建对象

使用该方法可以大批量创建对象



## 构造函数

1. 构造函数就是一个普通的函数，创建方式和普通函数相同，不同的是构造函数习惯上首字母大写
2. 构造函数的执行流程：
3. 立即创建一个新的对象
4. 将创建的对象设置为函数中的this，在构造函数中可以使用this来引用新建的对象
5. 逐行执行函数中的代码
6. 将新建的对象作为返回值返回



1. 构造函数每执行一次，就会创建一个新的sayName方法，占用空间



1. 使用instanceof 可以检查一个实例是否属于一个类



1. this的情况
   1. 当以函数的形式调用时，this是window
   2. 当以方法的形式调用时，谁调用方法this就是谁
   3. 当以构造函数调用时，this就是新建的那个对象

## 原型prototype

1. 我们所创建的每一个函数，解析器都会向函数中添加一个属性prototype,这个属性对应着一个对象，这个对象就是原型对象
2. 当函数以构造函数的形式调用，它所创建的对象中都会有一个隐含的属性，指向该构造函数的原型对象，我们可以通过\_\_proto\_\_来访问该属性
3. **原型对象就像一个公共的区域，所有同一个类的实例都有可以访问到这个原型对象，我们可以将对象中共有的内容，统一设置到原型对象中！！！**



直接添加方法：



原型对象：



1. 当我们访问对象的一个属性或方法时，它会在对象自身中寻找，如果有则直接使用，如果没有则会去原型对象中寻找。
2. 原型对象中的方法共用一个区域

## 创建数组

* + 1. 使用构造函数创建数组时，可以同时添加元素，将要添加的元素作为构造函数的参数传递
    2. 创建指定长度的数组
    3. 数组中的元素可以是任意的数据类型，包括函数、数组（二维数组）



## length

* + 1. 对于连续的数组，使用length可以获取到数组的长度

对于非连续的数组，可获取数组到的长度为最大索引+1



* + 1. 如果使用length修改长度，若修改后的值大于原长度，则多出部分会被空出，

若修改后的值小于原长度，则多出的元素会被删除



* + 1. 向数组的最后一个位置添加元素

语法：数组[数组.length] = 值



## push（）和 unshift（）

1. push（）方法可以向数组的末尾添加一个或多个元素，并返回数组的新长度，可以将要添加的元素作为方法传递



1. unshift（）方法可以向数组的开头添加一个或多个元素，并返回数组的新长度，插入元素后，其它元素的索引会依次调整



## pop（） 和 shift（）

1. pop（）方法可以删除数组的最后一个元素,并返回该值



1. shift（）方法可以删除数组的第一个元素，并返回该值



## 遍历数组



遍历数组后，将指定条件的元素筛选出来





## forEach（）

1. forEach()方法需要一个函数作为参数
2. 像这种由我们创建函数但不由我们调用的，称为回调函数
3. 数组中有几个元素，函数就会执行几次
4. 浏览器会在回调函数中传递三个参数，分别是value(元素)，index(索引)，obj(数组)

## slice（）

* + 1. 可以用来从数组中提取指定元素

参数：

* + - 1. 截取开始的位置的索引，包含开始索引
      2. 截取结束的位置的索引，不包含结束索引
    1. 该方法不会改变数组元素，而是将截取到的元素封装到一个新数组中返回



* + 1. 若第二个参数省略，则会截取从开始索引往后的所有元素





* + 1. 若第二个参数为负数（最后一个元素索引为-1）



## splice（）

* + 1. 可以用于删除数组中的指定元素

参数：

1. 第一个表示开始位置的索引
2. 第二个表示删除的数量
3. 第三个及以后，表示所添加的元素
   * 1. 会影响原数组，会将指定元素从原数组中删除，并将被删除的元素的值返回



* + 1. 练习：删除数组中重复的数字



## concat（）

1. concat()可以连接两个或者多个数组，并将新的数组返回，该方法不会对原数组产生影响



## join（）

* + 1. 该方法可以将数组转换为一个字符串，但不会对原数组产生影响，而是将转换后的字符串作为结果返回
    2. 在join（）中可以指定一个字符串作为参数，这个字符串将会成为数组中元素的连接符

## reverse（）

* + 1. 该方法用来反转数组（所有元素前后反转）
    2. 该方法会直接修改原数组



## sort（）

* + 1. 可以用来对数组中的元素进行排序
    2. 该方法会影响原数组，默认会按照Unicode编码进行排序
    3. 可以在方法中添加一个回调函数，来指定排序规则

1. 回调函数中需定义两个形参，浏览器将会分别使用数组中的元素作为为实参去调用回调函数
2. 使用哪个元素不确定，但肯定的是在数组中a一定在b的前面
3. 浏览器会根据回调函数的返回值来决定元素的顺序

如果返回值大于0，则元素交换位置

如果返回值小于0，则不会交换

如果返回值等于0，则不会交换

1. 如果需要升序排列，则返回a-b；若降序，则b-a



## call（）和apply（）

1. 这两个方法都是函数对象的方法，需要通过函数对象来调用
2. 当对函数调用call（）和apply（）都会调用函数执行
3. 在调用时可以将一个对象指定为一个参数





1. call（）方法可以将实参在对象之后依次传递



1. apply（）方法需要将实参封装到一个数组中



## arguments

在调用函数时，浏览器每次回传递两个隐含的参数：①函数的上下文对象this；

②封装实参的对象arguments

1. arguments是一个类数组对象，它也可以通过索引来操作数据，也可以获取长度
2. 在调用函数时，所传递的实参都会在arguments中保存
3. arguments.length可以用来获取实参的长度



1. 即使不定义实参，也可以通过arguments来使用形参



1. 里边也一个属性callee，对应着一个对象，就是当前正在执行的函数对象



## Date

1. 在JS中使用Date对象来表示一个时间
2. 若直接使用构造函数创建一个Date对象，则会封装当前代码执行的时间
3. 创建一个指定的时间对象，需要传递一个表示时间的字符串作为参数，格式：月份/日/年/ 时:分:秒



1. getDate()可以获取当前日期是几日



1. getDay()可以获取当前日期是周几（周日为0）



1. getMonth()可以获取当前月份（0表示1月，11表示12月）



1. getFullYeah()可以获取当前年份



1. getHours()、getMinutes()、getSeconds()获取时、分、秒
2. getTIme()、Date.now()可以获取当前日期的时间戳，可测试代码执行时间

## Math

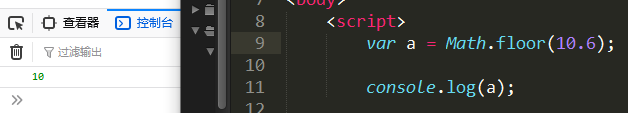
1. Math和其他对象不同，不是一个构造函数，属于工具类，不需要创建对象，里边封装了数学运算相关的属性和方法



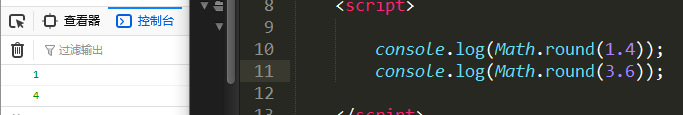
1. Math.ceil（）可以对一个数向上取整



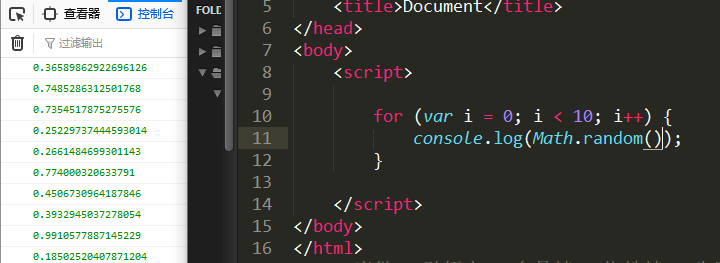
1. Math.floor（）可以对一个数向下取整



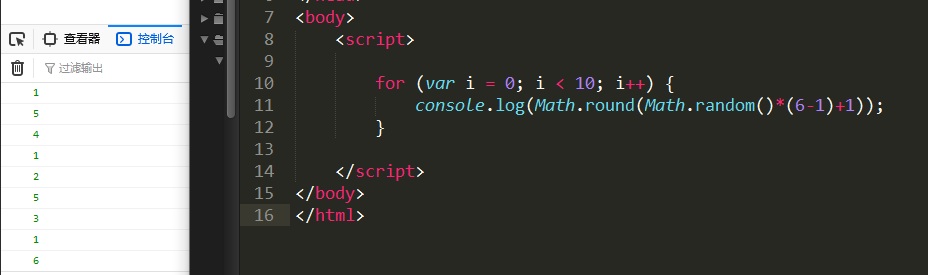
1. Math.round（）可以对一个数四舍五入取整



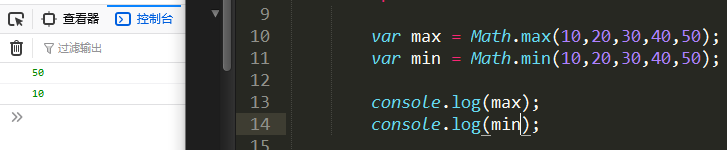
1. Math.random（）可以用来生成一个0-1之间的随机数



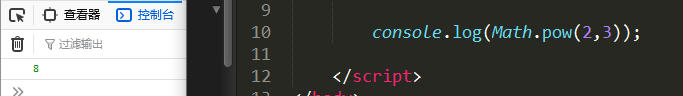
若要生成x~y之间随机数，Math.round(Math.random()\*(y-x)+x)



1. Math.max（）可以获取多个数中的最大值，Math.min（）可以获取多个数中的最小值



1. Math.pow（）可以返回x的y次幂



## 包装类

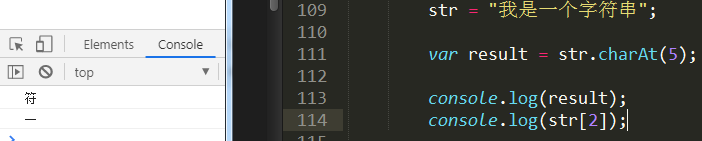
JS提供了三个包装类，通过这三个包装类可以将基本数据类型转换为对象：

①String（）；②Number（）；③Boolean（）

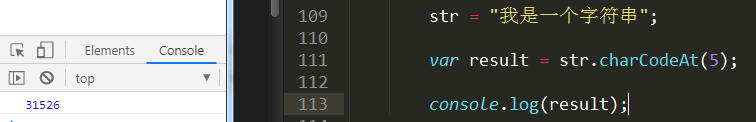
方法和属性只能添加给对象，不能添加给基本数据类型，当我们对一些基本数据类型的值去调用属性和方法时，浏览器会临时使用包装类将其转换为对象，然后再调用对象的属性和方法，调用完之后再将其转换为基本数据类型

## 字符串的方法

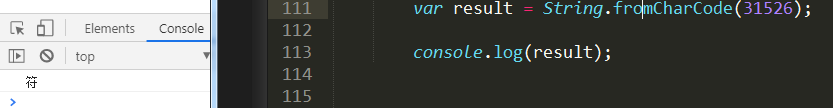
1. charAt（）根据索引获取指定的字符



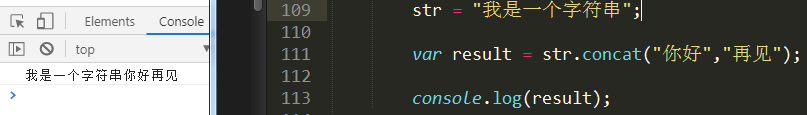
1. charCodeAt（）获取指定位置的字符串编码（Unicode编码）



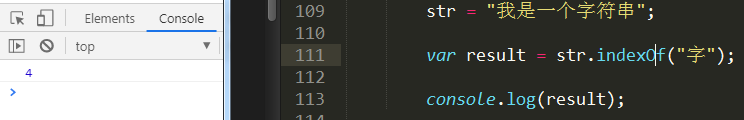
1. String.fromCharCode（）可以根据字符编码去提取字符

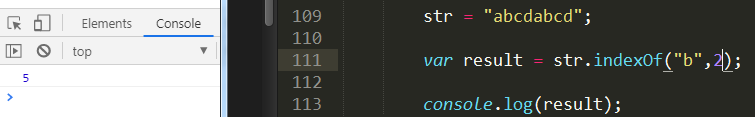


1. concat（）可以用来连接两个或多个字符串

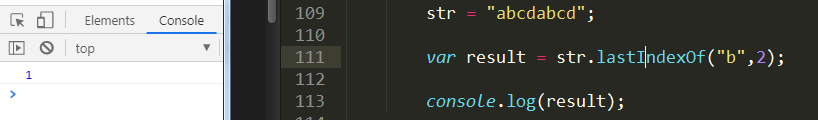


1. indexOf（）可以检索一个字符串中是否含有指定内容，若含有指定内容，则会返回其第一次出现的索引，若没有，则返回-1；可以直接开始查找的位置





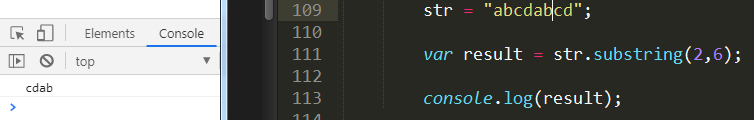
1. lastIndexOf（）用法与indexOf（）相同，不同的是lastIndexOf（）是从后往前找



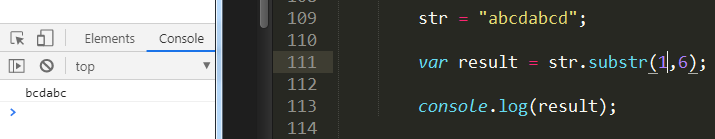
1. substring（）可以用来截取字符串，和slice（）类似，不同的是不能接受负值作为参数，若传递了一个负值，则默认使用0。

第一个参数表示开始截取位置的索引，第二个参数表示结束位置的索引（不包 括结束位置）

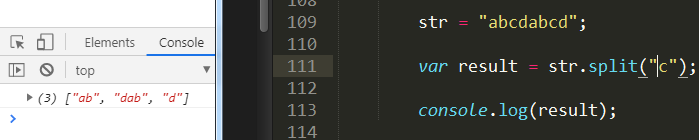
substring（）还会自动调整参数的位置，如果第二个参数小于第一个，则自动 交换



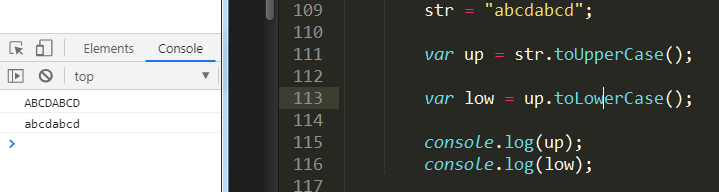
1. substr（）用来截取字符串，第一个参数表示截取开始的位置，第二个参数表示截取的长度



1. split（）可以将一个字符串拆分为一个数组。需要一个字符作为参数，以它作为分隔符拆分

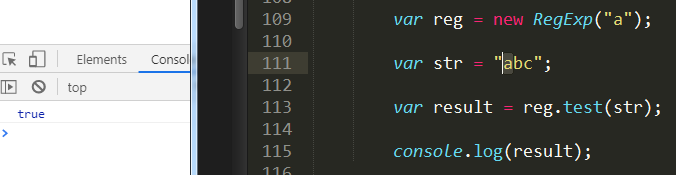


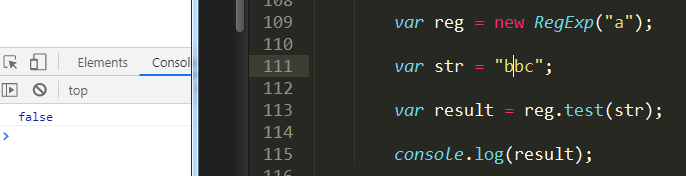
1. toUpperCase（）将字符串转换为大写；toLowerCase（）将字符串转换为小写



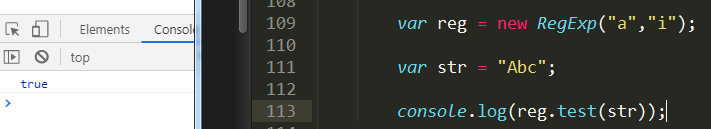
## 正则表达式简介

1. 用于定义一些字符串的规则
2. test（）使用这个方法可以用来检查一个字符串是否符合正则表达式的规则



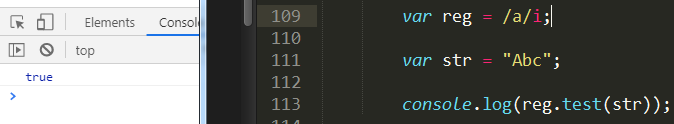


1. 在构造函数中可以传递一个匹配模式作为第二个参数：i（忽略大小写）；g（全局匹配模式）

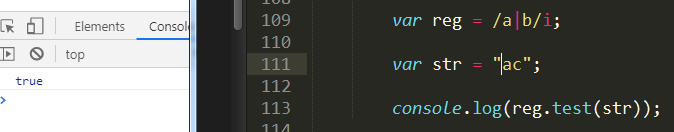


## 使用字面量创建正则表达式

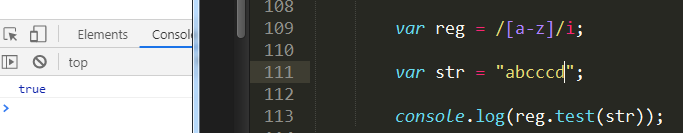
1. 语法：var 变量 = /正则表达式/匹配模式



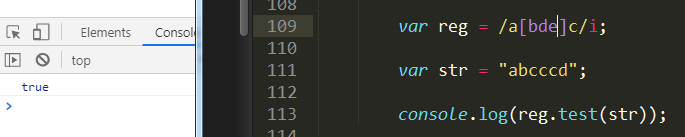
1. 检查字符串中是否含有a或b，使用|表示或者的意思



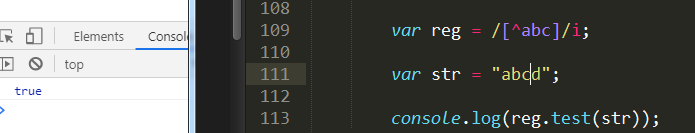
1. 检查一个字符串是否有字母



1. 检查一个字符串是否含有abc，adc，aec

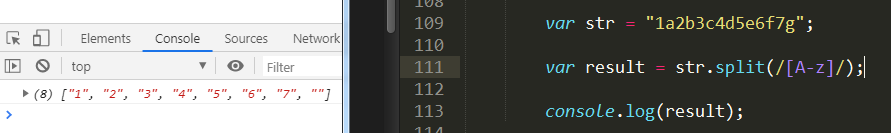


1. [^ ]除了

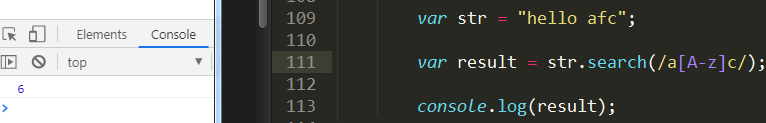


## 字符串和正则表达式的相关方法

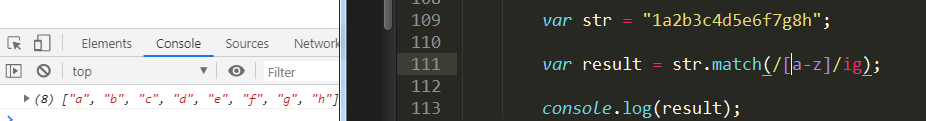
1. 在split（）方法中传递一个正则表达式作为参数，这样方法将会根据正则表达式去拆分字符串



1. search（）方法搜索字符串中是否含有指定内容



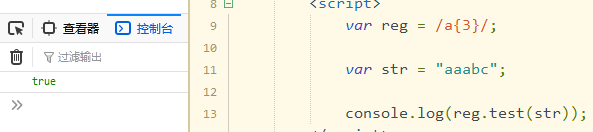
1. match（）从一个字符串中将符合条件的内容提取出来，默认情况下只会找到第一个符合要求的内容，若设置正则表达式为全局匹配模式，可匹配到所有的内容



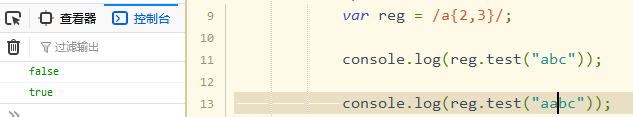
## 正则表达式

1. **通过量词可以设置一个内容出现的次数，只对前面的一个字符有效**

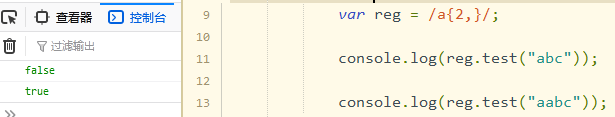
**{n}正好出现n此**

****

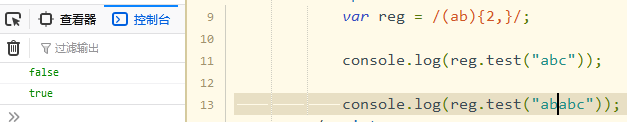
**{m，n}出现m-n次**

****

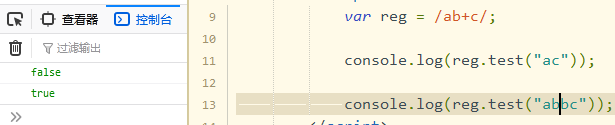
**{m，}m次以上**

****

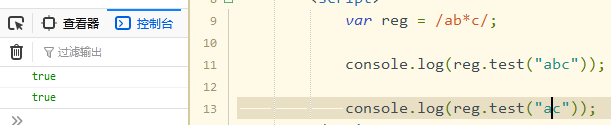
**若要对多个内容设置**

****

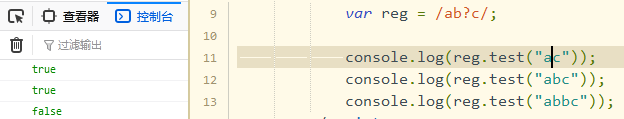
**m+ 至少一个，相当于{1，+}**

****

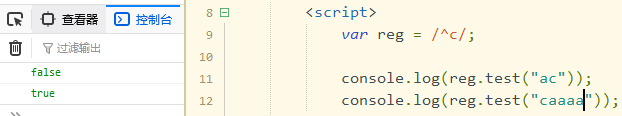
**m\* 0个或多个，相当于{0，}**

****

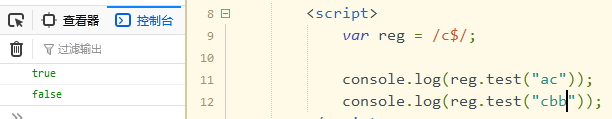
**m？ （0个或1个，相当于{0,1}）**

****

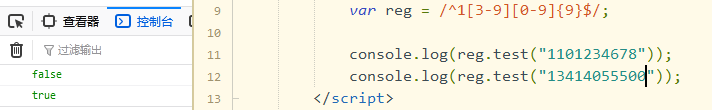
**^c （检查一个字符串是否以c开头）**



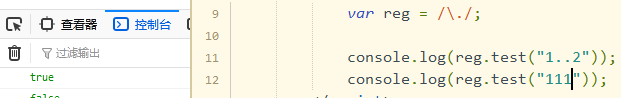
**c$ （检查是否以c结尾）**



**检查一个字符串是否是一个合法手机号**

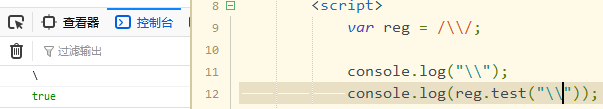


. 表示任意字符，\.可检查字符串中是否含有.

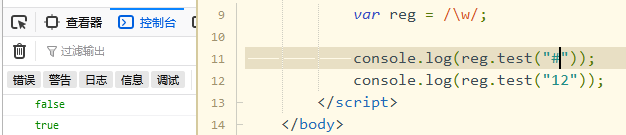


\\ 表示\

注意，若使用构造函数，由于参数是一个字符串，而\是字符串中转义字符， 如果要检查\，则需使用\\代替

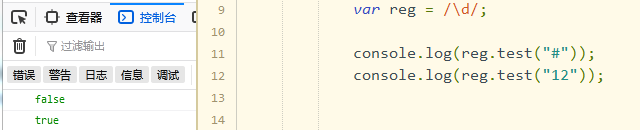


\w 查找单词字符

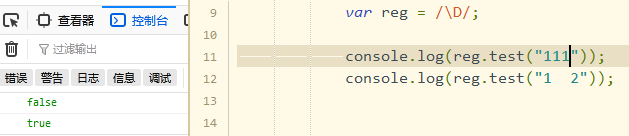


\W 查找非单词字符

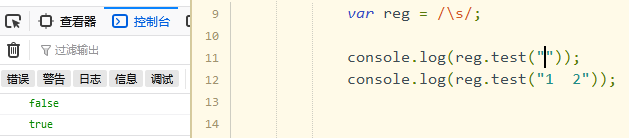
\d 查找数字



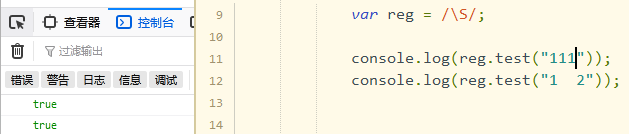
\D 查找非数字字符



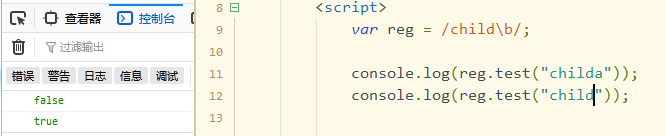
\s 查找空白字符



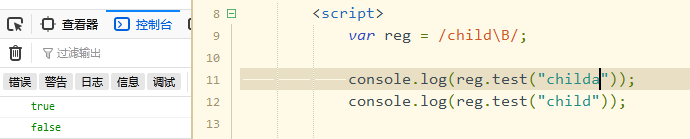
\S 查找非空白字符



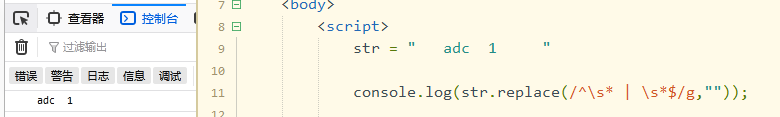
\b 匹配单词边界



\B 匹配非单词边界



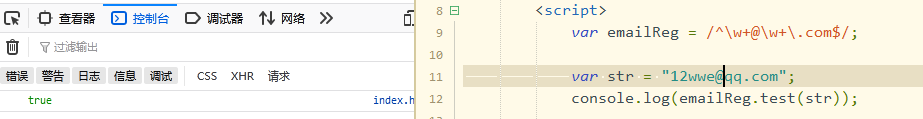
去掉字符串中前后的空格



## 邮件的正则

规则：

任意字母数字下划线 @ 任意字母数字 .com



## DOM

1. 浏览器已为我们提供文档节点对象，代表整个网页

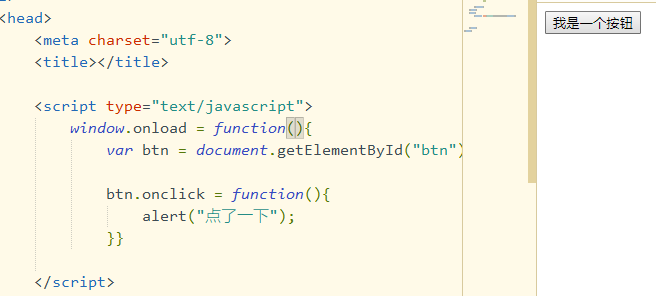


1. 事件，就是用户和浏览器之间的交互

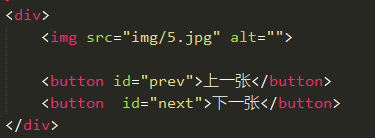
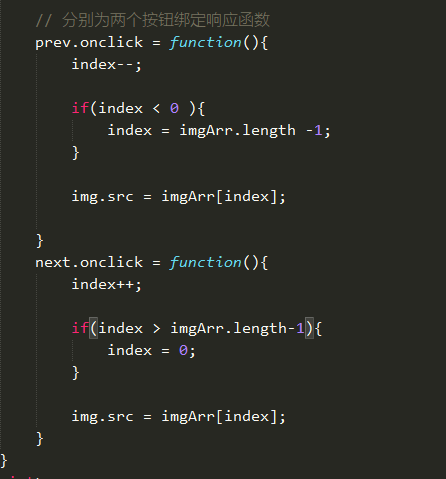
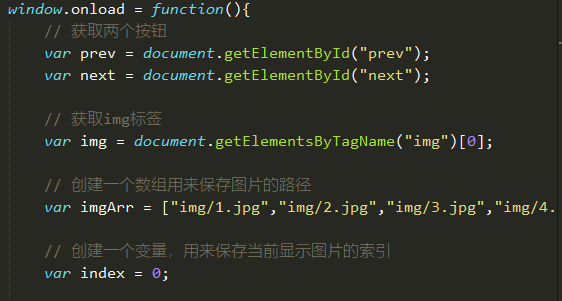
onclick、ondbclick



1. onload（）事件会在整个页面加载完成后才触发，可确保JS代码执行时，所有DOM对象已加载完毕

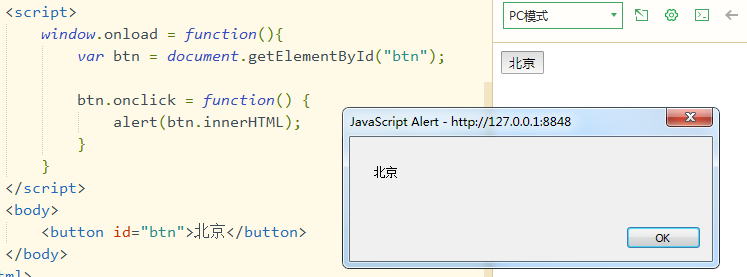


1. 切换图片练习



## dom查询

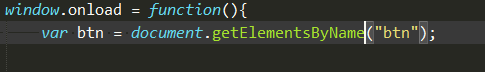
1. 通过id查询



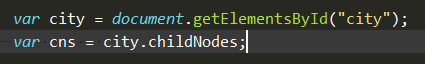
1. 通过标签名查询



1. 通过name获取



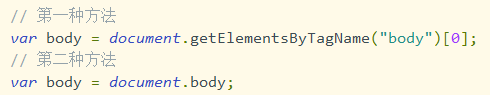
1. childNodes表示当前节点下的所有子节点



1. firstChild表示当前节点的第一个子节点
2. lastChild表示当前节点的最后一个子节点
3. parentNode表示当前节点的父节点
4. previousSibling表示当前节点的前一个兄弟节点
5. nextSibling当前节点的后一个兄弟节点
6. innerText该属性可以获取元素内部的文本内容



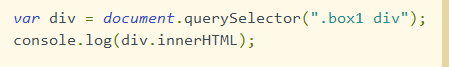
1. 获取body标签



1. 获取页面中所有元素



1. getElementsByClassName（）可以根据class属性值获取一组元素节点对象，但不支持IE8及以下浏览器
2. document.querySelector（）可以根据一个css选择器来查询元素节点对象，只会返回一个



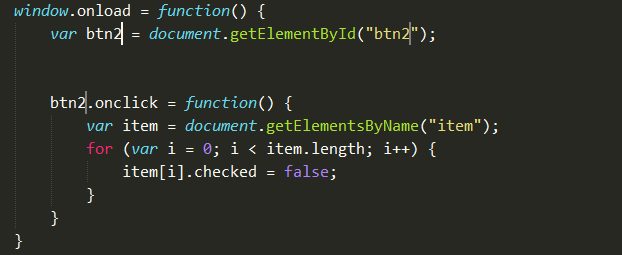
1. document.querySelectorAll（）,会将所有符合条件的元素封装到数组中

## 全选练习

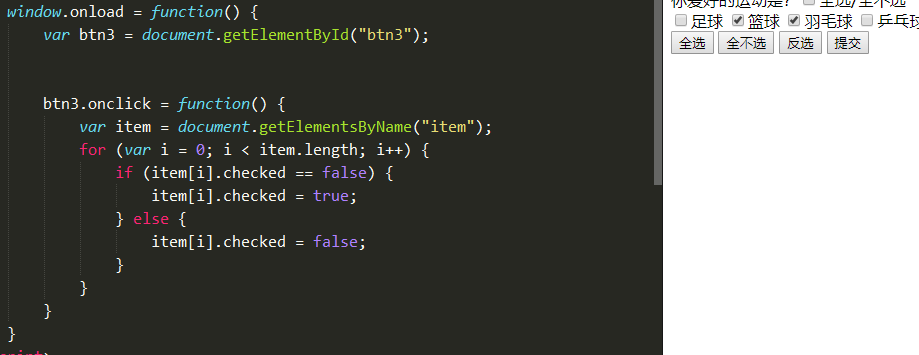
1. 设置按钮全选

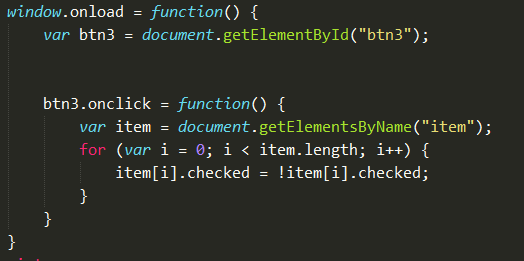


1. 全不选

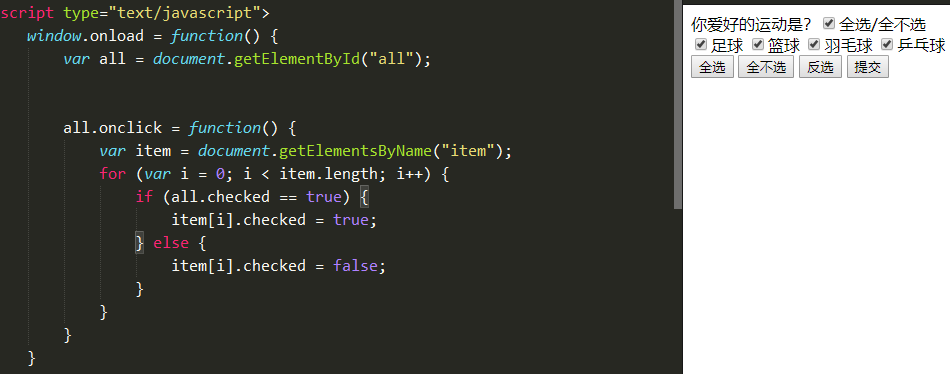


1. 反选

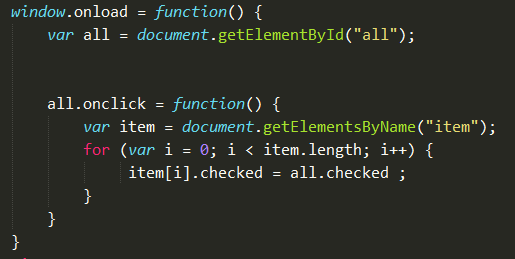
简化



1. 全选/全不选



简化



1. 提交



## dom增删改

1. document.createElement()可以用于创建一个元素节点对象，需要一个标签名作为参数



1. document.createTextNode()可以用于创建文本节点对象



1. appendChild()可以向一个元素中加入子节点



1. insertBefore()可以向一个在某个子节点前加入一个新的子节点



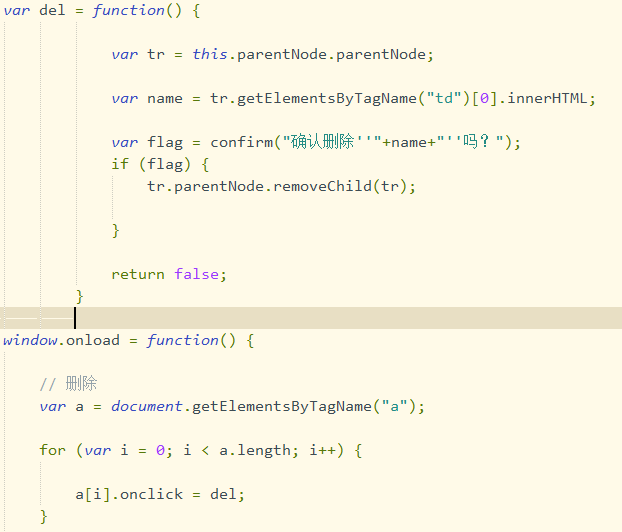
1. replaceChild()可以替换某个子节点



1. removeChild()可以删除一个子节点



1. 练习：删除信息

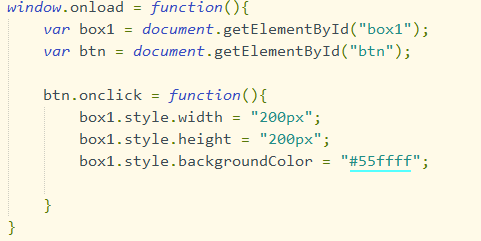


1. 练习：添加信息

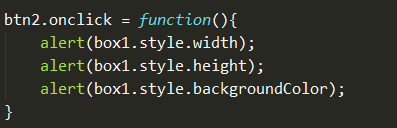


## DOM操作CSS

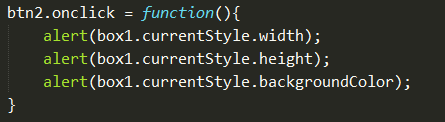
1. 修改样式



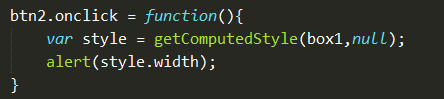
1. 获取样式（内联样式）



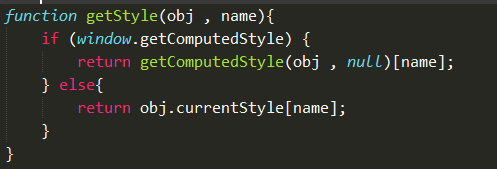
1. 获取正在显示的样式（currentStyle，只支持IE）

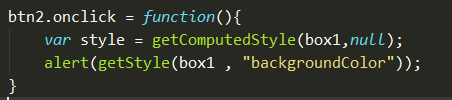


1. getComputedStyle（元素，null），支持IE8以上



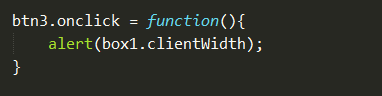
1. 创建函数，兼容浏览器获取样式



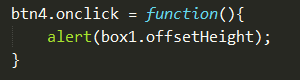


## 其他样式操作的属性

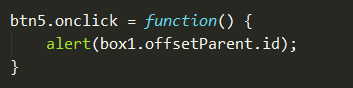
1. clientWidth、clientHeight可以获取元素的可见高度，包括内容区和内边距



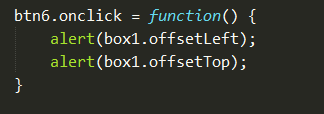
1. offsetWidth、offsetHeight可以获取元素的整个高度和宽度，包括内容区、内边距和边框



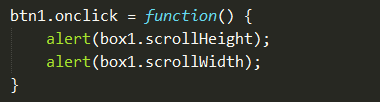
1. offserParent可以用来获取当前元素最近的开启了定位的祖先元素，如果所有祖先元素都没有开启定位，则返回body



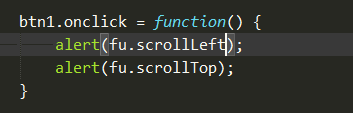
1. offsetLeft、offsetTop表示当前元素相对于其定位父元素的水平偏移量和垂直偏移量



1. scrollWidth、scrollHeight可以获取元素整个滚动区的高度和宽度



1. scrollLeft、scrollWidth可以获取水平、垂直滚动条滚动的距离

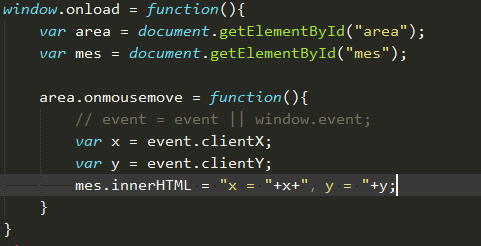


1. 判断是否已经滚动至底部

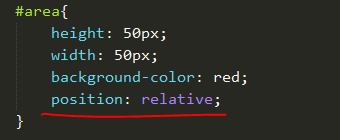
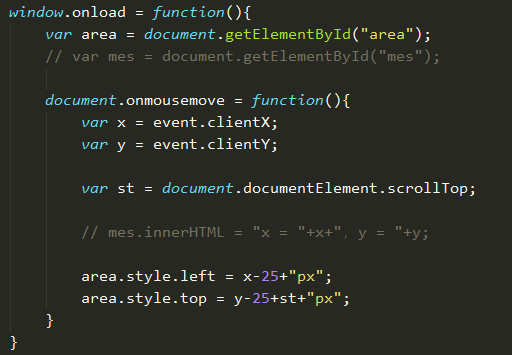


## 事件对象

1. 获取鼠标位置



1. div跟随鼠标移动



## 事件的冒泡

1. 所谓的冒泡就是事件的向上传导，当后代元素的事件被触发时，其祖先元素的相同事件也会被触发
2. 若不希望发生事件冒泡可以通过事件对象来取消
3. 将事件对象的cacelBubble设置为true，即可取消冒泡

## 事件的委派

1. 事件委派是利用了冒泡，通过委派可以减少事件绑定的次数，提高程序的性能

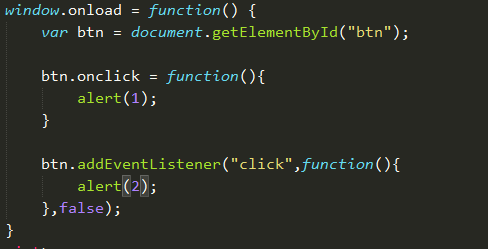


## 事件的绑定

1. 一个元素的一个事件只能绑定一个响应函数，若绑定多个，前面的会被覆盖
2. addEventListener()方法可以为一个元素的相同事件同时绑定多个响应函数

参数：

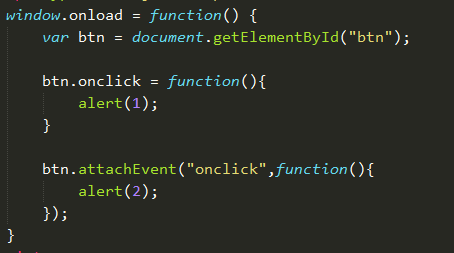
1. 事件的字符串，不要on
2. 回调函数，当事件触发时该函数会被调用
3. 是否在捕获阶段触发，一般传false



1. attachEvent()也可同时绑定多个函数，不同的是后绑定先执行，适用于IE8

参数：

1. 事件的字符串，要on
2. 回调函数



1. 定义一个函数，用来为指定元素绑定响应函数

参数：

obj 要绑定事件的对象

eventStr 事件的字符串（不要on）

callback 回调函数



## 拖拽练习

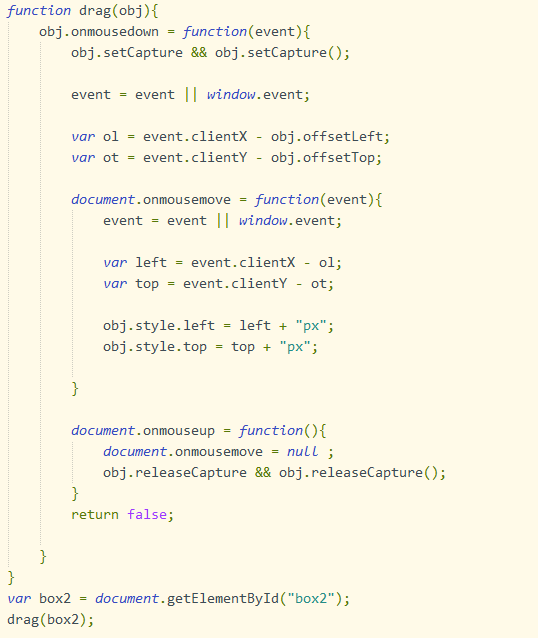
1. 拖拽div



1. 使div不偏移原位置

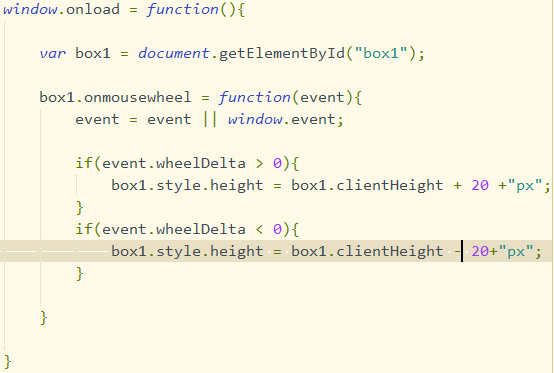


1. 封装函数



## 鼠标滚轮事件

1. onmousewheel,会在滚轮滚动时触发



1. 但火狐浏览器不支持该属性，需使用DOMMouseScroll来绑定事件，该事件需要通过addEventListener()函数来绑定