每一个在浏览器中打开的网页，浏览器都会自动的给当前文档创建一个window对象

window对象是js最外围的一个对象，也是一个全局对象，我们在window对象中所编写的一切程序，都会作为window对象的属性和方法而存在

三大系统弹出框：

警告提示弹出框：

alert方法名称 1参数（实际参数）

alert(1)

该方法没有返回值，执行结果是undefined

确认取消弹出框：

confirm(抛出的问题)

该方法的返回值 true 或者 false

输入弹出框：

prompt(提示，文本框默认值);

该方法的返回值：文本框内容 或者 null

页面输出

document.write("人生就像一杯茶，会苦一阵子，但不会苦一辈子！");

控制台输出：console.log()

查找变量值、排错（错误调试）

console.log(a);

单行注释和多行注释 // /\* \*/

量：是容器

变量、常量

变量：可变的量（容器）

常量：不可变的量（容器）

变量声明的关键字：var、let

常量声明的关键字：const

赋值的运算顺序是从右向左的

var声明变量：

var 变量名称 =(赋值) 所赋的值;（结束语）

var box = "bananas";

var Alert = alert;

变量声明未赋值：变量的值是undefined

一次声明多个变量，使用逗号隔开；多个变量，可以赋值，也可以不赋值。

var name = "苏志刚",age = 18,sex = "boy",wife = "cjk",son;

var name = window.name = a = "窗口名称";

非严格模式下：变量不声明，可以直接使用（赋值）

num = 10;

严格模式下：量必须先声明，后使用

function fn(){

// "use strict"; // 严格模式

var num1 = num = 10;

}

fn();

全局变量、局部变量：

var声明的变量是局部变量；不使用var声明的变量，是全局变量（非严格模式）；如果是严格模式下，window.a

function fn(){

var color = "red";

num = 20;

};

fn();

console.log(color,num);

变量声明提升：var

console.log(num);

var color,num = "blue";

var a\_1 = "a";

var a\_2 = "1";

标识符：变量名称、常量名称、属性名称、函数名称、参数名称

标识符的命名规则：

1、不能以数字开头

2、不能出现除$符、下划线以外的任何特殊符号

3、关键字不能作为标识符去使用，保留字不建议作为标识符去使用

阻断性：

1、报错阻断（只会阻断当前的栈内存）

2、弹出阻断（阻断整个文档的加载）

代码块：由一条以上的多条语句组合到一起叫代码块（花括号包起来的）

变量的声明：var

1、var声明的变量会作为window对象的属性和方法而存在

2、存在变量声明提升

3、可以重复声明

4、声明未赋值，其值是undefined

5、不存在块级作用域，存在函数作用域（局部变量、全局变量）

变量的声明：let

# 1、let声明的变量不会作为window对象的属性或者方法而存在

2、let声明的变量不存在变量声明提升

3、不可以重复声明

4、let声明不赋值，变量的值是undefined

5、let声明的变量存在块级作用域，也存在函数作用域（稍后讲）

常量的声明：const

1、常量一旦声明，不可改变

2、常量声明的同时，必须赋值

3、不可重复声明

4、不存在提升

5、不作为window对象的属性和方法而存在

6、存在块级作用域，也存在函数作用域

// 通过ID选择器获取一个元素节点对象：

// document.getElementById(id名称)

// var b = document.getElementById("box");

// console.log(b);

// 文档的加载是上而下的

// 预加载事件：

window.onload = function(){

文档加载完毕之后，执行的程序

}

// window.onload = function(){

// var b = document.getElementById("box");

// console.log(b);

// }

// 数据类型：字符串、数字、布尔、null、undefined、对象

// symbol（后面老师会讲到==》实训）

object string number boolean undefined null symbol

第二天

undefined:

变量声明未负值，变量的值是undefined

对象下不存在该属性，获取该属性时是undefined

函数没有返回值，函数的执行结果是undefined

数组中不存在下标对应的元素时，获取该下标对应的元素时值是undefined

字符串下标对应的没有字符，undefined;

null:是一个特殊的空对象，在内存中不站位

4、undefined是undefined类型的

5、函数没有返回值

6、形参对应的没有实参

null：

1、是一个特殊的空对象

2、在内存中不占位

3、DOM0级删除事件null

4、该对象下不存在任何属性，也不可以添加任何属性

1种复杂数据类型：对象（object） 构造函数：Object

直接量方式创建对象：{}

实例化对象的方式创建对象：new Object()

对象下属性之间使用逗号隔开，属性是一个键值对（key:value）

var obj = {

name: "武靖龙",

age: 21

};

对象属性的读取：

1、obj.属性名

2、obj[字符串属性名]

动态添加属性：

1、obj.属性名 = 属性值

2、obj[字符串属性名] = 属性值

删除对象下的属性：delete

delete obj.属性名

delete obj[字符串属性名]

修改对象的属性值：

1、obj.属性名 = 属性值

2、obj[字符串属性名] = 属性值

数据类型检测：

两种语法结构：检测结果都是一样的，如果是操作符，考虑操作符的优先级

检测结果：string、number、boolean、undefined、object、function

1、typeof 操作符

2、typeof() 方法

松散类型：变量本身不存在类型，变量的类型来自于所赋的值

从内存上划分变量类型：基本类型、引用类型

基本类型：5种简单数据类型

引用类型：1种复杂数据类型

基本类型的变量存储在栈内存

引用类型的变量存储在堆内存

程序是在栈内存中执行的

在文档中有一个大的堆内存（window对象），堆内存中可以存在多个栈内存，也可以存在多个堆内存...

console.log(a);

数据类型转换：

一、各种数据类型转成字符串类型

强转函数：String(待转换的数据)

对象下的方法：\*.toString()

二、各种数据类型转成数字类型

强转函数：Number()

转成整型数字：parseInt()

转成浮点型数字：parseFloat()

使用Number强转函数，转换结果为0：0、-0、false、""、" "、null、"0"

待转换的数据对应的是数字，则转成数字；如果不是数字，则转成NaN或者0

parseInt、parseFloat：整型数字、浮点型数字

1、数字或者字符串类型的数字，转换成数字

2、以数字开头的字符串，转换为数字

3、除去以上两点，剩余的数据都转换成NaN

三、各种数据类型转成布尔类型

强转函数：Boolean

操作符：逻辑非! 能把数据转成其对应的反的布尔值

对应布尔值为false：false、NaN、""、undefined、null、0、-0

所有对象都是真值

所有的方法或者函数都是真值

四、隐式类型转换

if("a"){

alert(1);

}

五、判断是否是NaN: isNaN

Number.isNaN：严格判断是否是NaN，不牵涉隐式类型转换

结果：true false

六、保留小数位：\*.toFixed(n)

1、\*必须是一个数字

2、返回值是一个字符串类型的数字

3、n代表的是保留的小数位（0 - 100）

4、遵循四舍五入的原则

七、判断是否是整型数字

Number.isInteger(待判断的数字)

不牵涉隐式类型转换

操作符：

一、按条目划分：

一元操作符||单目操作符

只能处理一个操作数的操作符

typeof ! + - ++ -- new delete

自增、自减：++ --

++ -- 必须得配合着变量使用

++在前，先加后取

++在后，先取后加

二元操作符||双目操作符

处理两个操作数的操作符

+ - \* / % && || instanceof...

三元操作符||条件操作符

判断条件 ? 结果1 : 结果2;

var num = 1 - true ? 3 : 4;

按功能划分操作符：

+功能：数学运算相加（操作数必须是数字）、拼接（操作数至少有一个是字符串）、正号（只有一个操作数，自动转型）

1、算术运算符：+ - \* / %

%求余：

1-1、求余数小于被求余数，余数是求余数本身

1-2、求余数大于被求余数，余数就是余数

1-3、任何整数对1求余，结果是0

1-4、任何小数对1求余，结果是小数部分

1-5、任何数对0求余，结果是NaN

2、比较运算符：> < >= <= == === != !==

凡是牵涉到比较运算的，其运算结果都是布尔类型（true false）

> < >= <=:

2-1、如果两边的操作数都是数字，直接比较大小

2-2、如果两边的操作数不同等类型，使用Number强转函数转换为数字，再进行比较

2-3、如果两边的操作数都是字符串类型的，比较的是unicode编码值

2-4、NaN和任何数据比较，结果都是false。包括它本身

== 比较值：

1、null除了和undefined以及它本身之外的任何数据比较，结果都是false

2、undefined除了和null以及它本身之外的任何数据比较，结果都是false

3、两边的操作数是同等类型的数据，直接进行值的比较

4、两边的操作数是不同等类型的数据，先使用Number强转函数转换，然后再进行值的比较

5、NaN和任何数据比较，结果都是false。包括它本身

6、引用类型之间的比较，比较的是地址指针

=== 值和类型都相同，结果为true

!= 只要值不相等，结果就为true

!== 值或者类型不相等，结果就为true

3、逻辑运算符

逻辑与&&、逻辑或||、逻辑非!

逻辑与或都是短路操作符

逻辑与是找假值，一旦找到假值，则不会再向下运算，其运算结果就是对应的那个假值；如果一直运算到最后，都没有找到假值，则运算结果就是最后一个操作数

逻辑或是找真值的，一旦找到真值，则不会再向下运算，其运算结果就是对应的那个真值；如果一直运算到最后，都没有找到真值，则运算结果就是最后一个操作数

逻辑与的优先级高于逻辑或的

4、赋值运算符：+= -= \*= /= %= <<= >>= &= |=（此处省略一万字）

+= 等同于 = +

5、条件运算符：三元运算符

6、加性运算符：+ -

7、乘兴运算符： \* / %

8、()运算符：优先级最高

9、[]运算符 数组

10、{}运算符 对象

11、逗号运算符：,

12、分号运算符：;结束语

13、反引号运算符：`${变量}`

14、typeof运算符：检测变量类型

15、instanceof运算符：检测一个实例对象是否属于一个函数对象（构造函数）

16、delete运算符：删除对象下属性

17、new运算符：实例化构造函数，得到实例对象

18、in运算符：判断一个属性（可枚举属性）是否属于一个对象

19、...运算符：取出数组中的元素

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

console.log(...arr,arr);

20、成员运算符.：优先级最高

21、字符串运算符：+ +=

22、自增、自减运算符：++ --

23、位移运算符（选学）：& | ~ << >> ......

第三天

// 如果省略花括号，有且仅有一条语句属于其相对应的代码块

// 流程控制语句：if语句、switch语句

// if语句：按分支结构划分

// 单分支：if(判断条件){代码块}

// 条件成立，执行其相对应的代码块

// var weather = prompt("请输入英文天气：","必须是英文...")

// if(weather == "rainy"){

// alert("出门带伞...");

// }

// 省略花括号：

// if(false){

// alert(1);

// }

// alert(2);

// \*\*\*：如果在外级作用域内使用声明了变量，则到内层作用域内，可以重复声明变量（该变量存在块级作用域）===>只要该变量存在块级作用域，则在不同的作用域内，可以重复声明

// let color = "red";

// if(color == "red"){

// const color = "blue";

// if(true){

// const color = "pink";

// console.log(color);

// }

// }

// console.log(color);

// 双分支：if(判断条件){代码块1}else{代码块2}

// var name = prompt("请输入姓名：","中文姓名...");

// if(name == "苏志刚"){

// alert("班长");

// }else{

// alert("....");

// }

// ;一条空语句

// if(false);else;

// var month = "6";

// if(month >= 4 && month <= 6){

// alert("夏季");

// }else{

// alert("其它季节");

// }

// // 判断条件 ? 结果1 : 结果2;

// month >= 4 && month <= 6 ? alert("夏季") : alert("其它季节");

// alert(month >= 4 && month <= 6 ? "夏季" : "其它季节");

// 多分支：会存在多个判断条件，但是只会执行一个相对应的代码块（判断条件成立）

// if(){}else if(){}else if().....else{}

// 多分支是双分支嵌套的变形

// if(true){

// alert(1);

// }else{

// if(true){

// alert(2);

// }else if(true){

// alert(3);

// }else{

// alert("默认");

// }

// }

// var score = prompt("请输入成绩：","成绩范围0 - 100之间...");

// var score\_1 = parseFloat(score);

// var score\_2 = Number(score);

// var res = null;

// if(score == null || score == ""){

// res = "请输入成绩..."

// }else{

// // if(isNaN(score\_1) || isNaN(score\_2)){

// // res = "请输入数字...";

// // }

// if(!Number.isFinite(score\_1) || !Number.isFinite(score\_2)){

// res = "请输入数字....";

// }else{

// if(score < 0 || score > 100){

// res = "请输入正确的数字...";

// }else{

// if(score == 0){

// res = "你是不是傻！！！";

// }else{

// if(score < 60){

// res = "你是不是有点傻！！！";

// }else{

// if(score < 90){

// res = "你是人了！！！";

// }else{

// if(score < 100){

// res = "别飘了！！！";

// }else{

// res = "作弊了！！！";

// }

// }

// }

// }

// }

// }

// }

// alert(res);

// var score = prompt("请输入成绩：","成绩范围0 - 100之间...");

// var score\_1 = parseFloat(score);

// var score\_2 = Number(score);

// var res = null;

// if(score == "" || score == null){

// res = "请输入成绩...";

// }else if(!Numbere.isFinite(score\_1) || !Number.isFinite(score\_2)){

// res = "请输入数字...";

// }else if(score < 0 || score > 100){

// res = "请输入正确的数字...";

// }else if(){}.....

// else{

// }

// alert(res);

// switch语句：当分支结构超过3个时，不要再使用if多分支语句（规范）

// 关键字：switch、case、break、default

// break:跳出执行的代码块

/\*

switch(true){

case 判断条件1:

代码块1

break;

case 判断条件2:

代码块2

break;

case 判断条件3:

代码块3

break;

case 判断条件4:

代码块4

break;

.....

default:

默认代码块

}

// var score = prompt("请输入成绩：","成绩范围0 - 100之间...");

// var score\_1 = parseFloat(score);

// var score\_2 = Number(score);

// var res = null;

// switch(true){

// case score == "" || score == null:

// res = "请输入成绩...";

// break;

// .....

// }

// if(score == "" || score == null){

// res = "请输入成绩...";

// }else if(!Numbere.isFinite(score\_1) || !Number.isFinite(score\_2)){

// res = "请输入数字...";

// }else if(score < 0 || score > 100){

// res = "请输入正确的数字...";

// }else if(){}.....

// else{

// }

// 季节判断：输入月份

// 班级输入判断：长度是5位，必须以四位数字开头，后跟一个字母

// 循环：在一定的条件内重复不断的执行一件事情（循环体）

// js:四种 for while do...while for...in

// 一种：for...of

// for:是一种先判断，后运行的循环语句。判断条件成立，执行循环体。

// for有可能一次都不执行，有可能一直执行（死循环）

// 语法结构：

for(初始值;判断条件;步长更新){

循环体

}

// for(var i=1;i<=5;i++){

// console.log(i);

// }

// 循环的执行顺序：

// for(1;2;3){4}

// 1==>2==>4==>3

// 2==>4==>3

// 2==>4==>3

// ....

// 2（判断条件不成立，循环执行结束）

// 初始值只执行一次

// 判断条件比循环体执行的次数多一次

// 步长更新和循环体执行的次数相同

// 省略初始值：

// i = 1;

// var i = null;

// for(;i<=5;i+=2){

// console.log(i);

// }

// var i = 1;

// 省略初始值、判断条件、步长更新

// for(;;){}

// console.log("123");

// 死循环：

// for(var i=1;i>=0;i++){}

// for(var i=1;i<=10;i--){}

// for(var i=1;true;i++){}

// for(var i=1;{};i++){}

// for(var i=0;"a";i++){}

// 循环执行完毕，程序才会继续向下执行

// for(var i="";i<51;i+=7){

// console.log(i);

// }

// console.log(i);

// for(let i=0;i<10;i++){}

// console.log(i);

// 循环也是可以省略花括号：有且仅有一条语句属于其相对应的代码块

// for(var i=0;i<=10;i+=1)

// if(i%3 == 0)

// ;

// console.log(i);//

// 有一群猴子，摘了一堆桃子，第一天吃掉了所有桃子的一半，有一个特殊的猴子，偷吃了一个，第二天又吃掉了剩余桃子的一半，还是这个特殊的猴子，又偷吃了一个，长此以往，第十天，这堆桃子，就剩下了一个，这个特殊的猴子被发现了，然后被揍死了。问题：猴子摘了多了个桃子？

// var peach = 1; // 第10天

// var days = 9;

// for(var i=days;i>=1;i--){

// peach = (peach + 1)\*2;

// console.log("第" + i + "天：" + peach + "桃子");

// }

第四天

跳转语句：

break:中断当前循环

continue:跳出当前循环，继续下一次循环

// while循环：先判断后运行的循环语句

// 区别：while使用在循环次数不固定时

// for使用在循环次数固定时

/\*

while(判断条件){

循环体

}

\*/

// var str = "fsgahaafaaaaagagjsa";

// while(str.includes("a")){

// str = str.replace("a","\*");

// }

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

// str = str.replace("a","\*");

// }

// for(var i=0;str.includes("a");i++){

// str = str.replace("a","\*");

// }

// do...while:先执行，后判断的循环语句

// 也是用在循环次数不固定时

// do{

// alert(1);

// }while(false)

// var str = "fsgahaafaaaaagagjsa";

// do{

// str = str.replace("a","\*");

// }while(str.includes("a"));

// for...in...:遍历字符串、数组、对象

// 遍历出来的是属性、下标

// 使用for...in时，只能去遍历对象

// for(var 变量 in 对象){}

// var str = "abcdef";

// for(var index in str){

// console.log(index);

// }

// var obj = {

// width:"100px",

// height: "100px",

// background: "pink"

// }

// for(var p in obj){

// // console.log(p);

// if(p == "width"){

// console.log(obj[p]);

// }

// }

// 使用for...in遍历字符串、数组，效率低下

// var str = "";

// console.time("执行时间：");

// for(var i=0;i<str.length;i++){}

// console.timeEnd("执行时间：");

// console.time("执行时间：");

// for(var i in str){}

// console.timeEnd("执行时间：");

// for...of:遍历对象，遍历出的是对象属性的值

// var obj = {

// name: "张三",

// age: 18

// }

// var arr = [1,2,3];

// for(var val of arr){

// console.log(key + ":" + val);

// }

// var m = new Map();

// m.set("name","张三");

// m.set("age",18);

// for(var [key,val] of m){

// console.log(key,val);

// }

金字塔

/\*

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

1 4

2 3

3 2

4 1

5 0

\*/

// var rows = 10;

// for(var i=1;i<=rows;i++){ // 行

// // 输出空格的循环

// for(var j=1;j<=rows - i;j++){

// document.write("&nbsp;");

// }

// // 输出\*的循环

// for(var k=1;k<=i;k++){

// document.write("\*&nbsp;");

// }

// document.write("<br>");

// }

// var rows = 10;

// for(var i=1;i<=rows;i++){ // 行

// // 输出空格的循环

// for(var j=1;j<=rows - i;j++){

// document.write("&nbsp;");

// }

// // 输出\*的循环

// for(var k=1;k<=i;k++){

// if(i == 1 || i == rows){

// document.write("\*&nbsp;");

// }else{

// if(k == 1 || k == i){

// document.write("\*&nbsp;");

// }else{

// document.write("&nbsp;&nbsp;");

// }

// }

// }

// document.write("<br>");

// }

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

...

1 1

2 3

3 5

4 7

5 9 2\*i-1

// var rows = 5;

// for(var i=1;i<=rows;i++){

// for(var j=1;j<=rows - i;j++){

// document.write("&nbsp;");

// }

// for(var k=1;k<=2\*i - 1;k++){

// if(i == 1 || i == rows){

// document.write("\*");

// }else if(k == 1 || k == 2\*i - 1){

// document.write("\*");

// }else{

// document.write("&nbsp;");

// }

// }

// document.write("<br>");

// }

/\*

\*\*\* \*\*\*

\* \*\*\*\*\*\*\* \*

\* \*

\* \*

\* \*

\* \*

\*/

// 在浏览器上输出1 - 10000之间能同时被3、5、7整除的整数

// var min = 1;

// var max = 10000;

// var c = 10;

// var n = 10;

// for(var i=min;i<=max;i++){

// if(i%3 == 0 && i%5 == 0 && i%7 ==0){

// document.write(i + "&nbsp;");

// if(++c%n == 0) document.write("<br>");

// }

// }

数组

// 数组是对象中的一种，属于引用类型，存储在堆内存

// 数组对应的构造函数：Array

// 数组的创建：

// 直接量：[]

// new关键字实例化构造函数的方式：new Array()

// 数组的长度属性：length（可读写）

// var arr = [1,2,3,4,5]; // 紧密数组

// 数组的读取：数组[下标]

// 数组的动态添加以及修改：数组[下标] = 赋值

// 数组元素的删除：delete 删除之后，数组是稀疏数组

// var arr = new Array(1,2,3,4,5,6);

// var a = [1,2,3,4,5];

// var b = a;

// b.length = 3;

// console.log(a.length);

// 使用...克隆arr

// var b = [...a];

// 多维数组：

var arr = []; // 北京站

arr[0] = []; // 昌平站

arr[0][0] = []; // 昌平线

arr[0][0][0] = ["西山口","十三陵","昌平"]; // 某一列地铁

第五天

// 多维数组：

// var arr = [[],[],[],[1,[2]],3];

// arr[4] arr[3][0] arr[3][2][0]

// arr.length = 9;

// 数组的遍历：for、while、do...while

// for...of遍历可迭代的对象

// var arr = [

// 1,2,3,4

// ];

// for(var v of arr){

// console.log(v);

// }

// var arr = [1,2,3,4,5,6,7,[8,9,10]];

// var sum = 0;

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

// if(Array.isArray(arr[i])){

// for(var j=0;j<arr[i].length;j++){

// sum += arr[i][j];

// }

// }else{

// sum += arr[i];

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

// }

// }

// 数组的检测：

// typeof ==> object

// instanceof ==> [] instanceof Array

// instanceof:检测一个实例对象是否属于一个函数对象

// Array.isArray()

// [].constructor.name ==> "Array"

// Object.prototype.toString.call(待检测的数据) ==》"[Object Array]"

// eval:让字符串作为程序去执行

// 数组的方法：

// 一、静态方法：

// 是否是数组：Array.isArray()

// 伪数组转数组：Array.from()

// 创建数组：Array.of()

// function createArray(){

// return Array.of(...arguments);

// }

// 二、原型方法：

// 数组的栈方法：

// 压栈：\*.push(ele1,ele2,...)

// 向数组的尾部添加一个或者多个元素，并返回数组新的长度

// 出栈：\*.pop()

// 从数组的尾部删除一个元素，并返回被删除的元素

// 数组的队列方法：

// 压队：\*.unshift(ele1,ele2,...)

// 向数组的头部添加一个或者多个元素，并返回数组新的长度

// 出队：\*.shift()

// 从数组的头部删除一个元素，并返回被删除的元素

// 数组的反转（不使用reverse方法）：

//第一种

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

// var str = [];

// for(var i=0;i<=arr.length-1;i++){

// str.unshift(arr[i]);

// }

// 第二种

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

// var str = [];

// for(var i=arr.length-1;i>=0;i--){

// str.push(arr[i]);

// }

// var tempArr = [];

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

// // 1 2 3 4

// tempArr.unshift(arr[i]);

// }

// for(var i=arr.length - 1;i>=0;i--){

// // 4 3 2 1

// tempArr.push(arr[i]);

// }

// 不使用任何系统方法：

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

// str[arr.length-1-i] = arr[i];

// }

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

// tempArr[arr.length - 1 - i] = arr[i];

// }

// 数组的转换方法：

// 数组转字符串：\*.toString() \*.join(分隔符)

// 字符串转数组：\*.split(分隔符)

// 统计字符串中每一个字符出现的次数：

// {a:3,d:4...}

// var str = "fdsghfjgsajfgja";

// var obj = {};

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

// obj[str[i]] = str.split(str[i]).length - 1;

// }

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

// var count = 0;

// for(var j=0;j<str.length;j++){

// if(str[i] == str[j]){

// count++;

// }

// }

// obj[str[i]] = count;

// }

// 判断该字符串中是否存在某个字符或者字符串（使用split）

// var str = "fdshakfjhdsdkajf"; // dsk

// if(str.split("dsk").length - 1 == 0){

// alert("不存在");

// }else{

// alert("存在");

// }

// 数组的位置方法：如果未出现，返回值-1

// 从指定数组中从前向后查找某个元素首次出现的位置（下标）：\*.indexOf(元素)

// 从指定数组中从前向后查找某个元素最后出现的位置（下标）：\*.lastIndexOf(元素)

// 数组的去重：

// 方式一：

// var arr = [1,2,3,4,5,1,2,1,1,1,6,3,7];

// var tempArr = [];

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

// // 判断tempArr中是否存在遍历的arr中的元素，如果不存在，则添加到tempArr中

// if(tempArr.indexOf(arr[i]) == -1){

// tempArr.push(arr[i]);

// }

// }

// 方式二：

// var index = 0;

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

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

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

// j = ++i;

// }

// }

// tempArr[index++] = arr[i];

// }

// 方式三：利用的是对象属性的唯一性

// var obj = {};

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

// if(!(obj[arr[i]] in obj)){

// tempArr.push(arr[i]);

// obj[arr[i]] = arr[i];

// }

// if(obj[arr[i]] != 1){

// tempArr.push(arr[i]);

// obj[arr[i]] = 1;

// }

// }

// 今天晚上的作业：如果数组中存在多个NaN，一个都不要

// var arr = [1,2,3,4,1,4,2,3,NaN,6,5,NaN,7,9,6,NaN];

// var str = [];

// 第一种方法

// arr.join("").split(NaN).join("").split("")

//第二种方法

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

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

// if(arr[i]==arr[j] || Number.isNaN(arr[i]) == true){

// j=++i;

// }

// }

// str.push(arr[i]);

// }

// if(Number.isNaN(str[str.length-1])){

// str.splice(str.length-1,1);

// }

// console.log(str);

// 第三种方法

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

// if(!str.includes(arr[i])){

// if(!Number.isNaN(arr[i])){

// str.push(arr[i]);

// }

// }

// }

// str.sort(function(a,b){

// return a<b;

// })

// console.log(str);

// 数组的排序方法：

// 翻转：\*.reverse() str.split("").reverse().join("")

// 排序：\*.sort()

// 该方法的返回值是排序之后的数组，默认是升序（按照unicode编码值进行排序）；会改变原本的数组。（高阶函数）

// var arr = [9,12,23,87,98,4,114,66];

// // arr.sort(function(x,y){

// return x<y;

// });

随机排序

// arr.sort(function(x,y){

// return Math.random() > 0.5;

// });

// console.log(arr);

// 冒泡排序：（冒大泡、冒小泡）

// 15 18 9 6 7 33 88 i=0 j=1 2 3 4 5 6

// 18 15 9 6 7 33 88

// 18 15 9 6 7 33 88

// 18 15 9 6 7 33 88

// 18 15 9 6 7 33 88

// 33 15 9 6 7 18 88

// 88 15 9 6 7 18 33 i=1 j=2 3 4 5 6

// .....

// var arr = [15,18,9,6,7,33,88];

// arr[0] arr[1]

// var temp = arr[0];

// arr[0] = arr[1];

// arr[1] = temp;

// arr[0] = arr[1];

// arr[1] = arr[0];

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

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

// if(arr[i] < arr[j]){

// // 位置交换

// var temp = arr[i];

// arr[i] = arr[j];

// arr[j] = temp;

// }

// }

// }

// 数组的操作方法：

// 数组的合并：\*.concat()

// 该方法的返回值会创建一个新的数组，不会改变原本的数组

// 克隆数组（开辟新的内存空间，存放该数组）

// var arr1 = [1,2,3,4];

// var arr2 = [5,6,7,8];

// var arr = arr1.concat(arr2);

// var a = [].concat(arr1);

// 数组的截取：\*.slice(startIndex,endIndex)

// 该方法的返回值：截取的数组

// 不会改变原本的数组

// 1、省略参数：默认截取全部

// 2、只写一个参数，代表起始下标：如果为正数，从起始下标开始，截取剩余的全部；如果是负数，反向截取。

// 3、两个参数都不省略，起始下标，结束下标，截取时，包含起始下标对应的元素，不包含结束下标对应的元素。

// 数组的增删改查：\*.splice(startIndex,length,ele1,ele2,ele3...)

// 如果括号里面写一个数字表示从第几个开始截取后面的所有

// 该方法的返回值是截取的部分（数组），该方法会改变原本的数组

// var arr = [1,2,3,4,5,6,7,8,9];

// arr.sort(function(){return Math.random() > 0.5})

// arr.splice(arr.indexOf(5) + 1,0,666)

// var arr = [1,2,3,4,5,6,7,8,9];

// arr.splice(0);

// arr = [];

// 数组的遍历方法：

// arr.forEach(function(v,i,arr){})

// var arr = [1,2,3,4,5,6,7,8,9];

// arr.forEach(function(v,i,a){

// var num=0;

// console.log(v,i,a);

// });

// var str = "abcdefghijk";

// Array.prototype.forEach.call(str,function(v,i){

// console.log(v,i);

// });

// call:借用某个对象下的方法，使用另外一个对象替换掉当前对象，call方法里的第一个参数是另外一个对象，后面的若干个参数会传入到借用方法里

// var obj1 = {

// name: "张三",

// sayName: function(a,b){

// console.log(a + this.name + b);

// }

// }

// var obj2 = {

// name: "李四"

// }

// obj1.sayName.call(obj2,"Hello,","你长得好帅！");

// 元素的处理方法

// \*.map(function(v,i){})

// var arr = [1,2,3,4,5,6,7];

// var a = arr.map(function(v){

// return v == 6 ? v + "66" : v + 10;

// });

// 数组的过滤方法：\*.filter(function(){})

// var arr = [1,2,3,4,5,6,7,88,777,666];

// var a = arr.filter(function(v){

// return v > 6 && v < 777;

// });

// 数组的克隆方法：\*.copyWithin(替换元素的开始位置,startIndex,endIndex)

// startIndex-endIndex：复制的元素

// 数组的归并方法：

// 数组元素的查找方法：

// 数组的替换方法：

// 数组中元素的检测方法：

// 数组元素的包含方法：

// 数组的归并方法：累加器

// \*.reduce(function(prev,ele){},0)

// \*.reduceRight(function(prev,ele){},0)

// var arr = [1,2,3,4,5,6,[7,8,9]];

// var sum = arr.reduceRight(function(prev,ele){

// console.log(ele);

// // return prev \* ele;

// },1);

// 二维数组处理成一维数组：

// var tempArr = [];

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

// // if(Array.isArray(arr[i])){

// // tempArr.push(...arr[i]);

// // }else{

// // tempArr.push(arr[i]);

// // }

// tempArr = tempArr.concat(arr[i]);

// }

// 数组元素的查找方法：

// 查找值：\*.find(function(v,i){})

// 查找下标：\*.findIndex(function(v,i){})

// var arr = [1,2,3,4,5,6];

// var val = arr.find(function(v,i){

// return i == 3;

// });

// var index = arr.findIndex(function(v,i){

// return v == 3;

// });

// 数组的替换方法：

// \*.fill(替换的值,startIndex,endIndex);

// var arr = [1,2,3,4,5,6];

// arr.fill("Hello",2,3);

// 数组中元素的检测方法：返回值 true || false

// 检测每一个元素：\*.every(function(v,i){})

// 检测某些元素：\*.some(function(v,i){})

// var arr = [0,1,2,3,4,5,6];

// var b1 = arr.every(function(v,i){

// return v >= 1;

// })

// var b2 = arr.some(function(v,i){

// return v >= 6;

// })

// 数组元素的包含方法：true || false

// \*.includes(元素)

// 函数的概念：是由事件驱动的或者当被调用执行可重复使用的代码块

// 代码块：由一条以上的多条语句组合在一起

// btn.onclick = function(){

// alert(1);

// }

// 函数声明和函数表达式共用的关键字：function

// 函数对应的构造函数：Function

// 函数的创建：

// 函数声明：function 函数名(形参){函数体}

// 函数表达式（匿名函数）：

// var 变量名称 = function(形参){}

// var 变量名称 = new Function(形参)

// ES6新增：箭头函数

// var 变量名称 = ()=>{}

// var fn = num=>console.log(num);

// 函数的执行：

// 变量名||函数名(实参)

// 参数：实际参数、形式参数

// 1、实际参数出现在函数调用执行时

// 2、形式参数出现在函数封装时

// 3、形式参数的值只能来自于实际参数，实际参数的值不能来自于形式参数

// 4、形式参数和实际参数一一对应，数量上不用一一对应

// 5、形式参数对应的没有实际参数，形式参数的值undefined

// 6、所有的形式参数都是局部变量

// 7、形式参数不能是一个常数

// 函数的返回值：return

// 1、return返回的是函数的执行结果

// 2、中断函数体

// 3、return是一条语句

// 4、return只能出现在函数体内

// 5、一个函数体内可以出现多个return，但只会执行一个return

// 6、return可以返回任何数据类型

// 函数声明：

// var a = b = c = 20;

// function fn(a,b,c,d){

// a = b = c = 10;

// console.log(a + b + c);

// }

// fn(a,b,c,4,5);

// var a = 10;

// function fn(a,undefined){

// a = 20;

// // window.a = 30;

// console.log(window,undefined == window.undefined);

// }

// fn();

// console.log(undefined);

// function fn(arr,undefined){

// window.a = 10;

// arr.length = 0;

// }

// var arr1 = [1,2,3];

// fn(arr1,undefined);

// console.log(arr1.length);

// function fn(a,b,c){

// for(var i=0;i<10;i++){

// if(i == 5) return i;

// }

// console.log("输出吗？");

// }

// var num = fn(1,2,3);

// console.log(num);

// 作业：如何把顾子杰提出的问题给弄死

// function fn(){

// return function(){

// return function(){

// return fn();

// }

// }

// }

// fn()()()

// 函数声明（存储）：函数可以在任何时候、任何地方封装，也可以在任何时候，任何地方调用执行

// 函数声明是存储在堆内存的

//

fn();

function fn(){

console.log(1);

}

// 函数表达式（匿名函数）：函数表达式是存储在栈内存的

// var click = function(){

// alert(1);

// }

// btn.onclick = click;

// new关键字实例化构造函数的方式创建匿名函数

// new Function(param1,param2)

// param1、param2都是字符串

// param1代表的是形式参数的集合 "a,b"

// param2代表的是函数体 "return a + b;"

// var fn = new Function("a,b,c","return a + b + c;");

// btn.onclick = function(){

// // if(!btn.open){

// // box.style.background = "hotpink";

// // btn.innerHTML = "关灯";

// // }else{

// // box.style.background = "skyblue";

// // btn.innerHTML = "开灯";

// // }

// // btn.open = !btn.open;

// // box.style.background = !btn.open ? "hotpink" : "skyblue";

// // btn.innerHTML = !btn.open ? "关灯" : "开灯";

// // btn.open = !btn.open;

// (box.style.background = !btn.open ? "hotpink" : "skyblue") && (btn.innerHTML = !btn.open ? "关灯" : "开灯") && (btn.open = !btn.open);

// }

// btn.onclick = new Function("",`(box.style.background = !btn.open ? "hotpink" : "skyblue") && (btn.innerHTML = !btn.open ? "关灯" : "开灯") && (btn.open = !btn.open)`);

// 箭头函数：

// var fn = ()=>{

// alert(123);

// }

// btn.onclick = e=>alert(1223);alert(1);

// 自我执行函数（立即执行函数）：模拟块级作用域

// (function(){})()

// (function(){

// alert(1);

// })()

// ES6新增特性：let、const让js存在了块级作用域

// if(true){

// (function(){

// var color = "red";

// })()

// }

// console.log(color);

// (function(){

// if(true){

// }else{

// }

// })()

var lis = ul.getElementsByTagName("li");

// for(let i=0;i<lis.length;i++){

// (function(index){

// lis[index].onclick = function(){

// alert(index);

// }

// })(i)

// // lis[i].onclick = (function(index){

// // return function(){

// // alert(index);

// // }

// // })(i)

// lis[i].onclick = function(){

// alert(i);

// }

// }

// 函数重载（函数声明）：

// var i = 1;

// fn(); // 3

// function fn(){

// i++;

// console.log(i);

// }

// fn(); // 3

// function fn(){

// i += 2;

// console.log(i);

// }

// fn(); // 3

// 函数声明存储在堆内存，函数表达式存储在栈内存，查找值时，先在栈内存中查找，在栈内存中没找到（考虑未定义的值），然后再去堆内存中查找

// var fn;

// console.log(fn);

// fn(); // 2

// fn = function(){

// console.log(1);

// }

// function fn(){

// console.log(2);

// }

// console.log(fn); //function(){console.log(1);}

// fn(); // 1

// fn = 3;

// fn = 3;

// var fn;

// console.log(fn);

// fn(); //

// fn = function(){

// console.log(1);

// }

// function fn(){

// console.log(2);

// }

// fn = 3;

// console.log(fn);

// var fn = function(){}

// function fn(){}

// 封装函数：

// 原材料（实际参数）==》进厂（形式参数）==》加工（函数体）==》出厂（返回值）

// function sort(tempArr){

// tempArr = [...tempArr];

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

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

// if(tempArr[i] > tempArr[j]){

// var temp = tempArr[i];

// tempArr[i] = tempArr[j];

// tempArr[j] = temp;

// }

// }

// }

// return tempArr;

// }

// var arr = [1,2,3,4,5,6,7,8,9];

// arr.sort(function(){

// return Math.random() < 0.5;

// });

// var a = sort(arr);

// var arr1 = [2131,4141,99,6564,8612,3,321,76];

// var b = sort(arr1);

// 封装一个函数：实现简单的数学运算

// function calc(num1,num2,oprate){

// return eval(num1 + oprate + num2);

// }

// 数组求最大最小值

// function fn(arr,flag){}

// fn([12,888,2,1321,3,21313,543],false)

// 统计一个字符串中某个字符出现的个数

// 封装去重函数：

// 封装一个函数：从前向后查找指定数组中某个元素首次出现的位置

第七天

// 封装一个函数：从前向后查找指定数组中某个元素首次出现的位置

// function index(arr,ele){

// var pos = null;

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

// // pos = arr[i] == ele ? i : -1;

// // if(arr[i] == ele) break;

// if(arr[i] == ele){

// pos = i;

// break;

// }else{

// pos = -1;

// }

// }

// return pos;

// }

<!-- 怎样避免内存泄露

1）减少不必要的全局变量，或者生命周期较长的对象，及时对无用的数据进行垃圾回收；

2）注意程序逻辑，避免“死循环”之类的 ；

3）避免创建过多的对象 原则：不用了的东西要及时归还 -->

// 作用域：变量和方法的一个执行环境，考虑上下文

// ES6之前，只存在函数作用域；ES6之后，既存在函数作用域，也存在块级作用域。

// var a = 10;

// function fn(){

// console.log(a);

// var a = 20;

// }

// fn();

// 函数执行完毕，会把函数体内的局部变量和方法销毁掉

// 作用域链：查找值的顺序

// 从当前作用域一级一级的向上（上一级作用域）查找

// 从当前查找的位置，向上查找，如果向上没有找到，去形式参数里查找，形式参数里也不存在，考虑当前作用域的变量声明提升，如果当前作用域不存在变量声明提升，则去上一级作用域中查找，从函数执行的位置向上查找，以此类推....

// if(true){

// let color = "red";

// }

// color = 10;

// var color;

// console.log(color);

// function fn(){

// // var color = "red";

// console.log(color);

// // var color = "blue";

// }

// fn();

// var color = "pink";

// var color = "blue";

// function fn(){

// // console.log(color + "1"); // pink

// window.color = "red";

// let color = "green";

// (function(){

// console.log(color + "2"); // green

// color = "五颜六色";

// var color = "orange";

// })("yellow")

// console.log(color + "3"); // （不显示）

// }

// fn("pink");

// console.log(color + "4"); // red

// var color = "red";

// function fn(){

// function fn1(){

// window.color = "blue";

// function fn2(){

// }

// }

// fn1();

// }

// fn();

// console.log(color);

// this:肯定是一个对象

// 包含它的函数作为方法被调用执行所属的对象，如果没找到，则是window对象。（不适用于箭头函数，箭头函数里的this指向的是window对象）

// 1、包含它的函数

// 2、作为方法被调用执行

// 3、所属的对象

// 4、没有找到是window

// function fn(){

// console.log(this);

// }

// window.fn();

// // btn.onclick = function(){

// console.log(this);

// }

// var obj = {

// fn: function(){

// console.log(this);

// }

// }

// obj.fn();

// btn.onclick = (function(){

// console.log(this);

// return function(){

// console.log(this);

// }

// })()

// var obj = {

// fn: function(){

// console.log(this);

// }

// };

// var f = obj.fn;

// function fun(ff){

// arguments[0]();

// }

// fun(f);

// var obj1 = {

// name: "对象1",

// sayName: function(){

// console.log(this);

// }

// }

// var obj2 = {

// name: "对象2"

// }

// // 改变了this指向

// var f = obj1.sayName;

// // obj1.sayName.call(obj2);

// new f()

// function Animal(){

// this.name = "Cat";

// }

// function Cat(){

// Animal.call(this);

// console.log(this);

// }

// new Cat();

// var obj = {

// name: "张三",

// fun: function(){

// // console.log(this.name);

// this.name = "王五";

// // console.log(this);

// return (()=>{

// console.log(this);

// })()

// },

// fun1: ()=>{

// this.name = "李四";

// // console.log(this.name);

// return function(){

// console.log(this);

// }()

// }

// }

// obj.fun();

// obj.fun1();

// console.log(name);

// 改变this指向：apply、call、bind

// apply、call：执行结果是借用方法的返回值

// call：返回值是一个新的函数

// var obj = {

// name: "张三"

// }

// function f(){

// console.log(this);

// }

// var ff = f.bind(obj);

// f();

// ff();

//

// function fn(){

// console.log(this);

// }

// fn(1,2,3,4,5,6,7);

// console.log( fn.bind(window) == fn);

// fn.bind(new String("fdsaghfdsja"))();

// fn();

// 函数里的隐藏对象:this arguments

// arguments：是一个类似于数组的对象，是所有实参的集合

// function fn(a,b,c){

// // console.log(Object.prototype.toString.call(arguments));

// // console.log( Array.from(arguments).slice(1,4) );

// // console.log( [].slice.call(arguments,1,4) );

// // console.log( [].slice.apply(arguments,[1,4]) );

// // arguments[0] = 11;

// // console.log(a);

// b = 22;

// console.log(arguments[1]);

// }

// fn(1,2,3,4,5,6);

// 闭包：一个函数有权访问另外一个函数中局部变量或者方法（函数中包含函数）

// 作用域中存在作用域

// if(true){

// let a = 10;

// (function(){

// var c = a + 20;

// })()

// }

// var arr = [];

// for(let i=0;i<10;i++){

// arr[i] = function(){

// alert(i);

// }

// }

// arr[0]();

// function fn(a){

// return function (b){

// return function (c){

// return a + b + c;

// }

// }

// }

// fn("Hello,")("world!")("你长的好高！！！");

// Hello,world!你长的好高！！！

// box.onclick = function(color){

// return function(){

// this.style.background = color;

// }

// }("pink")

// Array.prototype.forEach = function forEach(){

// }

// var obj1 = {

// fun: function(){

// console.log(this);

// (()=>{

// console.log(this);

// })();

// }

// }

// obj1.fun();

// Array.prototype.each = function each(){

// // console.log(this);

// var \_this = this;

// // fn();

// (function(){

// console.log(this, \_this, "1");

// })()

// }

// var f = ()=>{

// console.log(this);

// }

// var arr = [1,2,3,4,5,6];

// arr.each();

// f();

// var obj = {

// a: [7,8,9,10,11,12,13]

// };

// function fn(arr,b){

// arr.forEach((v)=>{

// arr.length = 3;

// console.log(v,b);

// })

// }

// var f = fn.bind(obj,[1,2,3,4,5,6],"Hello,world!")

// f();

// 高阶函数：函数作为方法的实参传入

// sort forEach....

// 递归：函数调用函数本身

Array.prototype.each = function each(fn){

this.i || (this.i = 0);

for(;this.i<this.length;this.i++){

if(Object.prototype.toString.call(this[this.i]) == "[object Array]"){

// 是数组，再使用each函数遍历

// this[this.i].each(fn);

arguments.callee.call(this[this.i],fn);

}else{

// 非数组

if(fn) fn(this[this.i],this.i);

}

}

}

var arr = [1,2,[3,[4,5],6],[7,[8,9,[10,11]]]];

// var sum = 0;

// var c = 0;

var a = [];

arr.each(function(v,i){

// sum += v;

// c++;

a.push(v);

});

// var aver = sum/c;

// 函数里的隐藏对象：arguments

// arguments.callee：指向的函数本身

第八天

// 字符串：凡是被单引号或者双引号引起来的内容都是字符串

// 字符串对应的构造函数：String

// 字符串的创建：

// 直接量：var str = "abc"

// new关键字实例化对象的方式创建字符串：var strObj = new String(123)1231231331321113331211

// 字符串的长度属性：length只读属性

// 字符串的方法：

// 静态方法：String.fromCharCode(97,98,99)

// 原型方法：

// 字符串的转换方法：该方法支持正则表达式

// \*.split(分隔符)

// var str = "好好学习,天天吃鱼！！！";

// console.log(str.split(",",1));

// JSON.stringify() 对象转json字符串

// JSON.parse() json字符串转对象

// var obj = {

// name: "张三",

// age: 18

// }

// console.log(JSON.stringify(obj));

// 转换大小写：

// \*.toUpperCase()

// \*.toLowerCase()

// 封装函数：作业："get-element-by-id" 驼峰命名 "getElementById"

// 字符转unicode编码值："abcd".charCodeAt(index)

// 判断字符串以什么结尾和结束：

// \*.startsWith(判断的字符或者字符串)

// \*.endsWith(判断的字符或者字符串)

// 字符串的位置方法：

// \*.indexOf(字符或者字符串)

// \*.lastIndexOf(字符或者字符串)

// \*.search(字符或者字符串或者正则表达式)

// var str = "ff4ds51f23dsa1f3d2sa1fdsa";

// console.log(str.search(/\d+/));

// 根据位置查找字符：\*.charAt(index)

// 读取字符：str[index]不支持IE8以下浏览器

// 操作方法：

// 字符串拼接：\*.concat(数据)

// 字符串的截取：都不会改变原本的字符串

// \*.slice(startIndex,endIndex)

// \*.substr(startIndex,length)

// \*.substring(startIndex,endIndex) 不支持负数

// 替换方法：

// 不会改变原本的字符串

// 只替换第一个匹配字符或者字符串

// 返回值是替换之后的字符串

// \*.replace(正则或者待替换的字符或者字符串,替换的字符或者字符串或者函数)

// var str = "at,bat,cat,eat,fat,hat";

// var str = "18603820238";

// var exp = /^1[3|4|5|6|7|8][0-9]{9}$/;

// 正则下的test方法：true false

// console.log(exp.test(str));

// 分组：()

// var a = str.replace(/^1[3|4|5|6|7|8][0-9]([0-9]{4})[0-9]{4}$/,function($1,$2){

// return $1.replace($2,"\*\*\*\*");

// });

// 包含方法：\*.includes(字符或者字符串) true || false

// var str = " ab cd ";

// 忽略空格：

// 忽略左右：\*.trim()

// 忽略左边：\*.trimLeft()

// 忽略右边：\*.trimRight()

// 只忽略中间：

// String.prototype.trimCenter = function(){}

// var str = " a b c d ";

// var s = str.replace(/ /g,function($1,$2){

// $2 = "";

// return $2;

// })

// var str = "f4d5saf23dsa1f23dsa1f23dsa";

// var arr = str.match(/\d+/g);

// 封装函数：忽略所有空格，并把该方法挂载到字符串的构造函数的原型对象上

// var str = "fdsfa ds";

// str.trimAll();

// function trimAll(str){}

// trimAll(str);

// var str = "f4ds3af3AsdaZf3d\_.s1af3dsa131fd3saf13";

// console.log(str.split(new RegExp("[a-z]+")));

// 正则表达式的创建：对应的构造函数RegExp

// 直接量：/ /

// new关键字创建正则表达式：new RegExp(正则,修饰符)

// \d \D /\d+/

// \w 字母数字\_ \W

// [a-zA-Z] 元字符

// 修饰符：gim global ignore multiple

// 数学对象：保存的是数学公式和数学信息

// 内置的静态对象

// 绝对值：Math.abs()

// 最大值：Math.max()

// 最小值：Math.min()

// 向上取整：Math.ceil()

// 向下取整：Math.floor()

// 四舍五入取整：Math.round()

// 开方：Math.sqrt()

// 次方：Math.pow()

// 封装函数：求任意范围内的水仙花数，放到数组里

// 153：1\*1\*1 + 5\*5\*5 + 3\*3\*3 == 153

// 随机数：Math.random() 0 - 1之间的浮点数 （0/1都不包含）

// 0 - 10之间的随机整数

// Math.random()\*10 0 - 10

// for(var i=0;i<10000;i++){

// console.log(Math.floor(Math.random()\*11));

// }

// Math.floor(Math.random()\*11)

// 10 - 20之间的随机整数

// 10 + Math.floor(Math.random()\*11)

// ~~Math.random()\*11 + 10 // 0 - 10 10 - 20

// min - max 之间的随机整数

// 公式：~~Math.random()\*(max - min + 1) + min

// 封装函数：求任意范围的n个不重复的随机整数

function randDigits(min,max,n){

// 容器：随机数字

var digits = [];

// 如果数组里的数字不足6个，则添加

while(digits.length < n){

// 得到随机数字

var randDigit = ~~(Math.random()\*(max - min + 1) + min);

// 如果随机数字在容器中未出现，则添加

if(!digits.includes(randDigit)){

digits.push(randDigit);

}

}

return digits;

}

randDigits(1,33,6);

// 作业：双色球

// 红球：1-33 6 不重复 排序（升序）

// 篮球：1-16 1

第九天

// 日期时间对象：Date

// 日期时间对象是以特定的时间段（时间戳）来保存日期时间的，从1970年向前和向后各推285616年。UTC世界协调时间、世界统一时间、格林尼治时间

// 实力化日期时间对象：得到的是格式化好的日期时间

// var date = new Date();

// 获取当前时间

// date.toLocaleTimeString();

// "下午2:04:21"

// date.toLocaleDateString();

// "2018/7/2"

// 如何把日期时间对象转为时间戳：

// 1、方式一

// console.log(Date.parse()); // 精确到秒

// 2、方式二

// console.log(date.getTime()); // 精确到毫秒的

// 3、方式三

// 把实力化对象转换为数字Number(date) // 精确到毫秒

// 4、方式四

// console.log(date.valueOf()); // 精确到毫秒的

// Date.UTC(年,月,日,时,分,秒,毫秒) 转为格林尼治时间

// 日期时间的设置：

// new Date(年,月,日,时,分,秒,毫秒)

// 如果设置日期时间，至少需要两个参数，年月，后面参数省略时，日期默认是当前月的1号，时分秒默认00

// 月的取值范围：0 - 11

// 日期的取值范围：1 - 31

// 小时的取值范围：0 - 23

// 分钟和秒：0 - 59

// 毫秒的：0 - 999

// 星期的取值范围：0 - 6

// 时间戳：

// 一天：

// 一个星期：

// 一个小时：

// 一分钟：

// 一秒钟：

// 组件方法：日期时间实例对象

// var date = new Date()

// 获取年：date.getFullYear()

// 设置年：date.setFullYear()

// 获取月：date.getMonth()

// 设置月：date.setMonth()

// 获取日期：date.getDate()

// 设置日期：date.setDate()

// 获取小时：date.getHours()

// 设置小时：date.setHours()

// 获取分钟：date.getMinutes()

// 设置分钟：date.setMinutes()

// 获取秒：date.getSeconds()

// 设置秒：date.setSeconds()

// 获取毫秒：date.getMilliseconds()

// 设置毫秒：date.setMilliseconds()

// 获取星期：date.getDay()

// 以上所有方法都对应的有UTC方法，比如getUTCHours()

// 格林尼治时间与本地时间相差的分钟数：

// \*.getTimezoneOffset()

// 间歇调用：setInterval() 定时调用

// 该方法的返回值代表当前的定时器

// 语法结构：时间间隔是毫秒

// 1、setInterval(函数,时间间隔);

// 2、setInterval(字符串,时间间隔);

// 3、setInterval("函数(参数)",时间间隔);

// 清除间歇调用：clearInterval(定时器)

// var i = 0;

// var timer = setInterval(function(){

// console.log(++i);

// },1000);

// setInterval("console.log(++i)",1000);

// function f(num){

// console.log(num);

// }

// var timer = setInterval("f(1000)",1000);

// clearInterval(timer);

// function fn(){

// console.log(arguments);

// setInterval(function(){

// console.log(arguments);

// },1000);

// }

// fn(1,2,3,4,5)

第十天

// DOM：文档对象模型

// DOM描绘的是一个层次化的节点树

// 节点：都是对象

// 节点名称(nodeName)节点类型(nodeType)节点值(nodeValue)

// 文档节点 #document 9 null

// 元素节点 大写的标签名称 1 null

// 标签上的所有属性

// 属性节点 属性名称 2 属性值

// 文本节点 #text 3 文本内容

// 注释节点 #comment 8 注释内容

var nodes = box.childNodes;

// var eles = Array.from(nodes).filter(function(node){

// return node.nodeType == 1;

// });

// var eles = [].filter.call(nodes,function(node){

// return node.nodeType == 1;

// });

// var eles = [].filter.apply(nodes,[node=>node.nodeType == 1]);

// var f = node=>node.nodeType == 1;

// var fn = [].filter.bind(nodes,f);

// var eles = fn();

// 标签元素的属性：

// 标签上的属性：

// 获取标签元素上所有的属性：元素节点对象.attributes

// 获取属性值：元素节点对象.getAttribute(属性名称)

// 设置属性：元素节点对象.setAttribute(属性名称,属性值)

// 删除标签上的属性：元素节点对象.removeAttribute(属性名称)

// 固有属性：标签原本就存在的属性

// 自定义属性：自己定义的属性

// 对象上的属性：

// 固有属性：

// 自定义属性：

// box.setAttribute("style","width:100px;height:100px;background:pink");

// box.setAttribute("style","width:200px");

// box.className = "";

// box.removeAttribute("class");

// console.log(box.id);

// delete box.index;

// 元素节点对象的查找方法：

// 通过id选择器获取元素节点对象：

// 该方法不是元素下的方法，是文档的方法

// window.onload = function(){

// var box = document.getElementById("box");

// console.log(box);

// }

// console.log(document.all);

// 通过标签选择器获取元素节点对象：

// 该方法是元素下的方法

// var tags = document.getElementsByTagName("div");

// var p = document.getElementById("box").getElementsByTagName("p");

// 通过类选择器获取元素节点对象：

// 该方法有兼容，不能兼容IE8以下的浏览器（包括IE8）

// 该方法是元素下的方法

// var cls = document.getElementsByClassName("box");

// if(!document.getElementsByClassName){

// document.getElementsByClassName = function getElementsByClassName(clsName){

// var all = document.all;

// var arr = [];

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

// if(all[i].className == clsName){

// arr.push(all[i]);

// }

// }

// return arr;

// }

// }

// 通过name属性名称获取元素节点对象：

// 该方法不是元素下的方法

// 通过各种选择器获取一个元素节点对象

// 该方法是元素下的方法

// document.querySelector()

// 通过各种选择器获取多个个元素节点对象

// 该方法是元素下的方法

// document.querySelectorAll()

// if(true){

// let fn = function(){} // 内存泄漏

// }

// console.log(fn);

// function fn(){

// {

// let num = 10;

// console.log("aaa");

// }

// // console.log(num);

// }

// fn();

// {{{{

// console.log("aa");

// }}}}

// {

// let i = 10;

// {

// let i = 20;

// }

// }

// b

// {

// var i = 10;

// }

// console.log(i);

// DOM属性（对象上的属性）：

console.log([].concat(box))

box.setAttribute("style","width: 200px;height: 200px;background:pink;border: 7px solid skyblue;text-align:center;line-height:100px;overflow:scroll");

// document.body.scrollTop || document.documentElement.scrollTop

// DOM的操作方法：

// 创建元素节点对象：document.createElement(标签名称)

// 创建文本节点对象：document.createTextNode(文本内容)

// 追加到当前父节点的尾部：

// 父节点.appendChild(子节点)

// 父节点.append(子节点) 兼容

// 追加到当前父节点的头部：

// 父节点.prepend(子节点)

// 父节点中插入节点：

// 父节点.insertBefore(新节点,旧节点)

// 父节点后追加兄弟节点：

// 父节点.after(子节点)

// 父节点前追加兄弟节点：

// 父节点.before(子节点)

// 父节点中替换节点：

// 父节点.replaceChild(新节点,旧节点);

// 一个节点替换另外一个节点：

// 当前节点.replaceWith(替换节点)

// 父节点中指定删除节点：

// 父节点.removeChild(子节点)

// 删除当前节点：

// 当前节点.remove()

// 克隆节点：

// 被克隆的节点.cloneNode(true || false) true 深度克隆 false 浅克隆

// 碎片整理：

// document.createDocumentFragment()

var div = document.createElement("div");

var text = document.createTextNode("这是一个文本！！！");

div.innerHTML = text.nodeValue;

// document.body.appendChild(div);

document.body.prepend(div);

var p = document.createElement("p");

p.innerHTML = "This is a paragraph!!!";

document.body.insertBefore(p,div);

var div1 = document.createElement("div");

div1.id = "box";

div.after(div1);

var div2 = document.createElement("div");

div2.id = "box1";

div.before(div2);

var div3 = document.createElement("div");

div3.id = "box2";

document.body.replaceChild(div3,p);

div.replaceWith(p);

document.body.removeChild(p);

div1.appendChild(p);

div1.remove();

div2.appendChild(div1);

var clone = div2.cloneNode(true);

document.body.appendChild(clone);

var fragment = document.createDocumentFragment();

for(var i=0;i<10;i++){

var div = document.createElement("div");

fragment.appendChild(div);

}

document.body.appendChild(fragment);

// 苏志刚去云南买媳妇儿：

// 方式一：

// 苏志刚走着去，买了一个媳妇儿，背着回来了，过了两天，感觉不够用，又去了，又买了一个，.....，买了四个媳妇儿。7趟

// 方式二：

// 苏志刚开了个带斗的拖拉机，一次拉回来四个媳妇儿....

// 请问苏志刚同学怎么去比较舒服？

console.log(box.style.width);

// 获取样式使用最终值：都是最终使用的样式对象

// IE:元素.currentStyle

// 非IE:window.getComputedStyle(元素,null)

// 封装兼容函数：获取样式使用最终值

function getStyle(ele,attr){

return parseFloat( ele.currentStyle ? ele.currentStyle[attr] : window.getComputedStyle(ele,null)[attr] );

}

getStyle(box,"font-size");

第十一天

// 属性或方法 说明

// caption 保存着<caption>元素的引用;

// tBodies 保存着<tbody>元素的HTMLCollection集合;

// tFoot 保存着对<tfoot>元素的引用;

// tHead 保存着对<thead>元素的引用;

// rows 保存着对<tr>元素的HTMLCollection集合;

// createTHead() 创建<thead>元素,并返回引用;

// createTBody() 创建tbody

// createTFoot() 创建<tfoot>元素,并返回引用;

// createCaption() 创建<caption>元素,并返回引用;

// deleteTHead() 删除<thead>元素;

// deleteTFoot() 删除<tfoot>元素;

// deleteCaption() 删除<caption>元素;

// deleteRow(pos) 删除指定的行;

// insertRow(pos) 向rows集合中的指定位置插入一行;

// <tbody>元素添加的属性和方法

// deleteRow(pos) 删除指定位置的行;

// insertRow(pos) 向rows集合中的指定位置插入一行;

// <tr>元素添加的属性和方法

// cells 保存着<tr>元素中单元格的HTMLCollectioin集合;

// deleteCell(pos) 删除指定位置的单元格;

// insertCell(pos) 向cells集合的指定位置插入一个单元格,并返回引用

// HTMLDOM获取表格的<caption>

// alert(table.caption.innerHTML); // 获取caption的内容;

// PS:在一个表格中<thead>和<tfoot>是唯一的,只能有一个;

// 而<tbody>不是唯一的,可以是多个,这样导致最后返回的<thead>和<tfoot>是元素引用;而<tbody>是元素集合;

// 事件：事件是javascript跳动的心脏，是用户和浏览器之间交互的桥梁

// 事件流：事件执行的顺序

// 事件捕获：从最不具体的元素向内捕获 从外向里

// 目标元素：触发该事件的元素

// 事件冒泡：从最具体的元素向外冒泡 从里向外

// 事件分为：

// DOM0级事件：以元素节点对象的属性而存在，以on开头加上事件名称

// DOM0级事件的绑定：元素.on+事件名称 = function(){}

// DOM0级事件的删除：元素.on+事件名称 = null

// DOM2级事件：兼容

// 绑定：

// IE：元素.attachEvent("on"+事件名称,函数fn)

// 非IE：元素.addEventListener(事件名称,函数fn,事件捕获true || false)

// 删除：

// IE：元素.detachEvent("on"+事件名称,函数fn)

// 非IE：元素.removeEventListener(事件名称,函数fn,事件捕获true || false)

// 事件捕获（浏览器默认是禁止的）、事件冒泡：

// document.body.addEventListener("click",function(){

// alert(1);

// },true);

// btn.addEventListener("click",function(){

// alert(2);

// },true);

// 一个对象中包含两个属性：addEvent、removeEvent

// 封装兼容函数：该函数支持DOM0级事件、DOM2级事件绑定

// 封装兼容函数：该函数支持DOM0级事件、DOM2级事件删除

// 作业：把DOM0自己加进去

var $ = {

addEvent: function(ele,eventName,fn,dc){

ele.addEventListener ? ele.addEventListener(eventName,fn,dc) : ele.attachEvent("on" + eventName,fn,dc);

},

removeEvent: function(ele,eventName,fn,dc){

ele.removeEventListener ? ele.removeEventListener(eventName,fn,dc) : ele.detachEvent("on" + eventName,fn,dc);

}

}

// function fn(){

// alert(1);

// }

// $.addEvent(document.body,"click",fn);

// $.addEvent(btn,"click",function(){

// $.removeEvent(document.body,"click",fn);

// })

// DOM0级事件和DOM2级事件的区别：

// 同一元素同类型的事件DOM0级只能执行一个

// 同一元素同类型的事件DOM2级能执行多个

// btn.onclick = function(){

// alert(1);

// }

// btn.onclick = function(){

// alert(2);

// }

// btn.onclick = function(){

// alert(3);

// }

// function fn(){

// alert(1);

// }

// var fn1 = function(){

// alert(2);

// }

// var fn2 = \_=>alert(3);

// $.addEvent(btn,"click",fn);

// $.addEvent(btn,"click",fn1);

// $.addEvent(btn,"click",fn2);

// $.addEvent(btn1,"click",function(){

// $.removeEvent(btn,"click",fn1);

// });

// var fn = \_=>console.log(this,1);

// function fn(){

// console.log(this,2);

// }

// fn = function(){

// console.log(this,3);

// }

// $.addEvent(btn,"click",fn);

// 事件对象：包含有关当前事件的所有信息

// IE：window.event

// 非IE：元素.on+事件名称 = function(形参=》事件对象){}

// btn.onclick = function(event){

// var e = event || window.event;

// console.log(e);

// }

// $.addEvent(btn,"click",function(event){

// var e = event || window.event;

// console.log(e);

// });

// 事件冒泡：

// maxBox.onclick = function(){

// alert(1);

// }

// $.addEvent(maxBox,"click",function(evt){

// var e = evt || window.event;

// this.style.background = "hotpink";

// console.log(e);

// });

// midBox.onclick = function(){

// this.style.background = "aqua";

// }

// minBox.onclick = function(evt){

// var e = evt || window.event;

// e.stopPropagation ? e.stopPropagation() : e.cancelBubble = true;

// this.style.background = "skyblue";

// }

// 阻止事件冒泡：

// IE：事件对象.cancelBubble = true;

// 非IE：事件对象.stopPropagation()

// 事件委托：把同一类型的事件委托到其父辈元素（爸爸、爷爷......）上

// 目标元素：

// IE：事件对象.srcElement

// 非IE：事件对象.target

//封装一个支持DOM0+DOM2事件的对象

var $ = {

addEvent: function(ele,eventName,fn,dc){

if(ele.addEventListener){

ele.addEventListener(eventName,fn,dc)

}else if(ele.attachEvent){

ele.attachEvent("on" + eventName,fn,dc)

}else{

ele["on"+eventName]=fn;

}

},

removeEvent: function(ele,eventName,fn,dc){

if(ele.removeEventListener){

ele.removeEventListener(eventName,fn,dc)

}else if(ele.detachEvent){

ele.detachEvent("on" + eventName,fn,dc);

}else{

ele["on"+eventName]=null;

}

}

}

function fn(a,b,c){

return a+b+c;

}

// var $ = {

// addEvent: function(ele,nam,fn,dc){

// ele.addEventListener ? ele.addEventListener(nam,fn,dc) : ele.addEvent("on"+nam,fn,dc)

// },

// removeEvent:function(ele,nam,fn,dc){

// ele.removeEventListener ? ele.removeEventListener(nam,fn,dc) : ele.detachEvent("on"+nam,fn,dc)

// }

// }

// $.addEvent(maxBox,"click",function(e){

// var evt = e || window.event;

// var t = evt.target || evt.srcElement;

// // t是maxBox里面的每一个元素

// t.style.background = "black";

// })

// $.addEvent(maxBox,"click",(function(color){

// return function (evt){

// var e = evt || window.event;

// var t = e.target || e.srcElement;

// t.style.background = color;

// }

// })("limegreen"))

// 王鹏强作业：三元运算嵌套

$.addEvent(maxBox,"click",\_=>{

var e = \_ || window.event;

var t = e.target || e.srcElement;

var color = null;

switch(true){

case t.id == "maxBox":

color = !t.show ? "orange" : "pink";

break;

case t.id == "midBox":

color = !t.show ? "yellow" : "red";

break;

case t.id == "minBox":

color = !t.show ? "chocolate" : "blue";

break;

}

t.show = !t.show;

t.style.background = color;

});

// 阻止默认行为：

// IE：事件对象.returnValue = true（不阻止） || flase（阻止）

// 非IE：事件对象.preventDefault()

// 阻止表单的提交和链接的跳转：javascript:void(0)

a.onclick = function(evt){

var e = evt || window.event;

console.log(e);

e.preventDefault ? e.preventDefault() : e.returnValue = false;

}

// 事件类型

// 内联事件：事件作为标签的属性而存在

// 脚本事件：

// DOM0级

// DOM2级

// box.onclick = function(){}

// box.addEventListener("click",function(){})

console.log(history);

// 鼠标事件：

// click（单击）、dblclick（双击）、mousedown（摁下）、mouseup（弹起）、mouseover（滑入）、mouseout（滑出）、mouseenter（进入）、mouseleave（离开）、mousemove（移动）、wheel（滚轮）、mousewheel（鼠标滚轮）

// 滑入、滑出支持事件冒泡

// 进入、离开不支持事件冒泡

// box.onmouseover = function(){

// console.log("进去了");

// }

// box.onmouseenter = function(){

// console.log("进去了");

// }

// 判断滑轮是向上滚动还是向下滚动：

// wheelDelta如果大于0，向上滚动；否则，向下滚动

// box.onwheel = function(evt){

// var e = evt || window.event;

// console.log(e);

// }

// 事件类型：type

// 键盘事件：keydown、keyup、keypress

// 键值：事件对象.keyCode

// 事件的触发：

// keydown、keypress是在字符写入之前触发事件

// keyup是在字符写入之后触发

// 事件对象：

// 键值：

// keyup、keydown：不区分大小写、数字区分小键盘

// keypress：区分大小写、数字不区分小键盘

// 触发事件的键：

// keydown、keydown：键盘上所有的键都触发

// keypress：键盘上的功能键不触发

// area.onkeyup = function(e){

// console.log(e);

// }

// html事件：load、focus、blur、submit、change、scroll、resize

// txt.onfocus = function(){

// if(!this.show && this.value == "洗衣机..."){

// this.value = "";

// }

// this.show = true;

// }

// txt.onblur = function(){

// if(!this.show || this.value == "洗衣机..." || this.value == ""){

// this.value = "洗衣机...";

// }

// }

// form.onsubmit = function(){

// this.action = "javascript:void(0)";

// if(!/^1[3|4|5|6|7|8|9][0-9]{9}$/.test(phone.value)){

// alert("手机号输入有误...");

// return ;

// }

// this.action = "http://www.baidu.com";

// }

// sel.onchange = function(){

// console.log(this.selectedIndex,this.selectedOptions,this.options);

// // this.add(new Option("选项11"));

// this.remove(0);

// }

// window.onscroll = function(evt){

// var scrollTop = document.body.scrollTop || document.documentElement.scrollTop;

// if(scrollTop <= window.innerHeight){

// fx.style.display = "none";

// }else{

// fx.style.display = "block";

// }

// }

// window.onresize = function(){

// console.log("浏览器窗口大小发生变化时，触发该事件");

// }

<body>

<a href="demo1.html">demo1.html</a>

<button id="btn1">打开新窗口</button>

<button id="btn2">关闭新窗口</button>

<button id="btn3">关闭自身窗口</button>

<div id="tz">跳转</div>

<div id="box1">链接1</div>

<div style="height: 800px;"></div>

<div id="box2">链接2</div>

<div style="height: 800px;"></div>

<div id="box3">链接3</div>

<div style="height: 800px;"></div>

<script>

// BOM：浏览器对象模型

// closed：表名当前窗口的关闭状态 true（关闭） false（打开）

// close()：关闭窗口 该方法不能关闭非脚本打开的窗口

// open(url,窗口名称\_blank\_self\_parent\_top,混合参数,不记)：打开新窗口 该方法的返回值是对新窗口的引用

// btn1.onclick = function(){

// newWindow = open("http://www.baidu.com","","width=200,height=200,left=200,top=200");

// console.log(newWindow.closed);

// }

// btn2.onclick = function(){

// newWindow.close();

// console.log(newWindow.closed);

// }

// 关闭自身窗口:

// var browser = b.browser;

// btn3.onclick = function(){

// if(browser == "Internet Explorer"){

// window.close();

// }else{

// location.href = "about:blank";

// }

// }

// console.log(b.browser);

// history:保存的是有关当前网页的浏览历史记录

// window.history

// history.forward() 下一级浏览历史记录

// history.back() 上一级浏览历史记录

// history.go(0) 任意一级浏览历史记录

// location:

// href:当前文档的路径地址（url）

// 可读写 :url地址重定向

// location.href = decodeURI("file:///D:/javascript/1805A/%E7%AC%AC%E5%8D%81%E4%BA%8C%E5%A4%A9/demo1.html");

// tz.onclick = function(){

// location.href = "http://www.baidu.com";

// }

// hash:url地址锚 获取的是url地址中#之后的字符串

// box1.onclick = function(){

// location.href = "#box2";

// }

// search:获取的是url地址中?之后的字符串

// navigator:对象包含有关浏览器的信息

</script>

</body>

// ipt.focus();

// ipt.blur();

// setTimeout延时执行

// clearTimeout() 清除延时执行

// setTimeout(function(){

// alert(1);

// },0);

// setTimeout("alert(2)",10);

// setTimeout(fn,100);

// var fn1 = function(\_){

// alert(\_);

// };

// setTimeout("fn1(4)",1000);

// function fn(){

// alert(3);

// }

// setTimeout(function(){

// alert(2);

// },0);

// alert(1);

// var count = 10;

// var id = setInterval(function(){

// ipt.value = --count;

// if(count < 1){

// clearInterval(id);

// }

// },1000);

// 递归：

// function fn(num){

// var \_arg = arguments;

// ipt.value = --num;

// if(num != 0){

// setTimeout(function(){

// \_arg.callee(num);

// },1000);

// }

// }

// fn(10);

// for(let i=0;i<=10;i++){

// setTimeout(function(){

// ipt.value = (10 - i);

// },i\*1000);

// }

原型链

<script>

var str = new String(123);

console.log(str.\_\_proto\_\_ === str.constructor.prototype,str.constructor.prototype === String.prototype);

// Function 构造函数 new 实例化得到的F实例对象

// Function构造函数上的显式原型对象和隐式原型对象共用同一个地址指针

// 函数既是实例对象，也可以是构造函数

// 把函数当成是Function的实例对象：函数的显式原型对象是从Function构造函数的隐式原型对象上来的

// 把函数当成构造函数：构造函数的隐式原型对象的显式原型对象来自于Object构造函数的隐式原型对象

// var F = new Function();

// var f = new F();

// \_\_proto\_\_ 存在于实例对象上，和创建该对象的构造函数的隐式原型对象共用同一个地址指针

// console.log(Function.prototype === F.\_\_proto\_\_);

// var f1 = Function.prototype;

// Function 隐式原型对象

// function fn(){

// }

// fn.\_\_proto\_\_

// fn.prototype

function Person(name){

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*重要部分（原型链）\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

var p1 = new Person("张三");

p1.\_\_proto\_\_ === Person.prototype

Person.prototype.\_\_proto\_\_ === Object.prototype

Object.prototype.\_\_proto\_\_ === null

Person.\_\_proto\_\_ === Function.prototype

Function.\_\_proto\_\_ === Function.prototype

Function.prototype.\_\_proto\_\_ === Object.prototype

Object.prototype.\_\_proto\_\_ === null

p1.constructor === Person

Person.constructor === Function

Function.constructor === Function

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*重要部分（原型链）\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

// p1.constructor.\_\_proto\_\_ === Function.prototype

// Person.constructor.prototype === Function.\_\_proto\_\_

// Object.constructor === Function

// String.constructor === Function

// Object.\_\_proto\_\_ === Function.prototype

// Object.\_\_proto\_\_.prototype === Function.prototype

继承

// 构造继承：子类Cat继承父类Animal里的所有构造属性

// function Animal(){

// this.name = "kitty";

// this.type = "波斯猫";

// }

// function Cat(){

// Animal(this;);

// }

// var c1 = new Cat();

// 原型继承：子类Cat继承父类Animal的原型对象

// function Animal(){}

// Animal.prototype.sleeping = function(){

// alert("正在睡觉...");

// }

// function Cat(){

// }

// Cat.prototype = Animal.prototype;

// Cat.prototype.eatting = function(){

// alert("正在吃饭...");

// }

// var c1 = new Cat();

// var c2 = new Cat();

// 实例继承：继承父类的实例对象

// function Animal(){

// this.name = "kitty";

// }

// function Cat(){

// var a = new Animal();

// a.age = 3;

// return a;

// }

// var c = new Cat();

// 拷贝继承：把父类实例对象中的拷贝给子类原型对象

// function Animal(){

// this.name = "kitty";

// this.age = 3;

// }

// Animal.prototype.type = "波斯猫";

// function Cat(){

// }

// (function(){

// var a = new Animal();

// for(var p in a){

// Cat.prototype[p] = a[p];

// }

// })()

// var c = new Cat();

// 寄生继承：

// function Cat(obj){

// var clone = Object.create(obj);

// clone.sayName = function(){

// console.log(this.name);

// }

// return clone;

// }

// var c = Cat({

// name: "kitty",

// age: 3,

// type: "波斯猫"

// });

// 寄生组合式继承：

// function Animal(){

// this.name = "kitty";

// this.type = "波斯猫";

// }

// Animal.prototype.sayName = function(){

// console.log(this.name);

// }

// function Cat(){

// Animal.call(this);

// this.age = 3;

// }

// Cat.prototype = new Animal();

// Cat.prototype.sayAge = function(){

// console.log(this.age);

// }

// var c = new Cat();

// 混合继承：增加复用性

// function Animal(){

// this.name = "kitty";

// this.type = "波斯猫";

// }

// Animal.prototype.sayName = function(){

// console.log(this.name);

// }

// function Cat(){

// Animal.call(this);

// this.age = 3;

// }

// (function(){

// var Super = function(){};

// Super.prototype = Animal.prototype;

// Cat.prototype = new Super();

// Cat.prototype.constructor = Cat;

// })()

// Cat.prototype.sayAge = function(){

// console.log(this.age);

// }

// var c = new Cat();

表格操作

3 属性或方法 说明

4 caption 保存着<caption>元素的引用;

5 tBodies 保存着<tbody>元素的HTMLCollection集合;

6 tFoot 保存着对<tfoot>元素的引用;

7 tHead 保存着对<thead>元素的引用;

8 rows 保存着对<tr>元素的HTMLCollection集合;

9 createTHead() 创建<thead>元素,并返回引用;

createTBody() 创建tbody

10 createTFoot() 创建<tfoot>元素,并返回引用;

11 createCaption() 创建<caption>元素,并返回引用;

12 deleteTHead() 删除<thead>元素;

13 deleteTFoot() 删除<tfoot>元素;

14 deleteCaption() 删除<caption>元素;

15 deleteRow(pos) 删除指定的行;

16 insertRow(pos) 向rows集合中的指定位置插入一行;

17

18 <tbody>元素添加的属性和方法

20 deleteRow(pos) 删除指定位置的行;

21 insertRow(pos) 向rows集合中的指定位置插入一行;

22

23 <tr>元素添加的属性和方法

24 cells 保存着<tr>元素中单元格的HTMLCollectioin集合;

25 deleteCell(pos) 删除指定位置的单元格;

26 insertCell(pos) 向cells集合的指定位置插入一个单元格,并返回引用

27

28 // HTMLDOM获取表格的<caption>

29 alert(table.caption.innerHTML); // 获取caption的内容;

30

31 // PS:在一个表格中<thead>和<tfoot>是唯一的,只能有一个;

32 // 而<tbody>不是唯一的,可以是多个,这样导致最后返回的<thead>和<tfoot>是元素引用;而<tbody>是元素集合;

// DOM属性（元素节点对象上的属性）：

console.log([].concat(box));

// attributes:标签上所有现有属性的集合

// childElementCount:父元素下所有子元素的数量

// childNodes:父元素下所有的子节点

// children:父元素下所有的子元素

// classList:元素节点对象上类名的集合

// className:当前元素的类名

// clientWidth、clientHeight:当前元素的宽高（不包含边框，包含填充）

// offsetWidth、offsetHeight:当前元素的宽高（包含边框线，包含填充）

// clientLeft、clientTop:当前元素的边框线的宽度

// offsetLeft、offsetTop:当前元素的左上偏移量

// contentEditable:当前元素是否可编辑

// draggable:当前元素是否可拖拽

// firstChild:父元素下的第一个子节点

// firstElementChild:父元素下的第一个子元素

// lastChild:父元素下的最后一个子节点

// lastElementChild:父元素下的最后一个子元素

// innerHTML:当前元素内的所有内容

// outerHTML:包含当前元素的所有内容

// innerText:当前元素内的所有的文本内容

// textContent:当前元素内的所有的文本内容

// previousSibling:上一个兄弟节点

// previousElementSibling:上一个兄弟元素节点

// nextSibling:下一个兄弟节点

// nextElementSibling:下一个兄弟元素节点

// 所有以on开头的都是DOM0级事件

// parentNode、parentElement:父节点和父元素

// scrollWidth、scrollHeight:当前元素的滚动宽高

// scrollLeft、scrollTop:当前元素的滚动左边距和滚动上边距

// style:当前元素的行内样式对象

// tagName:当前元素的标签名称（大写）

// DOM元素节点的操作方法:

// 一、创建元素节点对象的方法：document.createElement(标签名称)

// var p = document.createElement("p");

// p.innerHTML = "这是一个段落！";

// var comment = document.createComment("这是注释节点！");

// var text = document.createTextNode("这是一个文本节点！");

// var text1 = document.createTextNode("这是一个文本节点！");

// 二、追加元素节点对象的方法：

// 父元素.appendChild(子节点) 添加到当前父元素的尾部

// box.appendChild(comment);

// box.append(comment);

// box.firstElementChild.appendChild(text);

// box.appendChild(text);

// 三、追加到父元素的最前面的方法：

// 父元素.prepend(子元素)

// box.prepend(p);

// 四、删除元素节点对象的方法：

// 父元素.removeChild(子节点)

// box.removeChild(comment);

// 五、删除当前元素节点对象的方法：

// 元素.remove()

// box.remove();

// 六、插入元素节点对象的方法：

// 父元素.insertBefore(新节点,旧节点)

// box.insertBefore(p,comment);

// 七、在某个元素节点对象后插入一个元素节点对象：

// 兄弟元素.after(插入的元素)

// comment.after(p,document.createElement("a"));

// 八、替换元素节点对象的方法：

// 父元素.replace(新节点,旧节点)

// var div = document.createElement("div");

// div.innerHTML = "赵海龙睡着了！";

// box.replaceChild(div,comment);

// 九、一个元素节点对象替换另一个元素节点对象：

// 被替换的元素.replaceWith(替换的元素)

// box.replaceWith(div);

// 十、克隆元素节点对象的方法：

// 被克隆的节点.cloneNode(true|false)

// 如果不写，默认false

// var c = comment.cloneNode();

// var box1 = box.cloneNode(true);

// 十一、碎片整理：

// document.createDocument ()

// var fragment = document.createDocumentFragment();

// for(var i=0;i<5;i++){

// var div = document.createElement("div");

// div.innerHTML = 5 - i;

// fragment.prepend(div);

// }

// document.body.prepend(fragment);

// document.body.innerHTML = "这是一个body体！";

// options:当前下拉菜单下的所有的option选项

// selectedIndex:当前被选中选项的下标

// selectedOptions:当前被选中选项的集合

// DOM2级事件：同一元素，同类型的事件可以执行多次

// DOM2级事件的绑定：

// IE:元素.attachEvent("on"+事件名称,函数)

// 非IE:元素.addEventListener(事件名称,函数,事件捕获)

// DOM2级事件的删除：

// IE:元素.detachEvent("on"+事件名称,函数)

// 非IE:元素.removeEventListener(事件名称,函数,事件捕获)

// btn.onclick = function(){

// }

// btn.attachEvent("onclick",function(){

// alert(1);

// })

// btn.addEventListener("click",function(){

// alert(2);

// })

// 封装一个兼容函数：给元素绑定事件

function addEvent(ele,eventName,fn){

if(ele.attachEvent){

ele.attachEvent("on"+eventName,fn);

}else if(ele.addEventListener){

ele.addEventListener(eventName,fn);

}else{

ele["on"+eventName] = fn;

}

}

addEvent(btn,"click",f)

addEvent(btn,"click",f1)

addEvent(btn,"click",f2)

function f(){

alert(1);

}

function f1(){

alert(2);

}

function f2(){

alert(3);

}

// 封装一个兼容函数：给元素删除事件

function removeEvent(ele,eventName,fn){

if(ele.detachEvent){

ele.detachEvent("on"+eventName,fn);

}else if(ele.removeEventListener){

ele.removeEventListener(eventName,fn);

}else{

ele["on"+eventName] = null;

}

}

var f3 = function(){

// 删除btn上的点击事件

removeEvent(btn,"click",f);

removeEvent(btn,"click",f1);

// removeEvent(btn,"click",f2);

}

addEvent(btn2,"click",f3);

addEvent(btn1,"click",function(){

removeEvent(btn2,"click",f3)

});

// 递归：函数调用函数本身

// 必须有一个出口

// function fn(){

// fn();

// }

// fn();

// 阶乘：5\*4\*3\*2\*1

// num = 5 5\*4\*3\*2\*1

// function fn1(num){

// if(num == 1){ // 出口

// return 1;

// }else{

// return num\*arguments.callee(--num);

// }

// }

var events = {

addEvent: function(ele,eventName,fn){

if(ele.attachEvent){

ele.attachEvent("on" + eventName,fn);

}else if(ele.addEventListener){

ele.addEventListener(eventName,fn);

}else{

ele["on" + eventName] = fn;

}

},

removeEvent: function(ele,eventName,fn){

if(ele.detachEvent){

ele.detachEvent("on" + eventName,fn);

}else if(ele.removeEventListener){

ele.removeEventListener(eventName,fn);

}else{

ele["on" + eventName] = null;

}

}

}

// events.addEvent(window,"load",function(){

// alert(1);

// });

// events.addEvent(window,"load",function(){

// alert(2);

// });

// 不牵涉兼容性的阻止默认行为：javascript:void(0) 执行一个空事件

数组：

数组是对象中的一种，属于引用类型，存储在堆内存

数组对应的构造函数：Array

数组的创建：

直接量：[]

new关键字实例化构造函数的方式：new Array()

数组的长度属性：length（可读写）

var arr = [1,2,3,4,5]; // 紧密数组

数组的读取：数组[下标]

数组的动态添加以及修改：数组[下标] = 赋值

数组元素的删除：delete 删除之后，数组是稀疏数组

var arr = new Array(1,2,3,4,5,6);

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

var b = a;

b.length = 3;

console.log(a.length);

使用...克隆arr

var b = [...a];

多维数组：

var arr = []; // 北京站

arr[0] = []; // 昌平站

arr[0][0] = []; // 昌平线

arr[0][0][0] = ["西山口","十三陵","昌平"];

多维数组：

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

arr[4] arr[3][0] arr[3][2][0]

arr.length = 9;

数组的遍历：for、while、do...while

for...of遍历可迭代的对象

var arr = [

    1,2,3,4

];

for(var v of arr){

    console.log(v);

}

var arr = [1,2,3,4,5,6,7,[8,9,10]];

var sum = 0;

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

    if(Array.isArray(arr[i])){

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

            sum += arr[i][j];

        }

    }else{

        sum += arr[i];

        // console.log(arr[i]);

    }

}

数组的检测：

typeof ==> object

instanceof ==> [] instanceof Array

instanceof:检测一个实例对象是否属于一个函数对象

Array.isArray()

[].constructor.name ==> "Array"

Object.prototype.toString.call(待检测的数据) ==》"[Object Array]"

eval:让字符串作为程序去执行

数组的方法：

一、静态方法：

是否是数组：Array.isArray()

伪数组转数组：Array.from()

创建数组：Array.of()

function createArray(){

    return Array.of(...arguments);

}

二、原型方法：

数组的栈方法：

压栈：\*.push(ele1,ele2,...)

向数组的尾部添加一个或者多个元素，并返回数组新的长度

出栈：\*.pop()

从数组的尾部删除一个元素，并返回被删除的元素

数组的队列方法：

压队：\*.unshift(ele1,ele2,...)

向数组的头部添加一个或者多个元素，并返回数组新的长度

出队：\*.shift()

从数组的头部删除一个元素，并返回被删除的元素

数组的反转（不使用reverse方法）：

第一种

        var arr = [1,2,3,4];

        var str = [];

    for(var i=0;i<=arr.length-1;i++){

        str.unshift(arr[i]);

    }

        第二种

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

        var str = [];

for(var i=arr.length-1;i>=0;i--){

    str.push(arr[i]);

}

var tempArr = [];

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

    // 1 2 3 4

    tempArr.unshift(arr[i]);

}

for(var i=arr.length - 1;i>=0;i--){

    // 4 3 2 1

    tempArr.push(arr[i]);

}

不使用任何系统方法：

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

str[arr.length-1-i] = arr[i];

}

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

    tempArr[arr.length - 1 - i] = arr[i];

}

数组的转换方法：

数组转字符串：\*.toString() \*.join(分隔符)

字符串转数组：\*.split(分隔符)

统计字符串中每一个字符出现的次数：

{a:3,d:4...}

var str = "fdsghfjgsajfgja";

var obj = {};

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

    obj[str[i]] = str.split(str[i]).length - 1;

}

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

    var count = 0;

    for(var j=0;j<str.length;j++){

        if(str[i] == str[j]){

            count++;

        }

    }

    obj[str[i]] = count;

}

判断该字符串中是否存在某个字符或者字符串（使用split）

var str = "fdshakfjhdsdkajf"; // dsk

if(str.split("dsk").length - 1 == 0){

    alert("不存在");

}else{

    alert("存在");

}

数组的位置方法：如果未出现，返回值-1

从指定数组中从前向后查找某个元素首次出现的位置（下标）：\*.indexOf(元素)

    从指定数组中从前向后查找某个元素最后出现的位置（下标）：\*.lastIndexOf(元素)

数组的去重：

方式一：

var arr = [1,2,3,4,5,1,2,1,1,1,6,3,7];

var tempArr = [];

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

    // 判断tempArr中是否存在遍历的arr中的元素，如果不存在，则添加到tempArr中

    if(tempArr.indexOf(arr[i]) == -1){

        tempArr.push(arr[i]);

    }

}

方式二：

var index = 0;

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

    for(var j=i+1;j<arr.length;j++){

        if(arr[i] == arr[j]){

            j = ++i;

        }

    }

    tempArr[index++] = arr[i];

}

方式三：利用的是对象属性的唯一性

var obj = {};

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

    if(!(obj[arr[i]] in obj)){

        tempArr.push(arr[i]);

        obj[arr[i]] = arr[i];

    }

    if(obj[arr[i]] != 1){

        tempArr.push(arr[i]);

        obj[arr[i]] = 1;

    }

}

今天晚上的作业：如果数组中存在多个NaN，一个都不要

var arr = [1,2,3,4,1,4,2,3,NaN,6,5,NaN,7,9,6,NaN];

var str = [];

第一种方法

arr.join("").split(NaN).join("").split("")

第二种方法

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

    for(var j=i+1;j<arr.length;j++){

        if(arr[i]==arr[j] || Number.isNaN(arr[i]) == true){

            j=++i;

        }

    }

    str.push(arr[i]);

}

if(Number.isNaN(str[str.length-1])){

    str.splice(str.length-1,1);

}

console.log(str);

第三种方法

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

    if(!str.includes(arr[i])){

        if(!Number.isNaN(arr[i])){

            str.push(arr[i]);

        }

    }

}

str.sort(function(a,b){

    return a<b;

})

console.log(str);

数组的排序方法：

翻转：\*.reverse() str.split("").reverse().join("")

排序：\*.sort()

该方法的返回值是排序之后的数组，默认是升序（按照unicode编码值进行排序）；会改变原本的数组。（高阶函数）

var arr = [9,12,23,87,98,4,114,66];

// arr.sort();

arr.sort(function(x,y){

    return Math.random() > 0.5;

});

console.log(arr);

冒泡排序：（冒大泡、冒小泡）

15 18 9 6 7 33 88 i=0 j=1 2 3 4 5 6

18 15 9 6 7 33 88

18 15 9 6 7 33 88

18 15 9 6 7 33 88

18 15 9 6 7 33 88

33 15 9 6 7 18 88

88 15 9 6 7 18 33 i=1 j=2 3 4 5 6

.....

var arr = [15,18,9,6,7,33,88];

arr[0] arr[1]

var temp = arr[0];

arr[0] = arr[1];

arr[1] = temp;

arr[0] = arr[1];

arr[1] = arr[0];

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

    for(var j=i+1;j<arr.length;j++){

        if(arr[i] < arr[j]){

            // 位置交换

            var temp = arr[i];

            arr[i] = arr[j];

            arr[j] = temp;

        }

    }

}

数组的操作方法：

数组的合并：\*.concat()

该方法的返回值会创建一个新的数组，不会改变原本的数组

克隆数组（开辟新的内存空间，存放该数组）

var arr1 = [1,2,3,4];

var arr2 = [5,6,7,8];

var arr = arr1.concat(arr2);

var a = [].concat(arr1);

数组的截取：\*.slice(startIndex,endIndex)

该方法的返回值：截取的数组

不会改变原本的数组

1、省略参数：默认截取全部

2、只写一个参数，代表起始下标：如果为正数，从起始下标开始，截取剩余的全部；如果是负数，反向截取。

3、两个参数都不省略，起始下标，结束下标，截取时，包含起始下标对应的元素，不包含结束下标对应的元素。

数组的增删改查：\*.splice(startIndex,length,ele1,ele2,ele3...)

如果括号里面写一个数字表示从第几个开始截取后面的所有

该方法的返回值是截取的部分（数组），该方法会改变原本的数组

var arr = [1,2,3,4,5,6,7,8,9];

arr.sort(function(){return Math.random() > 0.5})

arr.splice(arr.indexOf(5) + 1,0,666)

var arr = [1,2,3,4,5,6,7,8,9];

arr.splice(0);

arr = [];

数组的遍历方法：

arr.forEach(function(v,i,arr){})

var arr = [1,2,3,4,5,6,7,8,9];

arr.forEach(function(v,i,a){

    var num=0;

    console.log(v,i,a);

});

var str = "abcdefghijk";

Array.prototype.forEach.call(str,function(v,i){

    console.log(v,i);

});

call:借用某个对象下的方法，使用另外一个对象替换掉当前对象，call方法里的第一个参数是另外一个对象，后面的若干个参数会传入到借用方法里

var obj1 = {

    name: "张三",

    sayName: function(a,b){

        console.log(a + this.name + b);

    }

}

var obj2 = {

    name: "李四"

}

obj1.sayName.call(obj2,"Hello,","你长得好帅！");

    元素的处理方法

\*.map(function(v,i){})

var arr = [1,2,3,4,5,6,7];

var a = arr.map(function(v){

    return v == 6 ? v + "66" : v + 10;

});

数组的过滤方法：\*.filter(function(){})

var arr = [1,2,3,4,5,6,7,88,777,666];

var a = arr.filter(function(v){

    return v > 6 && v < 777;

});

数组的克隆方法：\*.copyWithin(替换元素的开始位置,startIndex,endIndex)

startIndex-endIndex：复制的元素

数组的归并方法：

数组元素的查找方法：

数组的替换方法：

数组中元素的检测方法：

数组元素的包含方法：

数组的归并方法：累加器

\*.reduce(function(prev,ele){},0)

\*.reduceRight(function(prev,ele){},0)

var arr= [1,2,3,4,5,6,[7,8,9]];

var sum = arr.reduceRight(function(prev,ele){

    console.log(ele);

    // return prev \* ele;

},1);

二维数组处理成一维数组：

var tempArr = [];

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

    // if(Array.isArray(arr[i])){

    //  tempArr.push(...arr[i]);

    // }else{

    //  tempArr.push(arr[i]);

    // }

    tempArr = tempArr.concat(arr[i]);

}

数组元素的查找方法：

查找值：\*.find(function(v,i){})

查找下标：\*.findIndex(function(v,i){})

var arr = [1,2,3,4,5,6];

var val = arr.find(function(v,i){

    return i == 3;

});

var index = arr.findIndex(function(v,i){

    return v == 3;

});

数组的替换方法：

\*.fill(替换的值,startIndex,endIndex);

var arr = [1,2,3,4,5,6];

arr.fill("Hello",2,3);

数组中元素的检测方法：返回值 true || false

检测每一个元素：\*.every(function(v,i){})

检测某些元素：\*.some(function(v,i){})

var arr = [0,1,2,3,4,5,6];

var b1 = arr.every(function(v,i){

    return v >= 1;

})

var b2 = arr.some(function(v,i){

    return v >= 6;

})

数组元素的包含方法：true || false

\*.includes(元素)

// 数组的方法：

        // 一、静态方法：

        // 是否是数组：Array.isArray()

        // 伪数组转数组：Array.from()

        // 创建数组：Array.of()

        // function createArray(){

        //  return Array.of(...arguments);

        // }

// 向数组的尾部添加一个或者多个元素，并返回数组新的长度：\*.push(ele1,ele2....)

// 从数组的尾部删除一个元素，并返回被删除的元素：\*.pop()

// 从数组的头部删除一个元素，并返回被删除的元素：\*.shift()

// 向数组的头部添加一个或者多个元素，并返回数组新的长度：\*.unshift(ele1,....)

// 数组的翻转：\*.reverse()

// 位置方法（查找位置的）：\*.indexOf \*.lastIndexOf

// \*.findIndex(function(){})（高阶函数）

// 数组的合并：\*.concat(arr1,ele1....)

        // 数组的操作方法：

        // 数组的合并：\*.concat()

        // 该方法的返回值会创建一个新的数组，不会改变原本的数组

        // 克隆数组（开辟新的内存空间，存放该数组）

// 数组的截取、删除、添加、替换：\*.splice(startIndex,length,ele1,ele2....)

// 数组的截取：\*.slice(startIndex,endIndex)

// 数组的转换：\*.join(分隔符)

        // 字符串转数组：\*.split(分隔符)

// 数组的转换：\*.toString()

// 数组中是否包含该元素：\*.includes(ele)

// 数组的过滤：\*.filter(function(a){return 条件(a < 5)})

// 数组的遍历：\*.forEach(function(v,i){})

// 数组的归并方法：\*.reduce(function(prev,ele){},初始值) reduceRight

// 数组的排序，默认使用unicode编码值进行排序（高阶函数）：\*.sort(function(a,b){return a > b;})

// 数组的静态方法：判断是否是数组Array.isArray()

// 伪数组转数组:Array.from(伪数组)

// 数组元素的处理：\*.map(function(v){})

        // 数组的位置方法：如果未出现，返回值-1

        // 从指定数组中从前向后查找某个元素首次出现的位置（下标）：\*.indexOf(元素)

        // 从指定数组中从前向后查找某个元素最后出现的位置（下标）：\*.lastIndexOf(元素)

    // 排序：\*.sort()

        // 该方法的返回值是排序之后的数组，默认是升序（按照unicode编码值进行排序）；会改变原本的数组。（高阶函数）

        // var arr = [9,12,23,87,98,4,114,66];

        // // arr.sort();

\*.sort(function(x,y){

return x>y;

})

随机排序:

        // arr.sort(function(x,y){

        //  return Math.random() > 0.5;

        // });

        // console.log(arr);

        // 数组的遍历方法：

        // arr.forEach(function(v,i,arr){})

        // var arr = [1,2,3,4,5,6,7,8,9];

        // arr.forEach(function(v,i,a){

        //  var num=0;

        //  console.log(v,i,a);

        // });

        // var str = "abcdefghijk";

        // Array.prototype.forEach.call(str,function(v,i){

        //  console.log(v,i);

        // });

        // call:借用某个对象下的方法，使用另外一个对象替换掉当前对象，call方法里的第一个参数是另外一个对象，后面的若干个参数会传入到借用方法里

            // 元素的处理方法

        // \*.map(function(v,i){})

        // var arr = [1,2,3,4,5,6,7];

        // var a = arr.map(function(v){

        //  return v == 6 ? v + "66" : v + 10;

        // });

        // 数组的过滤方法：\*.filter(function(){})

        // var arr = [1,2,3,4,5,6,7,88,777,666];

        // var a = arr.filter(function(v){

        //  return v > 6 && v < 777;

        // });

        // 数组的克隆方法：\*.copyWithin(替换元素的开始位置,startIndex,endIndex)

        // startIndex-endIndex：复制的元素

        // 数组的归并方法：累加器

        // \*.reduce(function(prev,ele){},0)

        // \*.reduceRight(function(prev,ele){},0)

        // var arr = [1,2,3,4,5,6,[7,8,9]];

        // var sum = arr.reduceRight(function(prev,ele){

        //  console.log(ele);

        //  // return prev \* ele;

        // },1);

        // 二维数组处理成一维数组：

        // var tempArr = [];

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

        //  // if(Array.isArray(arr[i])){

        //  //  tempArr.push(...arr[i]);

        //  // }else{

        //  //  tempArr.push(arr[i]);

        //  // }

        //  tempArr = tempArr.concat(arr[i]);

        // }

        // 数组元素的查找方法：

        // 查找值：\*.find(function(v,i){})

        // 查找下标：\*.findIndex(function(v,i){})

        // var arr = [1,2,3,4,5,6];

        // var val = arr.find(function(v,i){

        //  return i == 3;

        // });

        // var index = arr.findIndex(function(v,i){

        //  return v == 3;

        // });

        // 数组的替换方法：

        // \*.fill(替换的值,startIndex,endIndex);

        // var arr = [1,2,3,4,5,6];

        // arr.fill("Hello",2,3);

        // 数组中元素的检测方法：返回值 true || false

        // 检测每一个元素：\*.every(function(v,i){})

        // 检测某些元素：\*.some(function(v,i){})

        // var arr = [0,1,2,3,4,5,6];

        // var b1 = arr.every(function(v,i){

        //  return v >= 1;

        // })

        // var b2 = arr.some(function(v,i){

        //  return v >= 6;

        // })

        // 数组元素的包含方法：true || false

        // \*.includes(元素)

alert(1);警告弹出框

confirm("输出的内容");确认系统能够弹出框

prompt("请输入你的成绩:","成绩范围在0-100之间");输入弹出框

alert(prompt("请输入信息:","成绩范围在 0-100之间"));

document.write("页面输出");

console.log(“控制台输出”);

变量声明提升: 提升到当前栈内存的最顶部

标识符的命名规则：变量名称。 常量名称 ，属性名称 ， 参数名称 ，函数名称 ，

1不能以数字开头，

2不能出现除了下划线和&以外的任何符号；、

3关键字不能作为标识符去使用，保留字不建议作为标识符去使用（不要去使用）

存在变量提升

// var

// 1、存在函数作用域，不存在块级作用域

// 2、可以重复声明

// 3、声明的变量会作为window对象的属性或者方法而存在

// 4、声明的变量存在声明提升

// 5、使用var声明的变量是局部变量，不适用var声明的变量是全局变量

常量得声明：const（Es6之后新增特性

不可变的量：常量一旦声明，其值不可改变

常量声明的同时，必须赋值

常量不能重复声明

常量不存在提升

常量存在块级作用域

常量不会做作为window对象下的属性而存在

数据类型：ES6之前 6种 ES6新增一种： symbol

基本类型的数据存储在栈内存

引用类型的存储在堆内存

五中简单的：（基本类型）

三种数据类型：

①字符串：凡是被单引号或者双引号引起来的都是字符串

字符串对应的构造函数（函数对象）String

new 实例化构造函数 得到实力对象

②数字：整数， 浮点数，正无穷Infinity， 负无穷Infinity，NaN，

数字对应的构造函数：Number

③布尔类型：boolean

true==>1 false==>0

两种特殊值：

undefined:

变量声明未负值，变量的值是undefined

对象下不存在该属性，获取该属性时是undefined

函数没有返回值，函数的执行结果是undefined

数组中不存在下标对应的元素时，获取该下标对应的元素时值是undefined

字符串下标对应的没有字符，undefined;

null:是一个特殊的空对象，在内存中不站位

var num =10;

num = null; 垃圾处理 手动清理垃圾

一种复杂的数据类型（引用类型）：就是对象（Object）

对象属性是一个键值对，键名和建值之间使用：隔开，属性之间使用，隔开

var obj {} 直接量方式创建对象

var obj = {

name:"名字",

sex:true,

age:10,

wife:null,

gf:"一个女的",

salary:undefined, } ;

直接量方式创建对象

读取对象上属性值：

方式一：

console.log(obj.gf);

方式二：

var gf ="gf";

console.log(obj[gf]);

修改属性值：

obj.gf = "别人";

obj["gf"] = "另一个人";

console.log(obj.gf);

动态添加属性：

obj.son = {}

obj.son.name = "张三";

obj["son"]["name"] = "王五";

删除队形下的属性：delete

delete obj.son;

delete obj["son"]

obj.sayName = ()=>{

console.log(this.name);

}

obj.sayName();

object string number undefined null true false symbol length

数据类型检测（2种语法结构

typeof:(操作符)

typeof:（待检测的变量或则常量或常数0"a" 2 true var num const

js弱类型，动态类型的语言：js中的变量类型是松散类型的

松散类型就是：变量本身没有类型，变量的类型来自于所附的值

检测结果有六种、了"string" "number" "bollean" "undefined" "object" "function"

检测结果都是字符串类型的.

数据类型的转化 ：

各种数据类型转换为字符串类型：

强转函数：String()

对象下的方法： \*.toString

出去数字的常熟，null ,undefined,其他的都能使用该方法进行转换

各种数据类型转换为数字类型：

强转数字：Number()

所有的数据中只有一下七个能转换为数字0：

0，-0，“0”，false, null,""，“ 任意多个空格”

如果带转换的数据对应的是数据，则转换为相对应的数字，否则转换为NaN

转换为整形数字：parseInt()

转换为浮点数字：parseFloat()

如果待转化的数据是数字类型，则转换整数或则浮点数

如果带转换的数据是字符串类型的数字，则转换为整数或则浮点数

如果带转换的数据是字符串类型的而且以数字开头，则转换为整数或则浮点数

出去以上三点之外的任何其他数据转换结果都是NaN

各种数据类型转换为布尔类型：

强转布尔类型：Boolean()

逻辑操作符号 ：逻辑非！

true=>1 false=>0

所有的函数 ，对象都是真值

所有数据中存在的假值 ：0 -0 null false NaN undefined " "

逻辑非！ 取反并把数据类型转换为你布尔类型

隐士类型的转换

// if("a"){

// alert(1)

// }

eval将字符串作为程序与运行

// var str = "1,2,3,4,5,6,7";

// var arr = eval("[" + str + "]");

// console.log(arr)

判断是不是NaN：isNaN(带判断的数据)

返回值：不是数字是true 是数字是false

隐士类型转换：先把带判断的数据使用Mumber强转函数函数转换为数字，然后在判断转换后的数字是否是NaN，

保留小数位：\*.toFixed(保留小数位)

\*必须是一个数字

和遵循四舍五入的原则

保留的位数在0-20之间或则0-100之间

默认情况下不保留小数位

该方法的返回值是保留在之后的数字，是字符串类型的

操作符 or 运算符

按条目划分：

一元运算符 || 一目运算符

只能处理一个操作数 的运算符叫做一元运算符

！ typeof

+ - 自动转型的功能

++自增 --自减

++ --配合着变量使用

变量值的交换：

1

// a=a-b;

// b=a+b;

// a=b-a;

2

// c=a;

// a=b;

// b=c;

3

// var [b,a] = [a,b];



// var {b,a} = {a:b,b:a}

二元运算符 || 双目运算符

处理两个操作数

+ - \* / % && ||

三元运算符 || 条件运算符

和if双分支功能是一样的

语法结构：判断条件? 结果1 ： 结果二；

// if(true){

// alert(1);

// }else{

// alert(2);

// };

// true ? alert(1) : alert(2);

操作符：&&逻辑与 || 逻辑或 !逻辑非

逻辑与 &&是找假值的，一旦找到假值则运算结果就是对应的那个假值，运算结束；如果运算到最后没有找到假值，则运算的结果就是最后一个

逻辑与或的使用

var month = prompt("请输入月份","月份取值范围1-12");

// var season = "请输入正确的月份";

// if(month == null){

// season = "请输入月份...";

// }else{

// if(month >= 1 && month <= 3){

// season = "春季";

// }else{

// if (month >= 4 && month <= 6) {

// season = "夏季";

// }else{

// if(month >= 7 && month <= 9){

// season = "秋季";

// }else{

// if(month >= 10 && month <=12){

// season = "冬季";

// }}}}}

// alert(season);

//

// var num = prompt("请输入一个整:");

// 判断该数字是否能被357之间人格一个整除

// if(num%3 ==0 || num%5 ==0 || num%7 ==0){

// alert(num);

// }

// var num = prompt("请输入一个整:");

// 判断该数字是否能被357之间人格一个整除

// if(!(num%3) || !(num%5) || !(num%7)){

// alert(num);

// }

NaN除了和字符串拼接，和其他数据执行数学运算的结果都是NaN

按功能划分运算符：

1算数运算符：

\* / % + -

+ 如果两边的操作数，有一个是字符串类型的，则起拼接作用；如果两边操作数都不是字符串类型的，则起相加作用，如果两边的操作数不是字符串，也不是数字，使用Number强转函数，转换为数字，在进行相加。

// 1数学运算相加

// 2字符串拼接

// 3正好

//

//求于：

// 1任何整数对1求于的结果是0

// 2任何小数对1求于的结果是小数部分

// 3任何数对0求于的结果NaN

// 4求于数小于被求于数结果是求于数本身

// 5求余数大于被求余数，结果是余数

//

//

// 2比较运算符: 优先级是从左向右的

// 凡是牵涉比较，其运算结果都是布尔类型的

// > < >= <= == === != !==

// 1两边都是数字，直接进行数字大小的比较

// 2两边的操作数有一个不是字符串类型的把两边的操作数，使用Number强转函数，转换为数字，再进行比较

// 3两边的操作数，都是字符串类型的，则比较的是unicode编码值

// 4NaN和任何数据比较，结果都是false，包括他本身

// 字符串转换为数字的方法："a".charCodeAt(0)

//

// ==比较的是值

// 1null除了和 undefined以及它本身之外的任何数据比较结果都是false

// 2undefined除了和 null以及它本身之外的任何数据比较结果都是false

// 3如果两边是同等类型的操作数，则直接进行值得 比较如果两边是不同等类型的，则使用Number强转函数，转换为数字再进行比较

// 4NaN和任何数据比较，结果都是false,包括它本身

// 5引用类型之间的比较，比较的是地址指针

//=== 全等比较，比较的是值和类型

//!=只要值不相同，结果就为真true

//!==只要值和类型中的一个不相同，结果为真true

// 3赋值运算符

// = += -= \*= /= %= <<= >== &= |=

// 赋值运算 的执行顺序是从右向左的

// += 等同于 = +

// 4逻辑运算符

// 逻辑与 逻辑或 逻辑非

// 5条件运算符

// 判断条件? 结果1：结果二;

// 6字符串运算符 + +=

// 7加性运算符 + -

// 8乘性运算符 \* / %

// 9小括号运算符 ()优先级是最高的

// 10中括号运算符[]数组

// 11花括号运算符{}对象

// 12逗号运算符 ，

// 13分好运算符 结束语;

// 14成员运算符 .的

// 15typeof运算符 检测变量类型

// 16instaceof运算符

// 检测一个实力对象是否属于一个函数对象： true ||false

// 例：new String("abc") instaceof String

// in:检测一个属性是否属于一个 对象

// 17反引号运算符

// `${变量}`

// 18...运算符

// 取出数组中的元素

// 例：

// function fn(){

// console.log(...arguments)

// }

// fn(1,2,3,4,5,)

//

// 19位移运算符（战略，慢慢学习）

// 20自增自减运算符

// ++ --

// 第四天上午

// 流程控制语句：if语句，switch语句

// if语句按照分支结构划分 ：

// 单分支：if(判断条件){代码块}

// 省略花括号，有且仅有一条语句属于其对应的代码块

// if(判断条件)代码块；

// 双分支（三元运算符）：if(判断条件){代码块1}else{代码块2}

// if(true) ;else ;

//

// 多分支：有多个判断条件，但只会执行一个代码块

// 是双分支嵌套的变形

// 如果分支结构超过3个，不能再使用多分支结果 ，使用switch语句

// if(判断条件1){

// 代码块

// }else if(判断条件2){

// 代码块

// }else if(判断条件3){

// 代码块

// }else{

// 代码块

// }

//

// if(判断条件1){

// 代码块

// }else{

// if(判断条件2){

// 代码块

// }else{

// if(判断条件3){

// 代码块

// }else{

// 代码块

// }}}

//switch语句中出现的关键字：

//switch . case . break . default

// . 默认

switch(表达式.可以写值，也可以写表达式){

case 判断条件：

代码块

break;

case 判断条件：

代码块

break;

case 判断条件：

代码块

break;

default:

默认代码块

}

表达式与判断条件是全等比较的

成绩输入判断：

var score = prompt("请输入成绩:","成绩范围0-100之间");

var res = null;

var s = parseFloat(score);

switch(true){

case score == null || score == "":

res = "请输入成绩:";

break;

case isNaN(score) || isNaN(s) :

res = "请输入数字:";

break;

case score < 0 || score >100 :

res = "数字输入有误";

break;

case score == 0:

res = "你是猪吗?";

break;

case score > 0 && score <60:

res = "好好学吧";

break;

case score > 0 && score <90:

res = "再接再厉";

break;

case score > 0 && score <100:

res = "不错";

break;

default:

res ="哈哈哈";

}

alert(res);

// 第四天下午

// 循环语句： for,while,do...while,for...in...

// 循环概念：在有限的条件内。重复不断地去执行一件事情（代码块==》循环体）

// 代码块：由一条或则多条语句组合起来的

// for循环：是一种先判断，后运行（循环体）的循环语句

// 循环与可能一次都不执行，循环可能一直都执行（死循环）

语法：

for(初始值；判断条件；步长更新){

循环体

}

循环的执行顺序：

for(1,2,3){

4

先执行初始值，执行判断，判断条件成立，执行循环体，循环体之行结束，执行步长更新，执行判断条件，判断条件成立，。。。。。。如果判断条件不成立，循环执行结束

1初始值==》2成立 ==》4循环体==》3步长更新

2成立 ==》4循环体==》3步长更新

2成立 ==》4循环体==》3步长更新

2成立 ==》4循环体==》3步长更新

。。。

2 （不成立）结束

初始值值执行一次

判断条件执行的次数比循环体执行的次数多一次

循环体执行的次数和步长更新执行的次数相同

// 把初始值，判断条件，步长更新全部省略：死循环

//死循环：

//for(;;){}

//for(;true;){}

//for(var i=0;i<=10;i--){}

//for(var i=10;i<=0;i++){}

//不循环

//for(;false;){}

//

//输出1-100之间的奇数

// for(i=1;i<=100;i+=2){

// console.log(i);

// }

// 输出18-1000的奇数

// 输出18-1000能同时被357整除的

// var min = 18;

// var max = 1000;

// for(i=min;i<=max;i++){

// if(i%3 == 0 && i%5 == 0 && i%7 == 0){

// console.log(i);

// } }

//

// 第五天 上午

// for循环：使用在循环次数固定时

// 循环的遍历：一一列举:反转下面的字符转

str[0]

str[1]

str[2]

str.length

\*/

// var str = "abcdefghigklmnopqrstuvwxiy";

// for(i=str.length - 1;i>=0;i--){

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

// }

//

// 拼接下面的字符串

// var newStr ="";

// var str = "abcdefghigklmnopqrstuvwxiy";

// for(i=str.length - 1;i>=0;i--){

// newStr = newStr + str[i]

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

// }

//

// 求和

// var str ="48446464854968461";

// var sum = 0;

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

// sum = sum + str[i]\*1;

// console.log(sum);

// // sum += str[i]\*1;

// }

// .求平均数

// var str ="123456789s";

// var sum = 0;

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

// sum = sum + str[i]\*1;

// }

// console.log(sum/str.length);

// str.length(是最大长度，从1开始数)

// [str.length](是下表最大长度，从0开始数)

//

//

// var str = "abcd";

// for(var i=0;str[i];i++){

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

// }

// 也可以遍历，如果能提取就提取abcd如果不能的时候就提取undefined

// 嵌套循环

// 1嵌套循环从从外层循环的第一次循环开始执行

// 2内层循环是作为外层循环的循环体而存在

// 3外层循环的下次循环是在外层循环的循环体执行结束后

// 4整个嵌套循环是在外层循环执行结束后结束

/\*

for(1;2;3){

for(1-1;2-2;3-3){

4-1

}

}

\*/

// var jsq =0; //计数器

// for(var i=0;i<10;i++){

// for(var j=8;j>=0;j--){

// console.log(i + j);

// ++jsq;

// }

// }

// console.log(jsq);

// while循环 ：也是一种先判断后执行的循环语句

// while循环：循环次数不固定时，使用while循环

//

while(判断条件){

循环体

把字符串中的a换成\*号

var str = "asdfdsaasdfasasaaaaa";

字符串的替换：字符串.replace(待替换的字符或则字符串，替换的字符或则字符串)

只替换第一个匹配字符或则字符串

该方法不会改变原本的字符串

判断当前字符串中是否存在该字符：字符串.includes(字符或则字符串) 返回值 包含true 不包含false

//

while(str.includes("a")){

str = str.replace("a","\*");

}

console.log(str);

// do...while...:是一种先运行，后判断的循环语句（几乎用不到，作为了解）

// do{

// console.log("会输出，只会输出一次");

// }while(false)

// 三四换成一二

// var str = "一二一二一二三四、三四、三四、三四、";

// do{

// str = str.replace("三四","一二");

// }while(str.includes("三四"))

// for...in...：遍历对象：

//

/\*

for(属性 in 对象){}

for(var p in obj){}

\*/

// var obj = {

// name: 1,

// age: 2,

// sex: 3

// }

// // in:判断一个属性是否属于一个对象

// for(var p in obj){

// console.log(p,obj[p]);

// }

//

// 遍历windowu对象

// for(var p in window){

// if(p == "alert")

// // window[p]("弹出来");

// eval(p + "('弹出来')");

// }

跳转语句：

// break:中断当前循环

// 1break出现在switch语句中

// 2出现在循环里

// contionue:跳出当前循环，继续下一次循环

//

//

// for(var i=0;i<10;i++){

// console.log(i);

// if(i == 5 || i == 1) break;

// }

// console.log(i);

//

// for(var i=0;i<10;i++){

// for(var j=8;j>=0;j--){

// if(i%j == parseInt(null)){

// break;}} }

// console.log(i,j);

// 判断条件如果有两个 ，以后边的为准。

// for(var i=0,j=5;i<=10,j>=0;i++,j--){

// console.log(i,j);

// 逗号运算后，取最后一个值

// var i = (1,2,3,4,5);

// var i = 0;

// while(i<=30){

// i += 2;

// if(i%2 == 0){

// continue;

// }

// console.log(i);

// }

//

// 第六天：数组

// 排序

// var str = "slkjhgfdoiuytrewmnbvcx";

// 排序：冒泡 冒大泡 冒小泡

数组：数组时对象中的一种，数组也是一种复杂数据量类型。是数据的无序排序的集合，，数组使用逗号隔开；

//

//

// 数组的构造函数是Array

数组创建的两种方式：

1：直接量方式：[]

2:new关键字方式创建数组实例对象：new Array()

// 直接量方式创建数组

// var arr = [1,2,3,];

// arr.name = "数组";

// 动态的给数组添加元素：

// arr[arr.length] = 4;

// 修改或则替换数组中元素的值

// arr[1] = 22;

// 删除

// delete arr["0"];

// delete arr.name;

// delete["name"];

// 数组元素的查找 ：下标查找

// 数组的长度属性是一个可读写的属性

// arr.length = 100;

// console.log(arr.length,arr[-1]);

// 自变量方式创建数组

/\*var num1 = 5;

var num2 = 6;

var num3 = 7;

var num4 = 8;

var arr1 = [num1,num2,num3,num4];

// 数组的克隆

var arr = new Array(...arr1);

arr.length = 9;

var arr2 = new Array(arr.length);

\*/

// 数组的遍历：

// var arr = [1,2,3,4,5,6];

// var sum = 0;

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

// if(typeof arr[i] == "number" && arr[i].toString() !="NaN"){

// sum += arr[i];

// }

// }

// 多维数组：

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

//

// 遍历：

// var arr = [[1,2],3,[4,5,6],[7,8,9]];

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

// var ele = arr[i];

// if(ele instanceof Array){

// for(var j=0;j<ele.length;j++){

// console.log(ele[j]);

// }

// }else{

// console.log(ele);

// }

// }

// 数组的检测：

// constructor:该属性指向的是 实力对象对应的构造函数

// instaceof:检测一个实力对象是否属于一个函数对象(构造函数)

// 借用对象Object构造函数下的toString方法：Object.prototype.toString.call(数据)

//排序

// var arr = [34,2,6,4566,2885,81,525];

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

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

// if(arr[i] > arr[j]){

// var temp = arr[i];

// arr[i] = arr[j];

// arr[j] = temp;

// }

// }

// }

// console.log(arr);

// 数组的转换方法：

数组转字符串方法（数组下的方法）： \*.toString() \*. jion(分隔符)

字符串转数组的方法（字符串下的方法）：\*.split(分隔符,length)

检测字符串中j出现的次数

var str = "sdfghjkjhgfdsdfghdfghhsdfghdfg";

str.split("j").length - 1

\*/

// 伪数组转数组的方法：Array.from(维数组)

数组的栈方法：

压栈：\*.push(ele,ele2...) \*是数组

向数组的尾部添加一个或则多个元素，并返回数组新的长度

出栈：\*.pop() \*是 数组

从数组的尾部删除一个元素，并返回被被删除的元素

数组的队列方法：

压队： \*.unshift(ele1,ele2...) \*是数组

向数组的头部添加一个或则多个元素，并返回数组的新的长度

出队：\*.shift() \*是数组

从数组的头部删除一个元素，并返回被被删除的元素

数组的位置方法：

\*.indexOf(元素):在指定数组中从前向后查找，某个元素首次出现的位置（下标）,

如果指定数组中没有出现该元素，则结果是-1

\*.lastIndexOf(元素):在指定数组中从前向后查找，某个元素最后出现的位置（下标）,

如果指定数组中没有出现该元素，则结果是-1

数组的包含方法：\*.includes(元素) 返回值 true false

组的排序方法：

反转方法：\*.reverse()

排序方法：\*.sort() 按照第一个数字的大小比较

\*.sort(function(x-y){

return x>y || x-y

} )

x>y x-y正序 x<y x+y 倒序

// 数组的操作方法：

// 数组的合并：\*数组.concat(数组)

// 转换为数组与前面的数组合并

// [].concat.(Array.from(document.all))

// 数组的克隆:[].concat(被克隆的数组)

// 例：

// var arr = [1,2,3,]

// [].concat(arr)

// var arr1 = [].concat(arr)

// 数组的截取：\*slice(startIndex,endIndex)

// 开始下标 结束下标

// 整数是前往后，负数是后往前，截取的值不包括结束下标对应的值

// 该方法不会改变原本的数组

// 数组的增删改：splice(startIndex,length,ele1,ele2,ele3......)

// 删除开始下标 删除长度 添加的元素

// 该方法会改变原本的数组

// // 数组的遍历方法：\*.forEach(function(value,index){})

// 固定值 不固定值

// 代表值 代表下标

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

// arr.forEach(function(v,i){

// console.log(i + ":" + v);

// })

// 借用方法：

// 获取所有标签元素：

// var all = document.all;

// Array.from(all).forEach(function(ele,i){

// console.log(ele);

// })

// [].forEach.apply(all,[function(ele,i){

// console.log(ele);

// }]);

// [].forEach.call(all,function(ele,i){

// console.log(ele);

// });

// Array.prototype.forEach.call(all,function(ele,i){

// console.log(ele);

// });

// 数组的过滤方法：\*.filter(fucntion(v){})

// 例子：

// var arr = [1,2,3,4,5,6,7,8,9,10];

// var a = arr.filter(function(v){

// return v > 2 && v < 5 || v > 8 && v < 10;

// });

// 函数的概念：

// 函数是由事件驱动的或则当前被调用执行时可重复使用的代码块

// 函数的创建：关键字 function 对应的构造函数Function

// 函数声明：（命名函数）：function 函数名（形参，实参）{函数体}

// 函数表达式：（匿名函数）：var 变量名 = function（形参）{}

// 函数执行:函数名（实参）变量名（实参）

// 返回值：return

// 返回值：

// 1函数的执行结果

// 2中断函数体

// 3一个函数中可以出现多个return，但只会执行一个return

// 4return的返回值可以是任何数据类型的，也可以是一个函数

//

// 参数：形式参数（形参），实际参数（实参）

// 1形参和实参 是一一对应的（指的不是数量上的完全相同），

// 形参的值来自实参。实参的值不能来自于形参

// 2形参不能是一个常数，如果形参的值没有实参，形参的值是undefined

// 3形参的值只能来自于实参，所有的形参都是局部变量

// function f(a(是10),形参，形参，形参){

// console.log(形参);

// }

// var a = 10;

// f(实参，实参，实参)

// 封装函数的顺序：

// 原材料（待处理的数据）》进场（参数）》

// 加工（函数体）》出厂（返回值）

// 去重：

// function unique(a){

// for(var i=0;i<a.length;i++)

// if(tempArr.includes(a[i])){

// tempArr.push(a[i]);

// }

// return tempArr;

// }

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

// unique(arr);

// 分装函数的排序：

// function sort(a){

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

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

// if(a[i] > a[j]){

// var temp =a[i];

// a[i] = a[j];

// a[j] = temp;

// }

// }

// }

// return a;

// }

// var arr = [2,5,6,3,4,8,9];

// 按照参数和返回值划分：

// 1：无参数,无返回值函数

// 2：无参数，有返回值

// 3：有参数，有返回值

// 封装克隆函数：

// function sort(a){

// var tempArr = [...a];

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

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

// if(tempArr[i] > tempArr[j]){

// var temp = tempArr[i];

// tempArr[i] = tempArr[j];

// tempArr[j] = temp;

// }

// }

// }

// return tempArr;

// }

// var arr = [6,9,12,8,3];

// sort(arr)

// 命名函数（函数声明）可以再任何时候任何地方封装，也可以在任何时候，任何地方调用执行

// 命名函数是存储在堆内存的，命名函数属于引用类型

// 匿名函数存储在栈内存的

// var f = function(){

// console.log(1);

// }

// console.log(f);

// 如果占内存中存在，则先去栈内存种查找，如果栈内存不存在，去堆内存中查找，如果堆内存中也不存在，考虑变量声明提升，如果不存在声明提升，报错

// function f(){

// console.log(1);

// }

// var f = 10;

// f();

// var f = function(){

// console.log(2);

// }

// 匿名函数的创建

// var f = function(){}

// var f = new Function(param1,param2);

// param1,param2是两个字符串，第一个字符串是形参，第二个字符串是函数体

// var param1 = "x,y";

// var param2 = "return x + y;";

// var f = new Function(param1,param2);

// 箭头函数：()=>{}

// var a = (a,b)=>{

// console.log(a + b);

// }

// a = function(){}

// a(1,2);

// box.onclick = ()=>{

// }

// box.onclick = function(){

// }

// box.onclick = new Function();

// function f(){}

// box.onclick = f;

// 函数的重载（函数声明）：

// 函数的重载叫做函数的重新载入

// var f = function(){}

// console.log(f);

// f = new Funtion();

// f = ()=>{}

// var i = 1;

// f(); // 3

// function f(){

// i++;

// console.log(i);

// }

// f(); // 5

// function f(){

// i += 2;

// console.log(i);

// }

// f(); // 7

// 函数里的隐藏对象：arguments 所有实参的集合

// 是一个类似于数组的一个对象（伪数组），是所有实参的集合

// 封装任意若干个实参的求平均数

// function average(){

// var sum = 0;

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

// sum += arguments[i];

// }

// return (sum/arguments.length).toFixed(2);

// }

// average(1,2,3,4,45,7);

// average(9,8,7,6,5,4,3,2,1);

// apply:

// 借用某个对象下的方法，使用另外一个对象替换当前对象，下一个参数是一个数组，把数组中元素作为借用方法的参数传入进去

// function f(){

// // console.log(Array.from(arguments).slice(0,4));

// console.log(Array.prototype.slice.apply(arguments,[0,4]));

// }

// f(1,2,3,4,5,6,7,8,9)

// var obj = {

// n: "张三",

// sayName: function(str,str1){

// console.log(this.n,str,str1);

// }

// }

// var obj1 = {

// n: "李四"

// }

// obj.sayName.apply(obj1,["借了之后，你是爷爷","孙子,你好"])

// var obj = {

// num: 10,

// f: ()=>{

// console.log(this)

// }

// }

// obj.f();

// 1、封装任意若干个实参的求平均数

// function average(){

// var sum = 0;

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

// sum += arguments[i];

// }

// return (sum/arguments.length).toFixed(2);

// }

// average(1,2,3,4,5,);

// 2、封装字符串中某个字符出现的个数的函数。

// function b(){}

// var s=0;

// var str = "sdfghjafgdhjskafjjjjjjjjjjjjjghja";

// while(str.includes("j")){

// str = str.replace("j","1");

// ++s;

// }

// console.log(s);

// 3、封装数组中的位置方法indexOf的功能函数

//

// 4、封装通过id名称获取元素节点对象的方法

//

// 5、封装通过className获取元素节点对象的方法

//

// 6、封装数组的排序方法

// function sort(a){

// var tempArr = [...a];

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

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

// if(tempArr[i] < tempArr[j]){

// var temp = tempArr[i];

// tempArr[i] = tempArr[j];

// tempArr[j] = temp;

// }

// } b

// }

// return tempArr;

// }

// var arr = [6,9,12,8,3];

// sort(arr)

// 7、封装数组的去重方法

// function unique(a){

// var tempArr = [];

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

// if(!tempArr.includes(a[i])){

// <div id="box" class="box"></div>

// <div class="box"></div>

// <div class="box"></div>

// <div class="box"></div>

// <div class="box"></div>

// <div class="box"></div>

// 封装字符串中某个字符出现的个数函数

// function count(str,char){

// // 方式一：

// // var c = 0;

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

// // if(str[i] == char){

// // c++;

// // }

// // }

// // return c;

// // 方式二：

// // return str.split(char).length - 1;

// // 方式三：

// var c = 0;

// str.replace(new RegExp(char,"g"),function(){

// c++;

// });

// return c;

// }

// count("fdghasugfhjasd","f");

// var char = "f":

// var exp = //g;

// 参数：形式参数 实际参数

// 1、形式参数函数声明或者函数表达式里

// 2、实际参数出现在函数调用执行时

// 3、形式参数的值，只能来自于实际参数

// 4、所有的形式参数都是局部变量

// var a = 10;

// function f(aa,b,c,d){

// console.log(a);

// }

// f(1,2,3,4)

// indexOf方法的功能：在指定数组中从前向后查找某个元素首次出现的位置

// function indexOf(arr,ele){

// var pos = null; // 找到的位置

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

// // if(ele === arr[i]){

// // pos = i;

// // break;

// // }else{

// // pos = -1;

// // }

// pos = ele === arr[i] ? i : -1;

// if(ele === arr[i]) break;

// }

// return pos;

// }

// 通过id选择器获取元素节点对象：

// function getId(idName){

// var all = document.all;

// var e = null;

// // // [].forEach(function(){})

// // [].forEach.call(all,function(ele,index){

// // if(ele.id == idName){

// // e = ele;

// // }

// // })

// // return e;

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

// if(all[i].id == idName){

// e = all[i];

// }

// }

// return e;

// }

// getId("box");

// getId("box1");

// 通过类选择器获取元素节点对象：

// document.getElementsByClassName(类名)

// if(!document.getElementsByClassName){

// // 不支持该方法时，给document对象添加getElementsByClassName属性，赋值只一个匿名函数，参数是类名

// document.getElementsByClassName = function(clsName){

// // 获取文档中所有的元素节点对象

// var all = document.all;

// // 如果找到类名是传入进来的类名，相同时，把该元素放入到集合里

// var arr = [];

// // 从文档中所有的元素节点对象里筛选出，类名是参数的那些元素

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

// if(all[i].className == clsName){

// // 放入集合

// arr.push(all[i]);

// }

// }

// // 循环执行完毕，筛选完成

// return arr;

// }

// }

// document.getElementsByClassName("box");

// 最大值，最小值 flag为真，求最小值 否则，最大值

// function computed(arr,flag){

// var val = arr[0];

// // var condition = null;

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

// // condition = flag ? val > arr[i] : val < arr[i];

// // if(flag){

// // condition = val > arr[i];

// // }else{

// // condition = val < arr[i];

// // }

// if(eval(val + flag + arr[i])){

// val = arr[i];

// }

// }

// return val;

// }

// computed([1,2,3,4,5,6,345234,23],">")

// 封装一个函数：执行简单运算 + - \* / %

// function computed(num1,num2,oprate){

// return eval(num1 + oprate + num2);

// }

// computed(1,2,"+");

// 作用域：ES6之前，是只存在函数作用域，不存在块级作用域。ES6新增特性：let、const 声明的量存在块级作用域

// function f(){

// var a = b = 10;

// window.a = 20;

// console.log(a);

// }

// f();

// console.log(a,b);

// 函数在执行完毕之后，会销毁函数体内的局部变量和方法，不会销毁全局变量

// function f(){

// b = 10;

// }

// function ff(){

// b = 20;

// }

// console.log(b);

// var a = 10;

// f(); // a = 30

// function f(a){

// a = 20;

// window.a = 30;

// // console.log(a); // 20

// }

// console.log(a); // 10

// function f(color){

// function ff(color){

// window.color = "red";

// color = "green";

// }

// ff(color);

// color = "pink";

// console.log(color,window.color);

// }

// console.log(color);

// var color = "blue";

// f("yellow"); // window.color

// console.log(color);

// 闭包：一个函数有权访问另外一个函数的局部变量或者方法。（函数中包含函数）作用域中还有作用域

// 上下文的执行环境

// var color = "blue";

// function f(color){

// window.ff = function(){

// console.log(color);

// }

// window.color = "pink";

// // window.ff();

// }

// window.color = "red";

// f(color);

// // console.log(color); // "pink"

// var a = window.ff;

// a();

// var color = "blue";

// // ff是属于window对象的

// // ff();

// function f(){

// var ff = function(){

// console.log(color);

// }

// function fff(color){

// color = "green";

// ff()

// }

// fff();

// }

// f("red")

// var color = "red";

// function ff(color){

// f = function(){

// console.log(color);

// }

// color = "green";

// f();

// }

// ff()

// f();

// console.log(window.a);

// var a = 20;

// window.a = 10;

// 作用域链（闭包）：一级一级的向上查找（上下文的执行环境）

// 1、找当前函数的栈内存，从当前位置向上查找，如果上面不存在，再去形式参数内查找，形式参数内不存在，当前函数体的变量声明提升

// 2、向上查找不存在，形参不存在，变量声明提升不存在，去上一级作用域中查找（从函数执行的位置）

// var color = "red";

// var f = function(){

// console.log(color);

// // color = "blue";

// }

// function ff(color){

// var f = function(){

// console.log(color);

// }

// color = "skyblue";

// fff("pink");

// function fff(color){

// color = "blue";

// f(); // color

// }

// // f();

// }

// ff("yellow");

// var name = "张三";

// var obj = {

// sayName: function(){

// // name = "李四";

// console.log(name);

// }

// }

// var ff = obj.sayName;

// function f(ff){

// var name = "李四";

// ff();

// // (function(){

// // console.log(name);

// // })()

// }

// f(ff)

// this指针：this肯定是一个对象

// this是包含它的函数，作为方法被调用执行时所属的对象；如果找不到，则是window对象。（对于箭头函数来说，是不正确的）

// function f(){

// console.log(this);

// }

// window.f();

// btn.onclick = function(){

// console.log(this);

// }

// this.name = "李四";

// var obj = {

// name: "张三",

// sayName: function(){

// console.log(this);

// },

// sayName1: (name)=>{

// this.name = name;

// console.log(this.name);

// }

// }

// // obj.sayName();

// // obj.sayName1();

// function Fn(){

// this.name = "王五"; // 构造函数的构造属性

// // obj.sayName();

// // obj.sayName1();

// // console.log(this);

// }

// // Fn();

// // new Fn();

// var fn = new Fn(); // 实例化对象

// // obj.sayName.call(fn);

// obj.sayName1.call(fn,"赵六");

// console.log(name);

// var name = "张三";

// var obj = {

// name: "李四",

// sayName: function(name){

// this.name = name;

// // this.name = name;

// console.log(this.name);

// },

// sayName1: (name)=>{

// // window.name = "马汉";

// this.name = name; // window.name

// console.log(this.name);

// }

// }

// function Fn(){

// this.name = "王五";

// }

// var fn = new Fn();

// // obj.sayName.call(fn,"王朝");

// // console.log(obj.name);

// obj.sayName1.call(fn,"赵六")

// console.log(this.name);

// btn.onclick = (function(){

// this.a = "a";

// // return function(){

// // alert(this.a);

// // }

// return ()=>{

// this.b = 5;

// consolethis.b

// alert(this.a);

// }

// })()

// console.log(this.a);

// var f = function(){

// console.log(this);

// }

// function demo(fn){

// fn = {

// innerFn: function(){

// console.log(this);

// }

// }

// arguments[0].innerFn();

// }

// demo(f)

// function f(num,num1,num2){

// num1 = 654;

// // arguments[1] = 654;

// console.log(arguments[1]);

// }

// f(123,456,789)

// var c = 1;

// var f = function(){

// console.log(c);

// }

// function F(c){

// c = 2;

// f();

// }

// F();

// var arr = [];

// for(let i=0;i<5;i++){

// arr[i] = function(){

// console.log(this);

// }

// }

// arr[1]();

第九天

// 字符串对象：对应的构造函数是String

// 实例对象的显示原型对象：\_\_proto\_\_

// 构造函数的隐式原型对象：prototype

// 动态方法：实例对象.方法名()

// 静态方法：构造函数.方法名()

// 创建字符串：

// 一、直接量

var str = "abcd";

// 二、new关键字方式创建字符串实例对象

var newStr = new String("abcd");

// 字符串的长度属性：length 只读属性

// 字符的读取：

// 字符串[下标] IE8以下浏览器不支持，读取到的是undefined

// 根据位置读取字符：字符串.charAt(index)

// 字符串的方法：

// 一、转换方法

// 转换大小写：

// 小写：\*.toLowerCase()

// 大写：\*.toUpperCase()

// var str = "aAbB";

// var newStr = "abcd";

// var tempStr = "";

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

// if(newStr.includes(str.charAt(i))){

// tempStr += str.charAt(i).toUpperCase();

// }else{

// tempStr += str.charAt(i).toLowerCase();

// }

// }

// str.replace(/([a-z]{1})([A-Z]{1})/g,function($1,$2,$3){

// return $2.toUpperCase() + $3.toLowerCase();

// })

// 转换为数组：\*.split(分隔符,转换之后数组的长度)

// 字符和unicode编码值之间的转换：

// 字符转unicode编码值：字符串.charCodeAt(下标)

// unicode编码值转字符：String.fromCharCode(unicode编码值)

// 二、查找方法：

// \*.indexOf()、\*.lastIndexOf()

// \*.search()==>从前向后首次出现，支持正则表达式

// 三、操作方法：

// 字符串的拼接：\*.concat(各种数据)

// 字符串的截取：

// \*.slice(startIndex,endIndex)

// \*.substr(startIndex,length)

// \*.substring(startIndex,endIndex) 不支持负数

// 四、替换方法：

// \*.replace(待替换的字符或者字符串,替换的字符或者字符串)

// 该方法只替换第一个匹配字符或者字符串，不会改变原本的字符串，返回值是替换之后的字符串，支持正则表达式

// var str = "图片1图片2图片3图片4";

// var newStr = "<ul>";

// newStr += str.replace(/[\u4e00-\u9fa5]{2}[0-9]{1}/g,function($1){

// return `<li>${$1}</li>`;

// });

// newStr += "</ul>";

// 五、判断字符串的开始和结束:

// 开始：\*.startsWith(字符或者字符串)

// 结束：\*.endsWith(字符或者字符串)

// 六、字符串是否包含：\*.includes(字符或者字符串)

// 七、字符串的重复：\*.repeat(重复的次数)

// var str = "abcdefghijklmnopqrstuvwxyz";

// var newStr = "";

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

// newStr += str[i] + str[i].toUpperCase();

// }

// [].forEach.call(str,function(char){

// newStr += char + char.toUpperCase();

// })

// newStr = str.replace(/[a-z]{1}/g,function($1){

// return $1 + $1.toUpperCase();

// });

// str = str.repeat(1).toUpperCase() + str;

// 八、忽略空格：

// 左右忽：\*.trim()

// 左忽：\*.trimLeft()

// 右忽：\*.trimRight()

// 全部忽：

// String.prototype.trimAll = function(){

// return this.replace(/[\s]+/g,"");

// }

// 中间忽：

// String.prototype.trimCenter = function(){}

// var str = " a b c d "

// var s = str.replace(//g,function($1,$2){

// $2 = "";

// return $2;

// })

var str = " a b c d ";

// var startIndex = str.search(/[a-zA-Z0-9]{1}/);

// var endIndex = str.search(/[a-zA-Z0-9]{1}(?=[\s]+$)/g); // 负向前瞻

// var newStr = "";

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

// if(i >= startIndex && i <= endIndex && str[i] == " "){

// newStr += "";

// }else{

// newStr += str[i];

// }

// }

// var s = str.replace(/^[\s]+(?=[a-z]+([\s]+))(?![\s]+$)/g,function($1,$2){

// console.log($2);

// $2 = "";

// return $2;

// })

// 数学对象：用来保存数学公式以及数学信息的

// 绝对值：Math.abs()

// 最大值：Math.max()

// 最小值：Math.min()

// 向上取整：Math.ceil()

// 向下取整：Math.floor()

// 四舍五入取整：Math.round()

// 次方：Math.pow()

// 开方：Math.sqrt()

// 随机数：Math.random()

// 0 - 1之间（不包含0，不包含1）的浮点数

// for(var i=0;i<10000;i++){

// console.log(Math.random());

// }

// 0 - 10之间的随机整数

// Math.random()\*11 0 - 11之间的浮点数

// Math.floor(Math.random()\*11) 0 - 10

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

// console.log(Math.floor(Math.random()\*11));

// }

// 16 - 66之间的随机整数

// Math.floor(Math.random()\*51) + 16

// 随机整数公式：

// Math.floor(Math.random()\*(max - min + 1)) + min

// 10 - 99 之间的随机整数

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

// console.log(Math.floor(Math.random()\*(99 - 10 + 1)) + 10);

// }

// 间歇调用（定时器）：

// setInterval(函数||字符串,时间间隔==》毫秒为单位)

// 该方法的返回值是代表的当前定时器

// 一个文档中可以存在若干个定时器

// 清除间歇调用：

// clearInterval(定时器)

// var i = 0;

// var id = setInterval(function(){

// console.log(++i);

// if(i == 100) clearInterval(id);

// },10);

// 封装函数：范围内得到n个不重复的随机整数

function randDigits(min,max,n){

var tempArr = [];

while(tempArr.length < n){

var randDigit = Math.floor(Math.random()\*(max - min + 1 )) + min;

if(!tempArr.includes(randDigit)){

tempArr.push(randDigit);

}

}

return tempArr;

}

randDigits(min,max,n)；

从大到小的字符串

<div id = “box”></div>

var str = "";

var newStr = "从大到小的字符串";

var fontSize = 82;

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

str += `<span style="font-size:${fontSize - i\*10}px">${newStr[i]}</span>`;

}

box.innerHTML = str;

数量加减

<input type="button" value="-" id="minus"><input type="text" value="1" name="" id="count" readonly><input type="button" value="+" id="plus">

<script>

minus["onclick"] = function(){

count.value = Math.max(--count.value,1);

}

plus.onclick = function(){

// if(count.value >= 10){

// count.value = 10;

// }

count.value = Math.min(++count.value,10);

}

第十天

日期时间对象：

在js里，日期时间是以特定的时间段（时间戳）来保存日期时间的，是从1970年向前和向后各推285616年

实例化：

var date = new Date();

把日期时间对象转换为时间戳（数字）：

date\*1

date.valueOf()

date.getTime()

Date.parse(date)

Number(date)

时间戳：

一年：365\*24\*60\*60\*1000

一天：24\*60\*60\*1000

一小时：60\*60\*1000

一分钟：60\*1000

一秒钟：1000

var years = Math.floor(date/(365\*24\*60\*60\*1000));

组件方法：

获取满年：date.getFullYear()

设置满年：date.setFullYear()

月的取值范围：0 - 11 0代表1月份

获取月：date.getMonth()

设置月：date.setMonth()

日期的取值范围：1 - 31

获取日期：date.getDate()

设置日期：date.setDate()

小时的取值范围：0 - 23

获取小时：date.getHours()

设置小时：date.setHours()

0 - 59

获取分钟：date.getMinutes()

设置分钟：date.setMinutes()

0 - 59

获取秒：date.getSeconds()

设置秒：date.setSeconds()

0 - 999

获取毫秒：date.getMilliseconds()

获取毫秒：date.setMilliseconds()

星期的取值范围：0 - 6

获取星期：date.getDay()

获取格林尼治时间和本地时间相差的分钟数：date.getTimezoneOffset()

以上所有方法对应的都有UTC（世界协调时间、世界统一时间、格林尼治时间）时间

如何获取设置年月的天数？

var sy = 2032;

var sm = 2;

var date = new Date(sy,sm,0).getDate();

function getDays(year,month){

return new Date(year,month,0).getDate();

}

console.log(getDays(2089,5));

n个月后的今天是星期几？

function getWeek(n){

var date = new Date();

return new Date(date.getFullYear(),date.getMnth() + n,date.getDate()).getDay();

}

console.log(getWeek(2));

设置年月的第一天是星期几？

function getFirst(year,month){

return new Date(year,month - 1).getDay();

}

console.log(getFirst(2018,9));

设置5天之后的日期时间（当前时间）

var date = new Date();

date.setDate(date.getDate() + 5);

// 日期时间对象：Date

// 日期时间对象是以特定的时间段（时间戳）来保存日期时间的，从1970年向前和向后各推285616年。UTC世界协调时间、世界统一时间、格林尼治时间

// 实力化日期时间对象：得到的是格式化好的日期时间

// var date = new Date();

// 获取当前时间

// date.toLocaleTimeString();

// "下午2:04:21"

// date.toLocaleDateString();

// "2018/7/2"

// 如何把日期时间对象转为时间戳：

// 1、方式一

// console.log(Date.parse()); // 精确到秒

// 2、方式二

// console.log(date.getTime()); // 精确到毫秒的

// 3、方式三

// 把实力化对象转换为数字Number(date) // 精确到毫秒

// 4、方式四

// console.log(date.valueOf()); // 精确到毫秒的

// Date.UTC(年,月,日,时,分,秒,毫秒) 转为格林尼治时间

// 日期时间的设置：

// new Date(年,月,日,时,分,秒,毫秒)

// 如果设置日期时间，至少需要两个参数，年月，后面参数省略时，日期默认是当前月的1号，时分秒默认00

// 月的取值范围：0 - 11

// 日期的取值范围：1 - 31

// 小时的取值范围：0 - 23

// 分钟和秒：0 - 59

// 毫秒的：0 - 999

// 星期的取值范围：0 - 6

// 时间戳：

// 一天：

// 一个星期：

// 一个小时：

// 一分钟：

// 一秒钟：

// 组件方法：日期时间实例对象

// var date = new Date()

// 获取年：date.getFullYear()

// 设置年：date.setFullYear()

// 获取月：date.getMonth()

// 设置月：date.setMonth()

// 获取日期：date.getDate()

// 设置日期：date.setDate()

// 获取小时：date.getHours()

// 设置小时：date.setHours()

// 获取分钟：date.getMinutes()

// 设置分钟：date.setMinutes()

// 获取秒：date.getSeconds()

// 设置秒：date.setSeconds()

// 获取毫秒：date.getMilliseconds()

// 设置毫秒：date.setMilliseconds()

// 获取星期：date.getDay()

// 以上所有方法都对应的有UTC方法，比如getUTCHours()

// 格林尼治时间与本地时间相差的分钟数：

// \*.getTimezoneOffset()

// 间歇调用：setInterval() 定时调用

// 该方法的返回值代表当前的定时器

// 语法结构：时间间隔是毫秒

// 1、setInterval(函数,时间间隔);

// 2、setInterval(字符串,时间间隔);

// 3、setInterval("函数(参数)",时间间隔);

// 清除间歇调用：clearInterval(定时器)

Dom

// DOM：描绘的是一个层次化的节点树（document文档）

// 节点： 所有的节点都是对象

// 节点名称 节点类型 节点值

// nodeName nodeType nodeValue

// 文档节点 #document 9 null

// 元素节点 大写的标签名称 1 null

// 属性节点 属性名称 2 属性值

// 文本节点 #text 3 文本内容

// 注释节点 #comment 8 注释内容

console.log(document.nodeName,document.nodeType,document.nodeValue);

console.log(box.nodeName,box.nodeType,box.nodeValue);

// attributes:元素节点对象上所有属性的集合

console.log(box.attributes[0].nodeName,box.attributes[0].nodeType,box.attributes[0].nodeValue);

// childNodes:父元素下所有子节点的集合

console.log(box.childNodes[0].nodeName,box.childNodes[0].nodeType,box.childNodes[0].nodeValue);

console.log(box.childNodes[3].nodeName,box.childNodes[3].nodeType,box.childNodes[3].nodeValue);

// var textNode = document.createTextNode("This is a paragraph!");

// box.childNodes[1].innerHTML = textNode.nodeValue;

查找元素节点对象的方法：

一、通过id选择器获取元素节点对象：（只能获取一个）

document.getElementById(id选择器名称)

该方法不是元素下的方法

如果不存在，则获取到的是null

console.log(document.getElementById("box"),1);

window.onload = function(){

// 预加载事件中的程序

console.log(document.getElementById("box"),2);

}

console.log( document.getElementById("s") );

二、通过标签选择器获取元素节点对象：（一组）

document.getElemnetsByTagName(标签名称)

如果不存在，则获取到的是一个空集合；如果存在，则该集合是一个类似数组的对象

该方法是元素下的方法

var sa = document.getElementsByTagName("a");

console.log(Array.isArray(Array.from(sa)),Array.isArray([]),sa instanceof Array,sa.constructor.name,Object.prototype.toString.call(sa));

instanceof:检一个实例对象是否属于一个函数对象

console.log(document.getElementById("box").getElementsByTagName("p"));

三、通过类选择器获取元素节点对象：（一组）

document.getElementsByClassName(类名)

不能兼容IE8以下浏览器（包括IE8）

是元素下的方法

console.log(document.getElementById("box").getElementsByClassName("p"));

四、通过name属性名称获取元素节点对象：（一组）

document.getElementsByName(name属性名称)

该方法不是元素下的方法

console.log(document.getElementsByName("type"));

五、通过各种选择器获取一个元素节点对象：

document.querySelector(各种选择器)

该方法是元素下的方法

console.log(document.querySelector("#box").querySelector("input:not([type=file])"));

六、通过各种选择器获取一组元素节点对象：

console.log(document.querySelectorAll("p:not(.p)"));

console.log(document.querySelectorAll("#box input:not([type=file])"));

// 属性的操作：自定义属性、固有属性、对象属性

// 自定义属性：标签上的自定义属性、对象上的自定义属性

// 固有属性：标签上的固有属性、对象上的固有属性

// a.dataIndex = 0; // 对象上的自定义属性=

// console.log(a.id,a.dataIndex,[].concat(a));

// 属性集合：元素节点对象.attributes

// 标签上的所有的属性：自定义属性、固有属性

// 属性的操作方法：操作的是标签上的属性

// 获取：元素.getAttribute(属性名称)

// 设置：元素.setAttribute(属性名称,属性值)

// 删除：元素.removeAttribute(属性名称)

十一天

DOM属性（元素节点对象上的属性）：

console.log([].concat(box));

标签

attributes:标签上所有现有属性的集合

childElementCount:父元素下所有子元素的数量

childNodes:父元素下所有的子节点

children:父元素下所有的子元素

classList:元素节点对象上类名的集合

className:当前元素的类名

clientWidth、clientHeight:当前元素的宽高（不包含边框，包含填充）

clientLeft、clientTop:当前元素的边框线的宽度

offsetWidth、offsetHeight:当前元素的宽高（包含边框线，包含填充)

offsetLeft、offsetTop:当前元素的左上偏移量

contentEditable:当前元素是否可编辑

draggable:当前元素是否可拖拽

firstChild:父元素下的第一个子节点

firstElementChild:父元素下的第一个子元素

lastChild:父元素下的最后一个子节点

lastElementChild:父元素下的最后一个子元素

innerHTML:当前元素内的所有内容

outerHTML:包含当前元素的所有内容

innerText:当前元素内的所有的文本内容

textContent:当前元素内的所有的文本内容

previousSibling:上一个兄弟节点

previousElementSibling:上一个兄弟元素节点

nextSibling:下一个兄弟节点

nextElementSibling:下一个兄弟元素节点

所有以on开头的都是DOM0级事件

parentNode、父节点 parentElement:父元素

scrollWidth、scrollHeight:当前元素的滚动宽高

scrollLeft、scrollTop:当前元素的滚动左边距和滚动上边距

style:当前元素的行内样式对象

tagName : 当前元素的标签名称（大写）

表格 属性或方法 说明

caption 保存着<caption>元素的引用;

tBodies 保存着<tbody>元素的HTMLCollection集合;

tFoot 保存着对<tfoot>元素的引用;

tHead 保存着对<thead>元素的引用;

rows 保存着对<tr>元素的HTMLCollection集合;

createTHead() 创建<thead>元素,并返回引用;

createTBody() 创建tbody

createTFoot() 创建<tfoot>元素,并返回引用;

createCaption() 创建<caption>元素,并返回引用;

deleteTHead() 删除<thead>元素;

deleteTFoot() 删除<tfoot>元素;

deleteCaption() 删除<caption>元素;

deleteRow(pos) 删除指定的行;

insertRow(pos) 向rows集合中的指定位置插入一行;

<tbody>元素添加的属性和方法

deleteRow(pos) 删除指定位置的行;

insertRow(pos) 向rows集合中的指定位置插入一行;

<tr>元素添加的属性和方法

cells 保存着<tr>元素中单元格的HTMLCollectioin集合;

deleteCell(pos) 删除指定位置的单元格;

insertCell(pos) 向cells集合的指定位置插入一个单元格,并返回引用

HTMLDOM获取表格的<caption>

alert(table.caption.innerHTML);

获取caption的内容;

PS:在一个表格中<thead>和<tfoot>是唯一的,只能有一个;

而<tbody>不是唯一的,可以是多个,这样导致最后返回的<thead>和<tfoot>是元素引用;而<tbody>是元素集合;

获取元素样式使用最终的方式：（兼容）

IE:元素.currentStyle （IE当前使用的样式对象）

非IE:window.getComputedStyle(元素,伪类选择器null) （非IE电脑使用样式对象）

封装

function getStyle(ele,attr){

var res = null;

if(window.getComputedStyle){ // 非IE

res = window.getComputedStyle(ele,null)[attr];

}else{ // IE

res = ele.currentStyle[attr];

}

return res;

}

DOM元素节点的操作方法:

一、创建元素节点对象的方法：document.createElement(标签名称)

var p = document.createElement("p");

p.innerHTML = "这是一个段落！";

var comment = document.createComment("这是注释节点！");

var text = document.createTextNode("这是一个文本节点！");

var text1 = document.createTextNode("这是一个文本节点！");

二、追加元素节点对象的方法：

父元素.appendChild(子节点) 添加到当前父元素的尾部

box.appendChild(comment);

box.append(comment);

box.firstElementChild.appendChild(text);

box.appendChild(text);

三、追加到父元素的最前面的方法：

父元素.prepend(子元素)

box.prepend(p);

四、删除元素节点对象的方法：

父元素.removeChild(子节点)

box.removeChild(comment);

五、删除当前元素节点对象的方法：

元素.remove()

box.remove();

六、插入元素节点对象的方法：

父元素.insertBefore(新节点,旧节点)

box.insertBefore(p,comment);

七、在某个元素节点对象后插入一个元素节点对象：

兄弟元素.after(插入的元素)

comment.after(p,document.createElement("a"));

八、替换元素节点对象的方法：

父元素.replaceChild(新节点,旧节点)

var div = document.createElement("div");

div.innerHTML = "赵海龙睡着了！";

box.replaceChild(div,comment);

九、一个元素节点对象替换另一个元素节点对象：

被替换的元素.replaceWith(替换的元素)

box.replaceWith(div);

十、克隆元素节点对象的方法：

被克隆的节点.cloneNode(true|false)

如果不写，默认false

var c = comment.cloneNode();

var box1 = box.cloneNode(true);

十一、碎片整理：

document.createDocumentFragment()

var fragment = document.createDocumentFragment();

for(var i=0;i<5;i++){

var div = document.createElement("div");

div.innerHTML = 5 - i;

fragment.prepend(div);

}

document.body.prepend(fragment);

document.body.innerHTML = "这是一个body体！";

options:当前下拉菜单下的所有的option选项

selectedIndex:当前被选中选项的下标

selectedOptions:当前被选中选项的集合

1. Dom0

// 事件：是javascript跳动的心脏，是用户和浏览器之间交互的桥梁

// 事件流：

// 顺序：事件捕获=》目标元素（触发该事件的元素）=》事件冒泡

// 事件捕获：从最不具体的元素向内捕获=》从外向里

// 事件冒泡：从最具体的元素向外冒泡=》从里向外

// btn.addEventListener("click",function(){

// alert(1);

// },true);

// document.body.addEventListener("click",function(){

// alert(2);

// },true);

// DOM0级事件：同一元素上，同类型的事件只能执行一个

// 添加：元素.on+事件名称 = function(){}

// 删除：元素.on+事件名称 = null;

// btn.onclick = function(){

// alert(1);

// }

// btn.onclick = function(){

// alert(2);

// }

// DOM2事件：稍后讲（你们猜？）

// 事件位置：

// 内联事件：事件作为标签的属性而存在

// 脚本事件：脚本事件

// 事件对象(event)：有关当前事件的所有的信息

// IE：window.event (IE8（包括）以下浏览器不支持)

// 非IE：函数内的参数 元素.on+事件名称 = function(e){}

// btn.onclick = function(e){

// // 事件对象的兼容

// var evt = window.event || e;

// }

// 阻止事件冒泡：

// IE：事件对象.cancelBubble true（阻止） false（不阻止） 默认情况false 不阻止

// 非IE：事件对象.stopPropagation()

// 事件冒泡的兼容：事件对象.stopPropagation ? 事件对象.stopPropagation() : 事件对象.cancelBubble = true;

minBox.onclick = function(evt){

var e = window.event || evt;

事件对象

阻止冒泡

e.stopPropagation ? e.stopPropagation() : e.cancelBubble = true;

this.style.background = "purple";

}

midBox.onclick = function(){

this.style.background = "aqua";

}

box.onclick = function(){

this.style.background = "hotpink";

}

// 事件委托（事件代理）：同类型的事件委托到其父辈元素上

// 目标元素（触发该事件的元素）：

// IE：事件对象.srcElement

// 非IE：事件对象.target

// 事件传参：

// DOM0级事件的函数是在事件被触发时执行

// box.onclick = function(evt){

// var e = window.event || evt;

// var t = e.srcElement || e.target;

// t.style.background = "hotpink";

// }

// 事件类型：

// 一、鼠标事件：

onclick点击事件、

ondblclick鼠标双击、

onnmouseover鼠标滑过、支持事件冒泡

onmouseout、鼠标滑出 支持事件冒泡

onmouseenter、光标首次划过 不支持事件冒泡

onmouseleave、鼠标离开事件 不支持事件冒泡

onmousedown、鼠标按下事件

onmouseup、鼠标谈起事件

onmousemove、鼠标指针在元素内部移动时重复触发 || 划过显示坐标

onwheel滑轮事件

// btn.onclick = function(){}

// box.ondblclick = function(){

// this.contentEditable = true;

// }

// mouseover和mouseout支持事件冒泡

// box.onmouseover = function(){

// this.innerHTML = "大板龙是个大睡神！";

// this.contentEditable = true;

// }

// box.onmouseout = function(){

// this.contentEditable = false;

// }

// mouseenter和mouseleave不支持事件冒泡

// box.onmouseenter = function(){

// this.innerHTML = "大板龙是个大睡神！";

// this.contentEditable = true;

// }

// box.onmouseleave = function(){

// this.innerHTML = "";

// this.contentEditable = false;

// }

// box.onmouseleave = function(){

// console.log("鼠标离开了");

// }

// box.onmousedown = function(){

// console.log("鼠标摁下了");

// }

// box.onmouseup = function(){

// console.log("鼠标弹起了");

// }

// document.onmousemove = function(evt){

// var e = evt || window.event;

// var pos\_x = e.x || e.pageX || e.clientX

// , pos\_y = e.y || e.pageY || e.clientY;

// x.value = pos\_x;

// y.value = pos\_y;

// }

// document.onwheel = function(){

// console.log("事件触发了...");

// }

// 二、键盘事件：

onkeydown、鼠标按下

onkeyup、鼠标谈起

onkeypress鼠标按压

// 三、html事件：

onsubmit、表单提交

onreset、表单重置

onchange、元素改变

onblur、元素失去焦点

onfocus、元素获取焦点

onscroll、滚动条事件

onload、预加载事件

onresize页面大小发生改变时发生时间

// down、press是在字符写入之前触发事件

// up是在字符写入之后触发事件

// down、up键值不区分大小写

// press键值区分大小写

// down、up数字键区分小键盘

// down、up键盘上的所有键都会触发

// press键盘上的功能键不会触发该事件

// 键盘摁下

area.onkeydown = function(evt){

var e = evt || window.event;

// 键值：keyCode

console.log(e.keyCode,1);

// alert(this.value);

}

// 键盘弹起

// area.onkeyup = function(evt){

// var e = evt || window.event;

// console.log(e.keyCode,2);

// // alert(this.value);

// }

// 键盘摁压

// area.onkeypress = function(evt){

// var e = evt || window.event;

// console.log(e.keyCode,3);

// // alert(this.value);

// }

// 阻止默认行为：

// IE：事件对象.returnValue true(不阻止) false(阻止)

// 非IE：事件对象.preventDefault();

// phone.onkeypress = function(evt){

// var e = evt || window.event;

// var code = e.keyCode;

// if(code < 48 || code > 57){

// e.preventDefault ? e.preventDefault() : e.returnValue = false;

// }

// }

// phone.onblur = function(){

// var val = this.value;

// var exp = /^1[3|4|5|6|7|8][0-9]{9}$/;

// if(!exp.test(val)){

// alert("手机号输入有误...");

// }

// }

// var exp = /^1[3|4|5|6|7|8][0-9]{9}$/;

// var phone = "11603820238";

// 正则.test(待检测字符串) true false

// console.log(exp.test(phone));

// form.onsubmit = function(){

// alert("表单被提交了....");

// }

// form.onreset = function(){

// alert("表单被重置了...");

// }

// txt.onselect = function(){

// console.log("内容被选中");

// }

// txt.onfocus = function(){

// console.log("获取了焦点");

// }

// sel1.onchange = function(){

// console.log(this.selectedIndex,this.selectedOptions,this.options);

// }

// window.onscroll = function(){

// document.title = document.body.scrollTop || document.documentElement.scrollTop;

// }

// window.onresize = function(){

// console.log("浏览器窗口大小发生了变化");

Dom2

// DOM2级事件：同一元素，同类型的事件可以执行多次

// DOM2级事件的绑定：

// IE:元素.attachEvent("on"+事件名称,函数)

// 非IE:元素.addEventListener(事件名称,函数,事件捕获)

// DOM2级事件的删除：

// IE:元素.detachEvent("on"+事件名称,函数)

// 非IE:元素.removeEventListener(事件名称,函数,事件捕获)

// btn.onclick = function(){

// }

// btn.attachEvent("onclick",function(){

// alert(1);

// })

// btn.addEventListener("click",function(){

// alert(2);

// })

// 封装一个兼容函数：给元素绑定事件

function addEvent(ele,eventName,fn){

if(ele.attachEvent){

ele.attachEvent("on"+eventName,fn);

}else if(ele.addEventListener){

ele.addEventListener(eventName,fn);

}else{

ele["on"+eventName] = fn;

}

}

addEvent(btn,"click",f)

addEvent(btn,"click",f1)

addEvent(btn,"click",f2)

function f(){

alert(1);

}

function f1(){

alert(2);

}

function f2(){

alert(3);

}

// 封装一个兼容函数：给元素删除事件

function removeEvent(ele,eventName,fn){

if(ele.detachEvent){

ele.detachEvent("on"+eventName,fn);

}else if(ele.removeEventListener){

ele.removeEventListener(eventName,fn);

}else{

ele["on"+eventName] = null;

}

}

var f3 = function(){

// 删除btn上的点击事件

removeEvent(btn,"click",f);

removeEvent(btn,"click",f1);

// removeEvent(btn,"click",f2);

}

addEvent(btn2,"click",f3);

addEvent(btn1,"click",function(){

removeEvent(btn2,"click",f3)

});

DOM

// Dom0级事件的绑定:元素.on + 事件名称 = function(){}

//Dom0级事件的删除:元素.on + 事件名称 = null;

// Dom2级时间的绑定：兼容 捕获

绑定

// IE:元素.attachEvent("on"+事件名称,函数fn)

//非IE:元素.addEventListener(事件名称，函数fn，事件捕获（true || false）)

删除

// IE：元素.detachEvent("on"+事件名称,函数fn)

// 非IE：元素.removeEventListener(事件名称,函数fn,事件捕获true || false)

// document.body.addEventListener("click",function(){

// alert(2);

// },true);

// btn.addEventListener("click",function(){

// alert(1)

// },true);

//把DOM2的事件方到对象里面直接调用：

var $ = {

//ele元素eventName事件名fn执行的函数dc事件捕获（true开启 || false默认阻止）

addEvent:function(ele,eventName,fn,dc){

ele.addEventListener ? ele.addEventListener(eventName,fn,dc) : ele.attachEvent("on" + eventName,fn);

},

removeEvent:function(ele,eventName,fn,dc){

ele.removerEventListener ? ele.removeEventListener(eventName,fn,dc) : ele.detachEvent("on" + eventName,fn);

}

}

//用法：

// $.addEvent(btn,"click",funnction(){})

// $.removeEvent(btn,"click",function(){})

// $.addEvent(btn,"click",function(){

// alert(1);

// },true);

// $.addEvent(document.body,"click",function(){

// alert(2);

// },true)

// 事件对象：包含有关当前事件的所有信息

// IE：window.event

//非IE：元素.on+事件名称 = function(e事件对象){}

// btn.onclick = function(e){

// var evt = e || window.event;

// console.log(e)

// }

// 或

// $.addEvent(btn,"click",function(event){

// var e = event || window.event;

// console.log(e);

// });

// 事件冒泡：

// $.addEvent(maxBox,"click",function(evt){

// var e = evt || window.event;

// this.style.background = "hotpink";

// console.log(e);

// });

// midBox.onclick = function(){

// this.style.background = "aqua";

// }

// minBox.onclick = function(evt){

// var e = evt || window.event;

// e.stopPropagation ? e.stopPropagation() : e.cancelBubble = true;

// this.style.background = "skyblue";

// }

// 阻止事件冒泡：

// IE：事件对象.cancelBulle = true;

// 非IE：事件对象.stopPropagation();

// 事件委托：把同一类型的事件委托到其父辈元素（爸爸、爷爷......）上

// 目标元素：

// IE：事件对象.srcElement

// 非IE：事件对象.target

// $.addEvent(maxBox,"click",(function(color){

// return function (evt){

// var e = evt || window.event;

// var t = e.target || e.srcElement;

// t.style.background = color;

// }

// })("limegreen"))

// $.addEvent(maxBox,"click",\_=>{

// var e = \_ || window.event;

// var t = e.target || e.srcElement;

// var color = null;

// switch(true){

// case t.id == "maxBox":

// color = !t.show ? "orange" : "pink";

// break;

// case t.id == "midBox":

// color = !t.show ? "yellow" : "red";

// break;

// case t.id == "minBox":

// color = !t.show ? "chocolate" : "blue";

// break;

// }

// t.show = !t.show;

// t.style.background = color;

// });

// 阻止默认行为：

// IE：事件对象.returnValue = true（不阻止） || flase（阻止）

// 非IE：事件对象.preventDefault()

// 阻止表单的提交和链接的跳转：javascript:void(0)

a.onclick = function(evt){

var e = evt || window.event;

console.log(e);

e.preventDefault ? e.preventDefault() : e.returnValue = false;

}