\* 内建对象

\* 宿主对象

\* 自定义对象

\*

\* 数组（Array）

\* - 数组也是一个对象

\* - 它和我们普通对象功能类似，也是用来存储一些值的

\* - 不同的是普通对象是使用字符串作为属性名的，

\* 而数组时使用数字来作为索引操作元素

\* - 索引：

\* 从0开始的整数就是索引

\* - 数组的存储性能比普通对象要好，在开发中我们经常使用数组来存储一些数据

使用typeof检查一个数组时，会返回object

\* 向数组中添加元素

\* 语法：数组[索引] = 值

\* 读取数组中的元素

\* 语法：数组[索引]

\* 如果读取不存在的索引，他不会报错而是返回undefined

\* 获取数组的长度

\* 可以使用length属性来获取数组的长度(元素的个数)

\* 语法：数组.length

\*

\* 对于连续的数组，使用length可以获取到数组的长度（元素的个数）

\* 对于非连续的数组，使用length会获取到数组的最大的索引+1

\* 尽量不要创建非连续的数组

\* 修改length

\* 如果修改的length大于原长度，则多出部分会空出来

\* 如果修改的length小于原长度，则多出的元素会被删除

向数组的最后一个位置添加元素

语法：数组[数组.length] = 值;

使用字面量来创建数组

语法:[]

使用字面量创建数组时，可以在创建时就指定数组中的元素

var arr = [1,2,3,4,5,10];

使用构造函数创建数组时，也可以同时添加元素，将要添加的元素作文构造函数的参数传递

元素之间使用,隔开

数组中的元素可以是任意的数据类型，也可以是对象，也可以是一个函数，数组中也可以放数组，如下这种数组我们称为二维数组

\* push()

\* - 该方法可以向数组的末尾添加一个或多个元素，并返回数组的新的长度

\* - 可以将要添加的元素作为方法的参数传递，

\* 这样这些元素将会自动添加到数组的末尾

\* - 该方法会将数组新的长度作为返回值返回

\* pop()

\* - 该方法可以删除数组的最后一个元素,并将被删除的元素作为返回值返回

\* unshift()

\* - 向数组开头添加一个或多个元素，并返回新的数组长度

\* - 向前边插入元素以后，其他的元素索引会依次调整

\* shift()

\* - 可以删除数组的第一个元素，并将被删除的元素作为返回值返回

所谓的遍历数组，就是将数组中所有的元素都取出来

<script type="text/javascript">

function Person(name , age , gender){

this.name = name;

this.age = age;

}

//修改Person原型的toString

Person.prototype.toString = function(){

return "Person[name="+this.name+",age="+this.age+"]";

};

//创建一个Person对象

var per = new Person("孙悟空",18);

var per2 = new Person("猪八戒",28);

var per3 = new Person("红孩儿",8);

var per4 = new Person("蜘蛛精",16);

var per5 = new Person("二郎神",38);

/\*

\* 将这些person对象放入到一个数组中

\*/

var perArr = [per,per2,per3,per4,per5];

/\*

\* 创建一个函数，可以将perArr中的满18岁的Person提取出来，

\* 然后封装到一个新的数组中并返回

\* arr

\* 形参，要提取信息的数组

\*/

function getAdult(arr){

//创建一个新的数组

var newArr = [];

//遍历arr，获取arr中Person对象

for(var i=0 ; i<arr.length ; i++){

var p = arr[i];

//判断Person对象的age是否大于等于18

if(p.age >= 18){

//如果大于等于18，则将这个对象添加到newArr中

//将对象放入到新数组中

newArr.push(p);

}

}

//将新的数组返回

return newArr;

}

var result = getAdult(perArr);

console.log(result);

</script>

\* 一般我们都是使用for循环去遍历数组，

\* JS中还为我们提供了一个方法，用来遍历数组

\* forEach()

\* - 这个方法只支持IE8以上的浏览器

\* IE8及以下的浏览器均不支持该方法，所以如果需要兼容IE8，则不要使用forEach

\* 还是使用for循环来遍历

\* forEach()方法需要一个函数作为参数

\* - 像这种函数，由我们创建但是不由我们调用的，我们称为回调函数

\* - 数组中有几个元素函数就会执行几次，每次执行时，浏览器会将遍历到的元素

\* 以实参的形式传递进来，我们可以来定义形参，来读取这些内容

\* - 浏览器会在回调函数中传递三个参数：

\* 第一个参数，就是当前正在遍历的元素

\* 第二个参数，就是当前正在遍历的元素的索引

\* 第三个参数，就是正在遍历的数组

\* slice()

\* - 可以用来从数组提取指定元素

\* - 该方法不会改变元素数组，而是将截取到的元素封装到一个新数组中返回

\* - 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

\* - 第二个参数可以省略不写,此时会截取从开始索引往后的所有元素

\* - 索引可以传递一个负值，如果传递一个负值，则从后往前计算

\* -1 倒数第一个

\* -2 倒数第二个

\* splice()

\* - 可以用于删除数组中的指定元素

\* - 使用splice()会影响到原数组，会将指定元素从原数组中删除

\* 并将被删除的元素作为返回值返回

\* - 参数：

\* 第一个，表示开始位置的索引

\* 第二个，表示删除的数量

\* 第三个及以后。。

\* 可以传递一些新的元素，这些元素将会自动插入到开始位置索引前边

<script type="text/javascript">

//创建一个数组

var arr = [1,2,3,2,2,1,3,4,2,5];

//去除数组中重复的数字

//获取数组中的每一个元素

for(var i=0 ; i<arr.length ; i++){

//console.log(arr[i]);

/\*获取当前元素后的所有元素\*/

for(var j=i+1 ; j<arr.length ; j++){

//console.log("---->"+arr[j]);

//判断两个元素的值是否相等

if(arr[i] == arr[j]){

//如果相等则证明出现了重复的元素，则删除j对应的元素

arr.splice(j,1);

//当删除了当前j所在的元素以后，后边的元素会自动补位

//此时将不会在比较这个元素吧，我需要在比较一次j所在位置的元素

//使j自减

j--;

}

}

}

console.log(arr);

</script>

\* concat()可以连接两个或多个数组，并将新的数组返回

\* - 该方法不会对原数组产生影响

\* join()

\* - 该方法可以将数组转换为一个字符串

\* - 该方法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

\* - 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

\* 如果不指定连接符，则默认使用,作为连接符

\* reverse()

\* - 该方法用来反转数组（前边的去后边，后边的去前边）

\* - 该方法会直接修改原数组

\* sort()

\* - 可以用来对数组中的元素进行排序

\* - 也会影响原数组，默认会按照Unicode编码进行排序

\* 即使对于纯数字的数组，使用sort()排序时，也会按照Unicode编码来排序，

\* 所以对数字进排序时，可能会得到错误的结果。

\*

\* 我们可以自己来指定排序的规则

\* 我们可以在sort()添加一个回调函数，来指定排序规则，

\* 回调函数中需要定义两个形参,

\* 浏览器将会分别使用数组中的元素作为实参去调用回调函数

\* 使用哪个元素调用不确定，但是肯定的是在数组中a一定在b前边

\* - 浏览器会根据回调函数的返回值来决定元素的顺序，

\* 如果返回一个大于0的值，则元素会交换位置

\* 如果返回一个小于0的值，则元素位置不变

\* 如果返回一个0，则认为两个元素相等，也不交换位置 \*

\* - 如果需要升序排列，则返回 a-b

\* 如果需要降序排列，则返回b-a

\* call()和apply()

\* - 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

\* - 当对函数调用call()和apply()都会调用函数执行

\* - 在调用call()和apply()可以将一个对象指定为第一个参数

\* 此时这个对象将会成为函数执行时的this

\* - call()方法可以将实参在对象之后依次传递

\* - apply()方法需要将实参封装到一个数组中统一传递

\*

\* - this的情况：

\* 1.以函数形式调用时，this永远都是window

\* 2.以方法的形式调用时，this是调用方法的对象

\* 3.以构造函数的形式调用时，this是新创建的那个对象

\* 4.使用call和apply调用时，this是指定的那个对象

\* 在调用函数时，浏览器每次都会传递进两个隐含的参数：

\* 1.函数的上下文对象 this

\* 2.封装实参的对象 arguments

\* - 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

\* - 在调用函数时，我们所传递的实参都会在arguments中保存

\* - arguments.length可以用来获取实参的长度

\* - 我们即使不定义形参，也可以通过arguments来使用实参，

\* 只不过比较麻烦

\* arguments[0] 表示第一个实参

\* arguments[1] 表示第二个实参 。。。

\* - 它里边有一个属性叫做callee，

\* 这个属性对应一个函数对象，就是当前正在指向的函数的对象

\* Date对象

\* - 在JS中使用Date对象来表示一个时间

创建一个Date对象

如果直接使用构造函数创建一个Date对象，则会封装为当前代码执行的时间

创建一个指定的时间对象

需要在构造函数中传递一个表示时间的字符串作为参数

日期的格式 月份/日/年 时:分:秒

var d2 = new Date("2/18/2011 11:10:30");

\* getDate()

\* - 获取当前日期对象是几日

\* getDay()

\* - 获取当前日期对象时周几

\* - 会返回一个0-6的值

\* 0 表示周日

\* 1表示周一

\* getMonth()

\* d2 = new Date("12/18/2011 11:10:30");

\* - 获取当前时间对象的月份

\* - 会返回一个0-11的值

\* 0 表示1月

\* 1 表示2月

\* 11 表示12月

\* getFullYear()

\* - 获取当前日期对象的年份

\* getTime()

\* - 获取当前日期对象的时间戳

\* - 时间戳，指的是从格林威治标准时间的1970年1月1日，0时0分0秒

\* 到当前日期所花费的毫秒数（1秒 = 1000毫秒）

\* - 计算机底层在保存时间时使用都是时间戳

利用时间戳来测试代码的执行的性能

获取当前的时间戳

var start = Date.now();

for(var i=0 ; i<100 ; i++){

console.log(i);

}

var end = Date.now();

console.log("执行了："+(end - start)+"毫秒");

\* Math

\* - Math和其他的对象不同，它不是一个构造函数，

\* 它属于一个工具类不用创建对象，它里边封装了数学运算相关的属性和方法

\* - 比如

\* Math.PI 表示的圆周率

\* abs()可以用来计算一个数的绝对值

\* Math.ceil()

\* - 可以对一个数进行向上取整，小数位只有有值就自动进1

\* Math.floor()

\* - 可以对一个数进行向下取整，小数部分会被舍掉

\* Math.round()

\* - 可以对一个数进行四舍五入取整

\* Math.random()

\* - 可以用来生成一个0-1之间的随机数

\* - 生成一个0-10的随机数

\* - 生成一个0-x之间的随机数

\* Math.round(Math.random()\*x)

\*

\* - 生成一个1-10

\* - 生成一个x-y之间的随机数

\* Math.round(Math.random()\*(y-x)+x)

\* max() 可以获取多个数中的最大值

\* min() 可以获取多个数中的最小值

\* Math.pow(x,y)

\* 返回x的y次幂

\* Math.sqrt()

\* 用于对一个数进行开方运算

\* 基本数据类型

\* String Number Boolean Null Undefined

\* 引用数据类型

\* Object

\*

\* 在JS中为我们提供了三个包装类，通过这三个包装类可以将基本数据类型的数据转换为对象

\* String()

\* - 可以将基本数据类型字符串转换为String对象

\* Number()

\* - 可以将基本数据类型的数字转换为Number对象

\* Boolean()

\* - 可以将基本数据类型的布尔值转换为Boolean对象

\* 但是注意：我们在实际应用中不会使用基本数据类型的对象，

\* 如果使用基本数据类型的对象，在做一些比较时可能会带来一些不可预期的结果

\* 方法和属性之能添加给对象，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，然后在调用对象的属性和方法

\* 调用完以后，在将其转换为基本数据类型

\* charAt()

\* - 可以返回字符串中指定位置的字符

\* - 根据索引获取指定的字符

\* charCodeAt()

\* - 获取指定位置字符的字符编码（Unicode编码）

\* String.formCharCode()

\* - 可以根据字符编码去获取字符

\* concat()

\* - 可以用来连接两个或多个字符串

\* - 作用和+一样

\* indexof()

\* - 该方法可以检索一个字符串中是否含有指定内容

\* - 如果字符串中含有该内容，则会返回其第一次出现的索引

\* 如果没有找到指定的内容，则返回-1

\* - 可以指定一个第二个参数，指定开始查找的位置

\*

\* lastIndexOf();

\* - 该方法的用法和indexOf()一样，

\* 不同的是indexOf是从前往后找，

\* 而lastIndexOf是从后往前找

\* - 也可以指定开始查找的位置

\* slice()

\* - 可以从字符串中截取指定的内容

\* - 不会影响原字符串，而是将截取到内容返回

\* - 参数：

\* 第一个，开始位置的索引（包括开始位置）

\* 第二个，结束位置的索引（不包括结束位置）

\* - 如果省略第二个参数，则会截取到后边所有的

\* - 也可以传递一个负数作为参数，负数的话将会从后边计算

\* substring()

\* - 可以用来截取一个字符串，可以slice()类似

\* - 参数：

\* - 第一个：开始截取位置的索引（包括开始位置）

\* - 第二个：结束位置的索引（不包括结束位置）

\* - 不同的是这个方法不能接受负值作为参数，

\* 如果传递了一个负值，则默认使用0

\* - 而且他还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

\* substr()

\* - 用来截取字符串

\* - 参数：

\* 1.截取开始位置的索引

\* 2.截取的长度

\* split()

\* - 可以将一个字符串拆分为一个数组

\* - 参数：

\* -需要一个字符串作为参数，将会根据该字符串去拆分数组

\* 如果传递一个空串作为参数，则会将每个字符都拆分为数组中的一个元素

\* toUpperCase()

\* - 将一个字符串转换为大写并返回

\* toLowerCase()

\* -将一个字符串转换为小写并返回

\* 正则表达式

\* - admin@atguigu.com

\* - admin@.com adminatguigu.com

\* - 邮件的规则：

\* 1.前边可以是xxxx乱七八糟

\* 2.跟着一个@

\* 3.后边可以是xxxx乱七八糟

\* 4..com获取其他的乱七八糟

\*

\* - 正则表达式用于定义一些字符串的规则，

\* 计算机可以根据正则表达式，来检查一个字符串是否符合规则，

\* 获取将字符串中符合规则的内容提取出来

\* 语法：

\* var 变量 = new RegExp("正则表达式","匹配模式");

\* 使用typeof检查正则对象，会返回object

\* var reg = new RegExp("a"); 这个正则表达式可以来检查一个字符串中是否含有a

\* 在构造函数中可以传递一个匹配模式作为第二个参数，

\* 可以是

\* i 忽略大小写

\* g 全局匹配模式

\* 正则表达式的方法：

\* 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，

\* 如果符合则返回true，否则返回false

\* 使用字面量来创建正则表达式

\* 语法：var 变量 = /正则表达式/匹配模式

\* 使用字面量的方式创建更加简单

\* 使用构造函数创建更加灵活

\* 使用 | 表示或者的意思

\* 创建一个正则表达式检查一个字符串中是否有字母

\* []里的内容也是或的关系

\* [ab] == a|b

\* [a-z] 任意小写字母

\* [A-Z] 任意大写字母

\* [A-z] 任意字母

\* [0-9] 任意数字

[^ ] 除了

\* split()

\* - 可以将一个字符串拆分为一个数组

\* - 方法中可以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

\* - 这个方法即使不指定全局匹配，也会全都插分

\* search()

\* - 可以搜索字符串中是否含有指定内容

\* - 如果搜索到指定内容，则会返回第一次出现的索引，如果没有搜索到返回-1

\* - 它可以接受一个正则表达式作为参数，然后会根据正则表达式去检索字符串

\* - serach()只会查找第一个，即使设置全局匹配也没用

搜索字符串中是否含有abc 或 aec 或 afc

\* match()

\* - 可以根据正则表达式，从一个字符串中将符合条件的内容提取出来

\* - 默认情况下我们的match只会找到第一个符合要求的内容，找到以后就停止检索

\* 我们可以设置正则表达式为全局匹配模式，这样就会匹配到所有的内容

\* 可以为一个正则表达式设置多个匹配模式，且顺序无所谓

\* - match()会将匹配到的内容封装到一个数组中返回，即使只查询到一个结果

\* replace()

\* - 可以将字符串中指定内容替换为新的内容

\* - 参数：

\* 1.被替换的内容，可以接受一个正则表达式作为参数

\* 2.新的内容

\* - 默认只会替换第一个

\* 量词

\* - 通过量词可以设置一个内容出现的次数

\* - 量词只对它前边的一个内容起作用

\* - {n} 正好出现n次

\* - {m,n} 出现m-n次

\* - {m,} m次以上

\* - + 至少一个，相当于{1,}

\* - \* 0个或多个，相当于{0,}

\* - ? 0个或1个，相当于{0,1}

\* 检查一个字符串中是否以a开头

\* ^ 表示开头

\* $ 表示结尾

\* 如果在正则表达式中同时使用^ $则要求字符串必须完全符合正则表达式

\* 创建一个正则表达式，用来检查一个字符串是否是一个合法手机号

手机号的规则：

\* 1 3 567890123 （11位）

\*

\* 1. 以1开头

\* 2. 第二位3-9任意数字

\* 3. 三位以后任意数字9个

\*

\* ^1 [3-9] [0-9]{9}$

\* 检查一个字符串中是否含有 .

\* . 表示任意字符

\* 在正则表达式中使用\作为转义字符

\* \. 来表示.

\* \\ 表示\

\*

\* 注意：使用构造函数时，由于它的参数是一个字符串，而\是字符串中转义字符，

\* 如果要使用\则需要使用\\来代替

\* \w

\* - 任意字母、数字、\_ [A-z0-9\_]

\* \W

\* - 除了字母、数字、\_ [^A-z0-9\_]

\* \d

\* - 任意的数字 [0-9]

\* \D

\* - 除了数字 [^0-9]

\* \s

\* - 空格

\* \S

\* - 除了空格

\* \b

\* - 单词边界

\* \B

\* - 除了单词边界

\* /^\s\*|\s\*$/g 匹配开头和结尾的空格

\* 电子邮件

\* hello .nihao @ abc .com.cn

\*

\* 任意字母数字下划线 .任意字母数字下划线 @ 任意字母数字 .任意字母（2-5位） .任意字母（2-5位）

\*

\* \w{3,} (\.\w+)\* @ [A-z0-9]+ (\.[A-z]{2,5}){1,2}

\* 浏览器已经为我们提供 文档节点 对象这个对象是window属性

\* 可以在页面中直接使用，文档节点代表的是整个网页

我们可以在事件对应的属性中设置一些js代码，

这样当事件被触发时，这些代码将会执行

这种写法我们称为结构和行为耦合，不方便维护，不推荐使用

\* 事件，就是用户和浏览器之间的交互行为，

\* 比如：点击按钮，鼠标移动、关闭窗口。。。

\* 可以为按钮的对应事件绑定处理函数的形式来响应事件

\* 这样当事件被触发时，其对应的函数将会被调用

绑定一个单击事件

像这种为单击事件绑定的函数，我们称为单击响应函数

\* 浏览器在加载一个页面时，是按照自上向下的顺序加载的，

\* 读取到一行就运行一行,如果将script标签写到页面的上边，

\* 在代码执行时，页面还没有加载，页面没有加载DOM对象也没有加载

\* 会导致无法获取到DOM对象

onload事件件会在整个页面加载完成之后才触发

\* 为window绑定一个onload事件

\* 该事件对应的响应函数将会在页面加载完成之后执行，

\* 这样可以确保我们的代码执行时所有的DOM对象已经加载完毕了

\* 将js代码编写到页面的下部就是为了，可以在页面加载完毕以后再执行js代码

getElementsByTagName()可以根据标签名来获取一组元素节点对象

这个方法会给我们返回一个类数组对象，所有查询到的元素都会封装到对象中

即使查询到的元素只有一个，也会封装到数组中返回

\* innerHTML用于获取元素内部的HTML代码的

\* 对于自结束标签，这个属性没有意义

\* 如果需要读取元素节点属性，

\* 直接使用 元素.属性名

\* 例子：元素.id 元素.name 元素.value

\* 注意：class属性不能采用这种方式，

\* 读取class属性时需要使用 元素.className

\* 定义一个函数，专门用来为指定元素绑定单击响应函数

\* 参数：

\* idStr 要绑定单击响应函数的对象的id属性值

\* fun 事件的回调函数，当单击元素时，该函数将会被触发

\* childNodes属性会获取包括文本节点在呢的所有节点

\* 根据DOM标签标签间空白也会当成文本节点

\* 注意：在IE8及以下的浏览器中，不会将空白文本当成子节点，

\* 所以该属性在IE8中会返回4个子元素而其他浏览器是9个

\* firstElementChild不支持IE8及以下的浏览器，

\* 如果需要兼容他们尽量不要使用

\* innerText

\* - 该属性可以获取到元素内部的文本内容

\* - 它和innerHTML类似，不同的是它会自动将html去除

previousElementSibling获取前一个兄弟元素，IE8及以下不支持

\* 根据元素的class属性值查询一组元素节点对象

\* getElementsByClassName()可以根据class属性值获取一组元素节点对象，

\* 但是该方法不支持IE8及以下的浏览器

\* document.querySelector()

\* - 需要一个选择器的字符串作为参数，可以根据一个CSS选择器来查询一个元素节点对象

\* - 虽然IE8中没有getElementsByClassName()但是可以使用querySelector()代替

\* - 使用该方法总会返回唯一的一个元素，如果满足条件的元素有多个，那么它只会返回第一个

\* document.querySelectorAll()

\* - 该方法和querySelector()用法类似，不同的是它会将符合条件的元素封装到一个数组中返回

\* - 即使符合条件的元素只有一个，它也会返回数组

\* document.querySelectorAll()

\* - 该方法和querySelector()用法类似，不同的是它会将符合条件的元素封装到一个数组中返回

\* - 即使符合条件的元素只有一个，它也会返回数组

\* document.createTextNode()

\* 可以用来创建一个文本节点对象

\* 需要一个文本内容作为参数，将会根据该内容创建文本节点，并将新的节点返回

\* appendChild()

\* - 向一个父节点中添加一个新的子节点

\* - 用法：父节点.appendChild(子节点);

\* insertBefore()

\* - 可以在指定的子节点前插入新的子节点

\* - 语法：

\* 父节点.insertBefore(新节点,旧节点);

\* replaceChild()

\* - 可以使用指定的子节点替换已有的子节点

\* - 语法：父节点.replaceChild(新节点,旧节点);

\* removeChild()

\* - 可以删除一个子节点

\* - 语法：父节点.removeChild(子节点);

\*

\* 子节点.parentNode.removeChild(子节点);

\* 使用innerHTML也可以完成DOM的增删改的相关操作

\* 一般我们会两种方式结合使用

\* confirm()用于弹出一个带有确认和取消按钮的提示框

\* 需要一个字符串作为参数，该字符串将会作为提示文字显示出来

\* 如果用户点击确认则会返回true，如果点击取消则返回false

\* 点击超链接以后，超链接会跳转页面，这个是超链接的默认行为，

\* 但是此时我们不希望出现默认行为，可以通过在响应函数的最后return false来取消默认行为

\* for循环会在页面加载完成之后立即执行，

\* 而响应函数会在超链接被点击时才执行

\* 当响应函数执行时，for循环早已执行完毕

\* 通过JS修改元素的样式：

\* 语法：元素.style.样式名 = 样式值

\*

\* 注意：如果CSS的样式名中含有-，

\* 这种名称在JS中是不合法的比如background-color

\* 需要将这种样式名修改为驼峰命名法，

\* 去掉-，然后将-后的字母大写

\*

\* 我们通过style属性设置的样式都是内联样式，

\* 而内联样式有较高的优先级，所以通过JS修改的样式往往会立即显示

\*

\* 但是如果在样式中写了!important，则此时样式会有最高的优先级，

\* 即使通过JS也不能覆盖该样式，此时将会导致JS修改样式失效

\* 所以尽量不要为样式添加!important

\* 语法：元素.style.样式名

\*

\* 通过style属性设置和读取的都是内联样式

\* 无法读取样式表中的样式

\* 获取元素的当前显示的样式

\* 语法：元素.currentStyle.样式名

\* 它可以用来读取当前元素正在显示的样式

\* 如果当前元素没有设置该样式，则获取它的默认值

\* 在其他浏览器中可以使用

\* getComputedStyle()这个方法来获取元素当前的样式

\* 这个方法是window的方法，可以直接使用

\* 需要两个参数

\* 第一个：要获取样式的元素

\* 第二个：可以传递一个伪元素，一般都传null

\*

\* 该方法会返回一个对象，对象中封装了当前元素对应的样式

\* 可以通过对象.样式名来读取样式

\* 如果获取的样式没有设置，则会获取到真实的值，而不是默认值

\* 比如：没有设置width，它不会获取到auto，而是一个长度

\*

\* 但是该方法不支持IE8及以下的浏览器

\*

\* 通过currentStyle和getComputedStyle()读取到的样式都是只读的，

\* 不能修改，如果要修改必须通过style属性

\* 定义一个函数，用来获取指定元素的当前的样式

\* 参数：

\* obj 要获取样式的元素

\* name 要获取的样式名

\* clientWidth

\* clientHeight

\* - 这两个属性可以获取元素的可见宽度和高度

\* - 这些属性都是不带px的，返回都是一个数字，可以直接进行计算

\* - 会获取元素宽度和高度，包括内容区和内边距

\* - 这些属性都是只读的，不能修改

\* offsetWidth

\* offsetHeight

\* - 获取元素的整个的宽度和高度，包括内容区、内边距和边框

\* offsetParent

\* - 可以用来获取当前元素的定位父元素

\* - 会获取到离当前元素最近的开启了定位的祖先元素

\* 如果所有的祖先元素都没有开启定位，则返回body

\* offsetLeft

\* - 当前元素相对于其定位父元素的水平偏移量

\* offsetTop

\* - 当前元素相对于其定位父元素的垂直偏移量

\* scrollWidth

\* scrollHeight

\* - 可以获取元素整个滚动区域的宽度和高度

\* scrollLeft

\* - 可以获取水平滚动条滚动的距离

\* scrollTop

\* - 可以获取垂直滚动条滚动的距离

当满足scrollHeight - scrollTop == clientHeight

说明垂直滚动条滚动到底了

\* disabled属性可以设置一个元素是否禁用，

\* 如果设置为true，则元素禁用

\* 如果设置为false，则元素可用

\* onmousemove

\* - 该事件将会在鼠标在元素中移动时被触发

\*

\* 事件对象

\* - 当事件的响应函数被触发时，浏览器每次都会将一个事件对象作为实参传递进响应函数,

\* 在事件对象中封装了当前事件相关的一切信息，比如：鼠标的坐标 键盘哪个按键被按下 鼠标滚轮滚动的方向。。。

\* 在IE8中，响应函数被处罚时，浏览器不会传递事件对象，

\* 在IE8及以下的浏览器中，是将事件对象作为window对象的属性保存的

解决事件对象的兼容性问题

event = event || window.event;

\* clientX和clientY

\* 用于获取鼠标在当前的可见窗口的坐标

\* div的偏移量，是相对于整个页面的

\*

\* pageX和pageY可以获取鼠标相对于当前页面的坐标

\* 但是这个两个属性在IE8中不支持，所以如果需要兼容IE8，则不要使用

\* 事件的冒泡（Bubble）

\* - 所谓的冒泡指的就是事件的向上传导，当后代元素上的事件被触发时，其祖先元素的相同事件也会被触发

\* - 在开发中大部分情况冒泡都是有用的,如果不希望发生事件冒泡可以通过事件对象来取消冒泡

取消冒泡

可以将事件对象的cancelBubble设置为true，即可取消冒泡

\* 为每一个超链接都绑定一个单击响应函数

\* 这里我们为每一个超链接都绑定了一个单击响应函数，这种操作比较麻烦，

\* 而且这些操作只能为已有的超链接设置事件，而新添加的超链接必须重新绑定

\* 我们希望，只绑定一次事件，即可应用到多个的元素上，即使元素是后添加的

\* 我们可以尝试将其绑定给元素的共同的祖先元素

\*

\* 事件的委派

\* - 指将事件统一绑定给元素的共同的祖先元素，这样当后代元素上的事件触发时，会一直冒泡到祖先元素

\* 从而通过祖先元素的响应函数来处理事件。

\* - 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

\* target

\* - event中的target表示的触发事件的对象

\* 使用 对象.事件 = 函数 的形式绑定响应函数，

\* 它只能同时为一个元素的一个事件绑定一个响应函数，

\* 不能绑定多个，如果绑定了多个，则后边会覆盖掉前边的

\* addEventListener()

\* - 通过这个方法也可以为元素绑定响应函数

\* - 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事件，需要一个布尔值，一般都传false

\*

\* 使用addEventListener()可以同时为一个元素的相同事件同时绑定多个响应函数，

\* 这样当事件被触发时，响应函数将会按照函数的绑定顺序执行

\*

\* 这个方法不支持IE8及以下的浏览器

\* attachEvent()

\* - 在IE8中可以使用attachEvent()来绑定事件

\* - 参数：

\* 1.事件的字符串，要on

\* 2.回调函数

\*

\* - 这个方法也可以同时为一个事件绑定多个处理函数，

\* 不同的是它是后绑定先执行，执行顺序和addEventListener()相反

\* addEventListener()中的this，是绑定事件的对象

\* attachEvent()中的this，是window

\* 需要统一两个方法this

\* 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

\* this是谁由调用方式决定

\* callback.call(obj)

\* 事件的传播

\* - 关于事件的传播网景公司和微软公司有不同的理解

\* - 微软公司认为事件应该是由内向外传播，也就是当事件触发时，应该先触发当前元素上的事件，

\* 然后再向当前元素的祖先元素上传播，也就说事件应该在冒泡阶段执行。

\* - 网景公司认为事件应该是由外向内传播的，也就是当前事件触发时，应该先触发当前元素的最外层的祖先元素的事件，

\* 然后在向内传播给后代元素

\* - W3C综合了两个公司的方案，将事件传播分成了三个阶段

\* 1.捕获阶段

\* - 在捕获阶段时从最外层的祖先元素，向目标元素进行事件的捕获，但是默认此时不会触发事件

\* 2.目标阶段

\* - 事件捕获到目标元素，捕获结束开始在目标元素上触发事件

\* 3.冒泡阶段

\* - 事件从目标元素向他的祖先元素传递，依次触发祖先元素上的事件

\*

\* - 如果希望在捕获阶段就触发事件，可以将addEventListener()的第三个参数设置为true

\* 一般情况下我们不会希望在捕获阶段触发事件，所以这个参数一般都是false

\*

\* - IE8及以下的浏览器中没有捕获阶段

\* 拖拽box1元素

\* - 拖拽的流程

\* 1.当鼠标在被拖拽元素上按下时，开始拖拽 onmousedown

\* 2.当鼠标移动时被拖拽元素跟随鼠标移动 onmousemove

\* 3.当鼠标松开时，被拖拽元素固定在当前位置 onmouseup

\* 当我们拖拽一个网页中的内容时，浏览器会默认去搜索引擎中搜索内容，

\* 此时会导致拖拽功能的异常，这个是浏览器提供的默认行为，

\* 如果不希望发生这个行为，则可以通过return false来取消默认行为

\*

\* 但是这招对IE8不起作用

当鼠标松开时，取消对事件的捕获

box1.releaseCapture && box1.releaseCapture();

设置btn01对鼠标按下相关的事件进行捕获

当调用一个元素的setCapture()方法以后，这个元素将会把下一次所有的鼠标按下相关的事件捕获到自身上

btn01.setCapture();

\* onmousewheel鼠标滚轮滚动的事件，会在滚轮滚动时触发，

\* 但是火狐不支持该属性

\*

\* 在火狐中需要使用 DOMMouseScroll 来绑定滚动事件

\* 注意该事件需要通过addEventListener()函数来绑定

event.wheelDelta 可以获取鼠标滚轮滚动的方向

向上滚 120 向下滚 -120

wheelDelta这个值我们不看大小，只看正负

wheelDelta这个属性火狐中不支持

在火狐中使用event.detail来获取滚动的方向

向上滚 -3 向下滚 3

\* 使用addEventListener()方法绑定响应函数，取消默认行为时不能使用return false

\* 需要使用event来取消默认行为event.preventDefault();

\* 但是IE8不支持event.preventDefault();这个玩意，如果直接调用会报错

\* 当滚轮滚动时，如果浏览器有滚动条，滚动条会随之滚动，

\* 这是浏览器的默认行为，如果不希望发生，则可以取消默认行为

function bind(obj , eventStr , callback){

if(obj.addEventListener){

//大部分浏览器兼容的方式

obj.addEventListener(eventStr , callback , false);

}else{

/\*

\* this是谁由调用方式决定

\* callback.call(obj)

\*/

//IE8及以下

obj.attachEvent("on"+eventStr , function(){

//在匿名函数中调用回调函数

callback.call(obj);

});

}

}

\* 键盘事件：

\* onkeydown

\* - 按键被按下

\* - 对于onkeydown来说如果一直按着某个按键不松手，则事件会一直触发

\* - 当onkeydown连续触发时，第一次和第二次之间会间隔稍微长一点，其他的会非常的快

\* 这种设计是为了防止误操作的发生。

\* onkeyup

\* - 按键被松开

\*

\* 键盘事件一般都会绑定给一些可以获取到焦点的对象或者是document

\* 可以通过keyCode来获取按键的编码

\* 通过它可以判断哪个按键被按下

\* 除了keyCode，事件对象中还提供了几个属性

\* altKey

\* ctrlKey

\* shiftKey

\* - 这个三个用来判断alt ctrl 和 shift是否被按下

\* 如果按下则返回true，否则返回false

\* BOM

\* - 浏览器对象模型

\* - BOM可以使我们通过JS来操作浏览器

\* - 在BOM中为我们提供了一组对象，用来完成对浏览器的操作

\* - BOM对象

\* Window

\* - 代表的是整个浏览器的窗口，同时window也是网页中的全局对象

\* Navigator

\* - 代表的当前浏览器的信息，通过该对象可以来识别不同的浏览器

\* Location

\* - 代表当前浏览器的地址栏信息，通过Location可以获取地址栏信息，或者操作浏览器跳转页面

\* History

\* - 代表浏览器的历史记录，可以通过该对象来操作浏览器的历史记录

\* 由于隐私原因，该对象不能获取到具体的历史记录，只能操作浏览器向前或向后翻页

\* 而且该操作只在当次访问时有效

\* Screen

\* - 代表用户的屏幕的信息，通过该对象可以获取到用户的显示器的相关的信息

\*

\*

\* 这些BOM对象在浏览器中都是作为window对象的属性保存的，

\* 可以通过window对象来使用，也可以直接使用

\* Navigator

\* - 代表的当前浏览器的信息，通过该对象可以来识别不同的浏览器

\* - 由于历史原因，Navigator对象中的大部分属性都已经不能帮助我们识别浏览器了

\* - 一般我们只会使用userAgent来判断浏览器的信息，

\* userAgent是一个字符串，这个字符串中包含有用来描述浏览器信息的内容，

\* 不同的浏览器会有不同的userAgent

\*

\* 火狐的userAgent

\* Mozilla/5.0 (Windows NT 6.1; WOW64; rv:50.0) Gecko/20100101 Firefox/50.0

\*

\* Chrome的userAgent

\* Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.82 Safari/537.36

\*

\* IE8

\* Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)

\*

\* IE9

\* Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)

\*

\* IE10

\* Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)

\*

\* IE11

\* Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E; rv:11.0) like Gecko

\* - 在IE11中已经将微软和IE相关的标识都已经去除了，所以我们基本已经不能通过UserAgent来识别一个浏览器是否是IE了

\* 如果通过UserAgent不能判断，还可以通过一些浏览器中特有的对象，来判断浏览器的信息

\* 比如：ActiveXObject

if(/firefox/i.test(ua)){

alert("你是火狐！！！");

}else if(/chrome/i.test(ua)){

alert("你是Chrome");

}else if(/msie/i.test(ua)){

alert("你是IE浏览器~~~");

}else if("ActiveXObject" in window){

alert("你是IE11，枪毙了你~~~");

}

\* History

\* - 对象可以用来操作浏览器向前或向后翻页

\* length

\* - 属性，可以获取到当成访问的链接数量

\* back()

\* - 可以用来回退到上一个页面，作用和浏览器的回退按钮一样

\* forward()

\* - 可以跳转下一个页面，作用和浏览器的前进按钮一样

\* go()

\* - 可以用来跳转到指定的页面

\* - 它需要一个整数作为参数

\* 1:表示向前跳转一个页面 相当于forward()

\* 2:表示向前跳转两个页面

\* -1:表示向后跳转一个页面

\* -2:表示向后跳转两个页面

\* Location

\* - 该对象中封装了浏览器的地址栏的信息

\* 如果直接将location属性修改为一个完整的路径，或相对路径

\* 则我们页面会自动跳转到该路径，并且会生成相应的历史记录

\* assign()

\* - 用来跳转到其他的页面，作用和直接修改location一样

\* reload()

\* - 用于重新加载当前页面，作用和刷新按钮一样

\* - 如果在方法中传递一个true，作为参数，则会强制清空缓存刷新页面

\* replace()

\* - 可以使用一个新的页面替换当前页面，调用完毕也会跳转页面

\* 不会生成历史记录，不能使用回退按钮回退

\* setInterval()

\* - 定时调用

\* - 可以将一个函数，每隔一段时间执行一次

\* - 参数：

\* 1.回调函数，该函数会每隔一段时间被调用一次

\* 2.每次调用间隔的时间，单位是毫秒

\*

\* - 返回值：

\* 返回一个Number类型的数据

\* 这个数字用来作为定时器的唯一标识

clearInterval()可以用来关闭一个定时器

\* 延时调用，

\* 延时调用一个函数不马上执行，而是隔一段时间以后在执行，而且只会执行一次

\*

\* 延时调用和定时调用的区别，定时调用会执行多次，而延时调用只会执行一次

\*

\* 延时调用和定时调用实际上是可以互相代替的，在开发中可以根据自己需要去选择

使用clearTimeout()来关闭一个延时调用

\* 通过style属性来修改元素的样式，每修改一个样式，浏览器就需要重新渲染一次页面

\* 这样的执行的性能是比较差的，而且这种形式当我们要修改多个样式时，也不太方便

\* 我们可以通过修改元素的class属性来间接的修改样式

\* 这样一来，我们只需要修改一次，即可同时修改多个样式，

\* 浏览器只需要重新渲染页面一次，性能比较好，

\* 并且这种方式，可以使表现和行为进一步的分离

/\*

\* 参数:

\* obj 要添加class属性的元素

\* cn 要添加的class值

\*

\*/

function addClass(obj , cn){

//检查obj中是否含有cn

if(!hasClass(obj , cn)){

obj.className += " "+cn;

}

}

/\*

\* 判断一个元素中是否含有指定的class属性值

\* 如果有该class，则返回true，没有则返回false

\*

\*/

function hasClass(obj , cn){

//判断obj中有没有cn class

//创建一个正则表达式

//var reg = /\bb2\b/;

var reg = new RegExp("\\b"+cn+"\\b");

return reg.test(obj.className);

}

/\*

\* 删除一个元素中的指定的class属性

\*/

function removeClass(obj , cn){

//创建一个正则表达式

var reg = new RegExp("\\b"+cn+"\\b");

//删除class

obj.className = obj.className.replace(reg , "");

}

/\*

\* toggleClass可以用来切换一个类

\* 如果元素中具有该类，则删除

\* 如果元素中没有该类，则添加

\*/

function toggleClass(obj , cn){

//判断obj中是否含有cn

if(hasClass(obj , cn)){

//有，则删除

removeClass(obj , cn);

}else{

//没有，则添加

addClass(obj , cn);

}

}

\* JSON

\* - JS中的对象只有JS自己认识，其他的语言都不认识

\* - JSON就是一个特殊格式的字符串，这个字符串可以被任意的语言所识别，

\* 并且可以转换为任意语言中的对象，JSON在开发中主要用来数据的交互

\* - JSON

\* - JavaScript Object Notation JS对象表示法

\* - JSON和JS对象的格式一样，只不过JSON字符串中的属性名必须加双引号

\* 其他的和JS语法一致

\* JSON分类：

\* 1.对象 {}

\* 2.数组 []

\*

\* JSON中允许的值：

\* 1.字符串

\* 2.数值

\* 3.布尔值

\* 4.null

\* 5.对象

\* 6.数组

\* 将JSON字符串转换为JS中的对象

\* 在JS中，为我们提供了一个工具类，就叫JSON

\* 这个对象可以帮助我们将一个JSON转换为JS对象，也可以将一个JS对象转换为JSON

\* json --> js对象

\* JSON.parse()

\* - 可以将以JSON字符串转换为js对象

\* - 它需要一个JSON字符串作为参数，会将该字符串转换为JS对象并返回

\* JS对象 ---> JSON

\* JSON.stringify()

\* - 可以将一个JS对象转换为JSON字符串

\* - 需要一个js对象作为参数，会返回一个JSON字符串

\* JSON这个对象在IE7及以下的浏览器中不支持，所以在这些浏览器中调用时会报错

如果需要兼容IE7及以下的JSON操作，则可以通过引入一个外部的js文件来处理

\* eval()

\* - 这个函数可以用来执行一段字符串形式的JS代码，并将执行结果返回

\* - 如果使用eval()执行的字符串中含有{},它会将{}当成是代码块

\* 如果不希望将其当成代码块解析，则需要在字符串前后各加一个()

\*

\* - eval()这个函数的功能很强大，可以直接执行一个字符串中的js代码，

\* 但是在开发中尽量不要使用，首先它的执行性能比较差，然后它还具有安全隐患