* js代码需要写到script标签中。如：<script type=”text/javascript”></script>
* alert:弹出一个框。如：alert(“hello world!”);
* document.write:输出内容(写到网页中的)。如：document.write(“你好！”);所以换行是br不是\n
* Console.log:向控制台输出一个内容。如：console.log(“您好”)；
* Console.error:打印的结果为红色的，并且前面带有×
* Prompt()函数:可以弹出一个提示框，该提示框会带有一个文本框，用户可以在文本框中输入一段内容，该函数需要一个字符串作为参数，该字符串作为提示框的提示文字。如：prompt("请输入一个成绩："); 。可以将prompt()函数的返回值内容传给另外一个变量。如：var a=prompt("请输入一个成绩：");
* Prompt()函数返回类型是string。可以在prompt前加个加号转成number。如：+prompt()
* Confirm()：用于弹出一个带有确认和取消按钮的提示框。注意：需要一个字符串作为参数，该字符串将会作为提示文字显示出来。如var flag=confirm(“确认删除吗”)
* Onclick：鼠标点击事件：将js代码放在onclick双引号里，当点击框框的时候就出发该事件。如：<button onclick=”alert(‘点我干嘛！’)”>点我！</button>
* js中获取html的标签只能用id不能用class。Css获取html标签class为. id为#
* 用var来声明一个变量。如：var index=10；var str=”1223”(字符串既可以单引号也可以双引号);
* 转义字符：\ 如：\”会输出”
* Number.MAX\_VALUE：表示数值的最大值（正数）。Number.MIN\_VALUE：表示0以上的最小值（正数0.000000000000000000000..............00001）
* 查询某值的类型：typeof。如 var index=10;console(type index);输出结果为number（类型）
* NaN：一个特殊的数字，表示not a number。但是使用typeof检查NaN时返回number。
* Infinity：正无穷，-infinity：负无穷。使用typeof检查infinity时返回number
* Null类型，值只有null一个。Typeof null 返回的值类型为object
* undefined类型（未定义）：值只有undefined一个。当声明一个变量但不给变量赋值时，编译器默认值就是undefined。Typeof undefined 返回的值类型也是undefined
* Boolean（布尔值）。True/false
* 转换成string类型：

1 调用.tostring()方法。如：var a=12; var b=a.tostring(); b值为”12”。注意：null、undefined没有tostring()方法，而boolean可以用tostring()，结果为”true”或者”false”。

2 调用String()函数(注意S为大写，一定要大写！！！)。如：var a=12；a=String(a)。a值为”12”。注意：null、undefined可以用string()函数，值结果为”null”,”undefined”。而boolean可以用tostring()，结果为”true”或者”false”。

* Number()函数(大写)。如var a=”123”;a=number(a);其结果为123。注意：1括号里只要有非数字，就会转成NaN。如：var a=”123abc”;a=number(a);其结果为NaN。2如果为空字符，则转换结果为0。如：var a=” ”;a=number(a);其结果为0。3boolean转数字，True为1，false为0。4类型null转换成数字结果为0。5undefined转成数字结果为NaN。
* parseInt():将把一个字符串转换为一个整数(从第一位数开始找)。如：var a=”123px”;a= parseInt(a);其结果为123。又如：var a=”123px456”;a= parseInt(a);其结果为123。又如：var a=”a123px456”;a= parseInt(a);其结果为NaN。又如：var a=”123.456”;a= parseInt(a);结果为123
* parseFloat()：把一个字符串转换成浮点数。如：var a=”123.456px”;a= parseFloat(a);结果为123.456。又如var a=”123.456.789”;a= parseFloat(a);结果为123.456。又如var a=”123”;a= parseFloat(a);结果为123（不会是123.000000！！！！！）
* 如果对非string的值，parseInt()、parseFloat()会把该值先转换成string值，在转成int/float
* 0x开头的数字为16进制，但是输出时会以十进制输出。如：var a=”0x123”;结果为123
* 0开头的数字为八进制
* 0b开头的数字为二进制
* 可以在parseInt()中传入第二个参数，来指定需要转成的进制。如：var a=”020”;a= parseInt(a，10);结果为20
* boolean()函数：转换成boolean类型。正数/负数--true，0--false，infinity--true，NaN--false，字符串--true，”false”（字符串）--true，””(空)--false，null--false，undefined--false，对象--true
* typeof是一个运算符，返回值是一个字符串（类型）typeof null，typeof xxx的返回值都是字符串
* 加法运算（+）：1当对非number的值进行加法运算时(除了字符串加法)，会先把这些值转换成number，如：1+false、1+true、1+NaN。2任何值和NaN运算结果都为NaN。3两个字符串相加会拼成一个字符串，如”123”+”456”=”123456”。4任何值和字符串做加法运算，该值都会先转换成字符串，再进行拼串，如：123+”a”=”123a”
* 减法运算（-）：1当对非number的值进行减法运算时(即使字符串减法)，会先把这些值转换成number，如：99-”1”=98，1-false=1
* 乘法运算（\*）：当对非number的值进行乘法运算时(即使字符串乘法)，会先把这些值转换成number，如：99\*”1”=99,2\*undefined=NaN
* 除法运算（/）：同-、\*
* 任何做-、\*、/运算都会自动转成number（+特殊），因此可以用减乘除将某个值转换成number
* 正负号（+/-）（一元运算符）：对于非number类型值，会先转换成number值在进行运算。如：var a=true;a=-a;结果为-1
* 非boolean类型的值进行与&&运算时，如果两个都为真，返回后面的值（返回的是原值，不是true/false）。如：result=1&&2，result结果为2，又如：result=1&&2，result结果为1
* 比较（>、>=、<、<=等等）：当非number类型值进行比较时，会先把非数字值转换成数字再进行比较。如1>true，结果为false，1>NaN，结果为false，1>undefined（undefined转成数字结果为NaN），结果为false。结论：1任何值和NaN作对比，结果都是false。2两个字符串比较时，比的是字符串的Unicode编码
* Unicode编码：1在字符串中要使用/u才能表示Unicode编码，格式：/u四位编码。如：”/u1C00”。2在网页中要使用&#编码才能表示Unicode编码，格式：&#编码（编码为十进制）。如：&#2620
* 相等运算符（==）（会自动做类型转换）：大部分都是转成number再比较，如：”1”==1，结果为true（“true”转成1），true==”1”结果为true（“true”和”1”都转成number）。例外：null==0，结果为false(null不会转成number！！！！)，NaN==NaN结果为false（NaN不和任何值相等包括它本身！！！！）。Null==undefined结果为true
* 不等（!=）（会自动做类型转换）
* 全等（===）（不会自动做类型转换！！！）如：”123”==123为true，但”123”===123为false！！
* 不全等（!==）（不会自动做类型转换！！）
* isNaN()函数：判断一个数是否为NaN。isNaN()的返回值类型为boolean
* JS的五大数据类型：number、string、null、boolean、undefined。如果看到的数据类型不是这五个中的某个，则表示这类型为对象（object）
* 对象（object）（用的最多的是内建对象和宿主对象）。分类：1内建对象：由ES标准中定义的对象，在任何的ES的实现中都可以使用。如Math、string、number、boolean、object。2宿主对象：由JS的运行对象环境提供的对象（即浏览器提供的对象）。如BOM、DOM。3自定义对象。如var obj=new object()（一般不采用这种方法声明对象，一般采用48）
* 对象没有括号。对象加括号叫做返回值或者函数
* 自定义对象：1向对象添加属性：语法：对象.属性名=属性值。如obj.name=”孙悟空”。2读取对象中的属性：对象.属性名。如console.log(obj.name)。3修改值，和添加属性一样：obj.name=”猪八戒”。4删除对象属性：delete 对象.属性名。如：delete obj.name。5如果要去特殊的属性名，则要采用另一种方式：对象[“属性名”]=属性值，读取时也要采用这种方式。如：obj[“123”]=789;console.log(obj[“123”]);
* JS的的对象可以当做另一个对象的属性。如obj1.name=obj2。注意：没有指定obj2的属性，所以结果为undefined或者object
* in运算符：通过该运算符可以检查一个对象中对否含有指定的属性，如果有则返回true，没有返回false。语法：”属性名” in 对象。如：检查obj中是够有test属性：console.log(“test” in obj);
* 对象字面量{}:使用对象字面量可以在创建对象时直接指定对象的属性。语法：{属性名：属性值，属性名：属性值......} 如：var obj={name:”猪八戒”，age:18}
* 函数也是对象，除了那5个类型，看到的都是对象。
* 函数（function）：类型为function。语法：function 名([形参1，形参2....]){函数体}，可以不要形参
* 函数表达式（匿名函数，没有函数名）。语法：function() ([形参1，形参2....]){函数体}。。注意：如果直接写函数表达式，则编译器会报错。解决方法：在整个函数表达式外边加一个括号：（function() ([形参1，形参2....]){函数体}）。此时这个函数表达式函数对象。1在这个函数对象后边再加一个括号，叫做立即函数，如（function() ([形参1，形参2....]){函数体}）（），这个立即函数可以直接编译运行。但是只能执行一次，因为没有被保存所以会丢失。2将匿名函数赋值给变量。如var func=function();
* 函数也能变成立即函数。如func(a,b){.....}(100,120)。这是一个参数函数变立即函数，注意加的括号里也要加参数
* 1调用函数时解析器不会检查实参的数量。2多余的实参不会被赋值。3如果实参的数量少于形参的数量，则没有对应实参的形参将是undefined。3函数里的return如果不跟值则返回undefined。如func(){........return;}返回的是undefined
* 调用函数对象（不带括号），则显示内容为函数所有的文字。如果调用的是函数（带有括号），则显示的是函数体里面的内容（返回值）
* Return：返回值可以是任意数据对象，也可以是对象，也可以是函数，也可以为空（此时相当于结束该函数）
* 方法是面向对象的，函数时面向过程的。如：obj.func()这个叫做调用obj的方法，而func()叫做调用func函数
* for....in....语句。语法：for(var i in obj){}。对象中有几个属性，循环体就好几次循环。1每次执行时都会将对象中的一个属性的名字赋值给变量i。如console.log(i)。2获取对象的属性值：要用数组形式[]，不能用.形式。如：console.log(obj[i])，不能是console.log(obj.i)是错的！！！
* JS只有两种作用域：1全局作用域、2函数作用域。
* 全局作用域：编写在script标签中的js代码，都在全局作用域中。全局作用域在页面打开时生成，在页面结束时销毁。1在全局作用域中，创建的的变量都会作为window对象的属性保存。如：console.log(window)，会输出window这个对象。如设置一个全局变量var a=3；则这个a的对象为window。console.log(window.a),则输出结果为3。2全局作用域的变量声明会提前。使用var关键字声明的的变量，会在所有代码执行之前被声明。如console.log(a);var a=3;执行时编译器不会报错，输出结果为undefined。原因：编译器会默认变成var a;console.log(a);a=3;！！！！！很特别!!!!! 3全局作用域的函数声明也会提前（和2一样）。使用函数声明形式创建的函数function 函数(){}。他会在所有代码执行之前就被创建，所以可以在函数声明前来调用该函数。但是使用函数表达式形式声明的函数不会被提前
* 函数作用域：调用函数时创建函数作用域，函数执行完毕后函数作用域销毁。注意：1函数作用域中可以访问全局作用域的变量，但全局变量不能访问函数作用域的变量。2当在函数作用域操作一个变量时，它会先在自身作用域中寻找，如果有就直接使用，如果没有则向上一级作用域中寻找，直到找到全局作用域。如果全局作用域还是没找到，则会报错referenceerror
* 在函数中，不是用var声明的变量，且全局中也没有声明这个变量，编译器会把它设置成全局变量。如function func(){console.log(“c=”+c);c=10;} func();执行func()，输出结果为c=10;原因：c没有使用var关键字，则会设置成全局变量，c=10;相当于window.c=10;
* 定义形参就相当于在函数作用域中声明了变量。如function func(index){.....}就相当于function func(index){var index;.....}
* 从函数开始就开始很乱，记得再看一遍！！！！（63-67）
* 先看局部里有没有声明局部变量，很重要！！！！
* 例子1：var a=123;function func(){alert(a);a=456;} func();alert(a);输出结果：先输出123，再输出456。分析：按顺序先执行全局func(),调用func函数，先看局部有没有声明a，没有声明，执行局部alert(a)时，会先找a，此时找到的是全局变量a=123，输出123，执行完后继续执行a=456，此时a=456。最后再实行alert(a)，输出456
* 例子2：var a=123;function func(){alert(a);var a=456;}func();alert(a);先输出undefined，再输出456。分析：先执行全局func(),然后调用func函数，先看局部有没有声明a，有声明，则alert(a);var a=456;相当于var a;alert(a);a=456;（参考58的2）,因此func函数里的alert(a)输出结果为undefined，再执行a=456，最后执行全局alert(a)，输出结果为456
* 例子3：var a=123;function func(a){alert(a);a=456;}func();alert(a);先输出undefined，再输出123。分析：先执行全局func(),然后调用func(a)函数（注意是有形参的），先看局部有没有声明a，没有声明，执行局部alert(a)时，alert(a)里的a是继承func(a)的形参a，由于形参a没有被实参赋值（参考15），因此alert(a)结果为undefined。然后继续执行局部里的a=456；再执行全局alert(a)，此时的a为全局a=123，和局部a没有联系，则再输出123。
* 例子4：var a=123;function func(a){alert(a);a=456;}func(123);alert(a);先输出123，再输出123。分析：先执行全局func(123),然后调用func(a)函数（注意是有形参的），先看局部有没有声明a，没有声明，执行局部alert(a)时，alert(a)里的a是继承func(a)的形参a，由于形参a继承了实参123，因此局部alert(a)输出结果为123，再执行局部a=456；最后执行全局执行全局alert(a)，此时的a为全局a=123，和局部a没有联系，则再输出123。
* this参数：根据调用的方式不同，this会指向不同的对象。1以函数的形式调用，this永远指向window。2以方法是形式调用，this就是调用方法的那个对象
* 构造函数(首字母大写)：构造函数和普通函数的区别就是调用方式不同：普通函数就是直接调用，而构造函数需要使用new关键字来调用。构造函数让对象有了分类（人类、动物类....）
* 构造函数的执行流程：1立刻创建新的对象。2将新建的对象设置为函数中的this,在构造函数中可以使用this来引用新建的对象。3逐行执行函数中的代码4将新建的对象作为返回值返回。
* 使用同一个构造函数创建的对象，我们称为一类对象，也将一个构造函数称为一个类
* 例子：function Person(){this.name=”孙悟空”;}---------------68的2、3点 var per=new Person()-----68的1、4
* 类：构造函数就是类，类就是构造函数
* 实例：通过构造函数创建的对象
* Instanceof：检查一个对象是否是一个类的实例。语法：对象 instanceof构造函数。如果是则返回true，不是则返回false
* 原型prototype：我们所创建的每一个函数，解析器都会向函数中添加一个属性prototype。这个属性对应一个对象，这个对象就是我们所谓的原型对象。如果函数作为普通函数调用prototype没有任何作用，当函数以构造函数的形式调用时，它所创建的对象中都会有一个隐含的属性，指向该构造函数的原型对象，可以通过\_proto\_来访问该原型属性。（prototype搭配构造函数使用，\_proto\_搭配对象使用）（prototype为原型，\_proto\_用来访问原型）
* 原型对象：相当于一个公共区域，所有一个类的全部实例（对象）都可以访问到这个原型对象，我们可以将对象中共有的内容统一设置到原型对象中。当我们访问对象的一个属性或方法时，它会先在对象自身中寻找如果有则直接使用，如果没有则会去原型对象中寻找，如果找到就直接使用。
* 以后我们创建构造函数时，可以将这些对象共有的属性和方法统一添加到构造函数的原型对象中，这样不用分别为每一个对象添加，也不会影响到全局作用域
* 使用in检查对象中是否含有某个属性。如果对象中没有但是原型中有，也会返回true。如：console.log(“name” in per1);
* 可以使用对象的hasOwnProperty()来检查对象自身中是否含有该属性。如：console.log(per1.hasOwnProperty(“name”));
* 原型对象也是对象，因此它也有原型。当我们使用一个对象的属性或者方法时，会现在自身中寻找，自身中如果有则直接使用，如果没有则去原型对象中找，如果原型对象中有则使用，如果原型对象中没有则去原型的原型中寻找。如果还没有则就继续找，直到找到object对象的原型，而object对象没有原型，如果在object中依然没有找到，则返回null。
* 垃圾回收（GC），回收没用的对象。在JS中拥有自动回收垃圾机制，会自动将这些垃圾对象从内存中销毁。我们不需要也不能进行回收的操作。
* 数组（Array，大写）。创建数组对象：var array=new Array();注意：1不用标明类型。2创建非连续的数组，编译器会将该元素置空。如array[0]=1,array[1]=2,array[2]=3,array[5]=6。遍历查询时显示结果为1,2,3，，，6。中间两个位置会置空
* 获取数组的长度：调用length()函数（既能查询长度也能修改设置长度）。查询长度，如：array.length()注意：1对于连续的数组，使用length可以获取数组的长度（元素的个数）。2对于非连续的数组，使用length会获取到数组的最大的索引+1。修改设置长度，如：数组原来的长度为4，此时修改成array.length=10,再查询长度，array长度变成10。但是注意：增加的数组长度的位置对应的元素会置空，如123，，，，，，，，10(有10个元素)
* 创建数组时就可以初始化指定元素：如var array=new Array(1,2,3,4);
* 使用字面量来创建数组（[]）。如：var array=[]。1：使用字面量创建数组，就可以初始化指定的数组元素。如：var array=[0,1,2,3]
* 字面量创建数组和普通创建数组区别：当初始化只有一个值时，字面量会将其当成一个元素，而普通创建数组会将其当成长度。如：var array=[10];console.log(array)结果为10，而var array=new Array(10);console.log(array)输出结果为，，，，，，，，，，，，(10个位置)。
* 数组中的元素可以使任意类型(包括对象，函数，数组)。如：arrary=[1,”abc”,true,null,undefined,对象]。如array=[func1(),func2(a)]，如：array=[[1,2,3],[1,1,1]]
* Push():该方法可以向数组末尾添加一个或多个元素，并返回数组的新的长度，可以将要添加的元素作为方法的参数传递，这样这些元素会自动添加到数组的末尾。如：array.push(“蜘蛛精”，”白骨精”，”玉兔精”)
* Pop()：删除数组最后一个元素，并将删除的元素作为返回值返回。如：result=array.pop();结果为result=”玉兔精”，并且在数组中删除掉”玉兔精”
* Unshift()：向数组开头添加一个或多个元素，并返回新的数组长度。如：array.unshift(“牛魔王”，”白龙马”)此时数组顺序为：牛魔王、白龙马、蜘蛛精。。。。。
* Shift()：删除数组第一个元素，并将删除的元素作为返回值返回。如：result=array.shift();结果为result=”牛魔王”。并且删掉牛魔王
* Foreach().原型为array（练习day29）。Foreach()方法需要一个函数作为参数。注意：1这种作为参数的函数由我们自己创建但是不由我们调用，称为回调函数。2数组中有几个元素就会执行几次.3每次执行时浏览器会将遍历到的元素以实参的形式传递进来，我们可以来定义形参来读取这些实参内容。3浏览器会在回调函数中传递3个参数：第一个参数就是当前正在遍历的数组元素（value）。第二个参数就是当前遍历的元素索引（index）。第三个参数就是正在遍历的数组（object）。如：arr.foreach(function(value,index,object){console.log(value)});
* Slice(start,end)：start：截取开始的位置（包括该位置）。End：截取结束的位置（不包括该位置）。如：slice(0,2)：意思截取[0]、[1]，注意没有[2]！！！！不包括结束位置的元素
* Slice(start):1只有一个参数，没有end，此时会截取start到最后的元素。如slice(1)，则截取内容为[1]、[2]、[3]........。2Slice()可以传递一个负值，如果传递一个负值，则从后往前计算：-1为倒数第一个，-2为倒数第二个。如slice(1,-2):截取第一道倒数第三个的元素
* Splice()：1两个参数：splice(start,index)：start表示开始位置的索引，index表示删除的元素个数。删除数组中指定的元素，并将删除的元素作删除元素为返回值返回。2三个参数：splice(start,index,新元素):第三个参数会自动插入到start开始索引前边的位置。特别的：如splice(2,0,”红孩儿”):第二个参数为0相当于没有删除元素，而是将”红孩儿”插入到[2]之前
* 数组去重（day29）
* Concat()：连接两个或多个数组（可以直接连接元素），并将新的数组以返回值返回。该方法不会对原数组产生影响。如：arr1、arr2、arr3。Var result=arr1.concat(arr2)；又如var result=arr1.concat(arr2,arr3,”牛魔王”)
* Join():将数组转换成字符串。1该方法不会对原数组产生影响，并把转换后的字符串作为结果返回。如：arr=[“123”,”456”]，console.log(arr)输出结果为”123”,”456”。var reslult=arr.join()，console.log(arr)输出结果为”123, 456”，且用typeof检测结果为string。2在join()中可以指定一个字符串作为参数，这个字符串将会成为数组中元素的连接符。如var reslult=arr.join(“----”)
* Reverse()：用来反转数组（前边的去后面边，后边的去前边）。如：arr.reverse()；
* Sort()；排序函数。注意：1会影响到原数组，2默认按照unicode编码进行排序（不是按照数字大小进行排序）。但是我们可以指定排序的规则：我们可以在sort()添加一个回调函数来指定排序规则，回调函数中需要定义两个形参 ，浏览器将会分别使用数组中的元素作为实参去调用回调函数。使用哪个元素调用不确定，但是肯定的是在数组中a一定在b的前面，浏览器根据回调函数的返回值来决定元素的顺序，如果返回大于0的值则元素会交换位置，如果返回小于0得值则元素位置不变。如果返回一个0则认为两个元素相等，也不交换位置。如：arr=[5,4]; --------传进5和4，输出结果为4,5

arr.sort(function(a,b){

if(a>b)

Return 1;

Else if(a<b)

Return -1;

Else

Return 0;

});

Call和apply、bind没搞懂！！！！

1. Call()和apply():这两个方法都是函数对象的方法，需要通过函数对象来调用。当对函数调用call()和apply()都会调用函数执行。1、func()和func.apply()和func.call()效果一样，都是调用func函数。2在调用call()和apply()可以将一个对象指定为第一个参数，此时这个对象将会成为函数执行时的this，通过call或apply可以改变对象，从而this也跟着被改变。但是func()这种方式this永远是window，但是func.apply()和func.call()可以改变this，你指定是谁就是谁。
2. Call和apply为立即执行函数，bind非立即执行函数
3. call和applay都是改变this的指向来改变内容（第一个参数为需要改变的this指向的内容），不同的是call()的第二个以后的参数为number、string，而apply第二个以后的参数为数组，bind也会改变this的指向，但是不会输出内容，得在后面加()。详细看例子。let person1 = {

        name: "mike",

        age: 18,

        say() {

          console.log(`姓名:${this.name},年龄:${this.age}`);

        },

      };

      let person2 = {

        name: "tom",

        age: 15,

      };

测试输出代码结果：

person1.say()此时输出结果为姓名：mike，年龄：18

person1.say.call(person2);//此时输出的结果为姓名：tom，年龄15

用this代替person2，输出结果一样：person1.say.call(this);//此时输出的结果为姓名：tom，年龄15

person1.say.apply(person2);//此时输出的结果为姓名：tom，年龄15

person1.say.bind(person2);//此时不会输出任何结果

person1.say.bind(person2)();//此时输出结果为姓名：tom，年龄15

      person1.say.call(person2);//此时输出的结果为姓名：tom，年龄15

      更改say(...args) {

        console.log(`姓名:${this.name},年龄:${this.age},参数:${args}`);

        },

      };

      person1.say.call(person2,123,555);输出结果为姓名：tom，年龄15，参数：123,555

person1.say.apply(person2，[123,456]);//此时输出的结果为姓名：tom，年龄15,参数：123,456

person1.say.bind(person2,123,88)()//此时输出结果为姓名：tom，年龄15,参数：123,88

1. Arguments。在调用函数时，浏览器每次都会传递两个隐含的参数：1是函数的上下文对象this，2是封装实参的对象arguments。Arguments是一个类数组对象（类数组对象和数组概念不一样）（它不是数组），它也以通过索引来获取实参数据（arguments[1]获取到第二个实参数据，arguments[0]获取到第一个实参数据}），也可以获取实 参的数量（arguments.length）。在调用函数时，我们所传递的实参都会封装到arguments中。
2. Argument.length：可以用来获取实参的数量。注意：1我们即使不定义形参，也可以通过arguments来使用实参，只不过比较麻烦。Arguments[0]表示第一个实参、Arguments[1]表示第二个实参。。。。2它里面有一个函数对象callee，就是当前正在指向的函数的对象。
3. Date对象（日期）。1如果直接创建时间对象则获取到的是当前时间。2可以设置指定的时间对象，日期格式：月份/日/年 时:分:秒
4. getDate():获取当前日期对象是几号
5. getDay():获取当前日期对象是周几（0表示周日）
6. getMonth():获取当前日期对象的月份（0表示1月，1表示2月）
7. getFullyear():获取当前日期对象的年份
8. getTime():获取当前设置的对象的时间戳。时间戳指的是从1970年1月1日，0时0分0秒到当前日期所花费的毫秒数（1秒=1000毫秒）。计算机底层在保存时间时使用的都是时间戳。如var date1-=new Date();var d1=date1.getTime()；
9. Date.now():获取笔记本当前的时间戳。如var d1=Date.now()
10. Math。Math和其他函数不同，他不是一个构造函数，它属于一个工具类不用创建对象，向上取整（小数位只要有值就进1）、floor()向下取整（去掉）、round()对一个数进行四舍五入、random()生成0~1之间的随机数、生成0~10的随机数：random()\*10、max()可以获取多个数中的最大值。如var max=Math.max(10,45,60);结果为60、min()可以获取多个数中的最小值
11. 包装类：js为我们提供了3个包装类，通过这3个包装类可以将基本数据类型的数据转换成对象。String()可以将基本数据类型转换成string对象，Number()可以将基本数据类型转换成number对象，Boolean()可以将基本数据类型转换成boolean对象
12. 对象转成boolean值都是true。对象除了函数function类型为function其他都为object
13. 一般不使用包装类。方法和属性只能添加给对象，不能添加给基本数据类型。当我们对一些基本数据类型的值去调用属性和方法时，浏览器会临时使用包装类将其转换成对象，然后再调用对象的属性和方法。如：var s=3;s=s.tostring();解析：s为基本数据类型，调用s=3的属性和方法时浏览器会先将s=3包装成number对象，再用这个对象调用tostring()方法
14. charAt():根据索引获取指定的字符，可以返回字符串中指定位置的字符，如：str=”hsahjka”;str.charAt(1)结果为s
15. CharCodeAt()：获取指定位置字符的字符编码（Unicode）。如str=”hsahjka”;str.charCodeAt(1)
16. formcharCode():根据字符编码去获取字符。调用形式：string.formCodeChar(74),结果为H
17. Concat():链接两个或者多个字符串。如str1.concat(“你好”,”再见”)；将str1字符串和“你好”,”再见”连接起来
18. Indexof()：1，一个参数：该方法可以检索一个字符串中是否含有指定内容，如果字符串中含有该内容，则返回第一次出现的索引。如：str1=”shkjhfa”，result=str1.indexof(“h”)，result的结果为1.如果没有找到指定的内容则返回-1。2两个参数：该方法的第二个参数指定开始查找的位置。如：result=str1.indexof(“h”,1)则结果为4，若为result=str1.indexof(“h”,0)则结果为1
19. Lastindexof()：用法和indexof()一样。有一个参数也可以有两个参数，不同的是indexof从前往后找，lastindexof从后往前找
20. Silce(),截取字符串指定内容。用法和数组的silce()一样（可以接受负值参数作为参数）
21. Substring():截取字符串。参数：第一个：开始截取位置的索引（包括开始的位置）、第二个：结束位置的索引（不包括结束位置）。这个方法不能接受负值作为参数，如果传递了负数，则默认使用0。注意substring当第二个参数大于第一个参数则会自动交换参数的位置。当substring只有一个参数时，表示从该参数截取到最后一位字符
22. Split()(默认全局和忽略大小写匹配):将一个字符串拆分成一个数组(需要一个字符串做参数，将会根据该字符串去拆分数组)。如：str1=”abc,def”;，str1.split(“,”);结果为”abc,def”，字面上没变，但是用isarray()函数测试时显示为true，则证明是数组
23. toUpperCase()：将一个字符串转换成大写并赶回
24. Tolowercase():讲一个字符串转换成小写并返回
25. 正则表达式（给计算机看的，一般使用131的方法）：定义了一些字符串的规则。计算机可以根据正则表达式来检查一个字符串是否符合规则并把字符串中符合负责的内容提取出来。创建正则表达式：var reg=new RegExp(“正则表达式”，”匹配模式”)
26. 正则表达式方法：test()：使用这个方法可以检查一个字符串是否符合正则表达式规则，如果符合则返回true，否则返回false。检测方法：根据自己写的正则表达式来检查输入的字符串是否规范。如：有一个正则表达式：var reg=new RegExp(“a”);要检测的：var result=reg.test(“abc”)，结果为true，分析：正则表达式为”a”，要检测的字符串为”abc”，字符串了包含a（有a就行，无论哪个位置，严格区分大小写），因此符合正则表达式返回true。如果检测的字符串为result=reg.test(“bcbc”)，因为没有符合正则表达式a，因此返回false
27. 正则表达式中的匹配模式：”i”：忽略大小写，”g”全局模式，部分大小写。如：var reg=new RegExp(“ba”,”i”)，result=reg.test(“cbAc”);result结果为true。1使用字面量来创建正则表达式更加简单。语法：var 变量= /正则表达式/匹配模式。如var reg=/a/i。2使用构造函数创建正则表达式更灵活。
28. 正则表达式中|、[]表示或者的意思。如：reg=/a|b|c/; [ab]意思就是a|b。 [a-z]任意小写字母. [A-Z]任意大写字母 [A-z]任意字母。如：reg=/a[bcd]e/则输出结果为abe或者
29. 正则表达式与字符串相关的方法。如split()：方法中可以传递一个正则表达式作为参数，这样方法将会根据正则表达式拆分字符串。Var str=”1a2b3c4d5e”;var result=str.split(/[A-z]/)则输出结果为”1,2,3,4,5”
30. Search()：搜索字符串中是否含有指定内容,找到的话则返回第一次找到的下标，如果没有找到任何匹配的子串，则返回 -1。如：str=”hello abc gello abc”;result=str.search(“abc”);result输出结果为1。（abc第一次出现的下标索引为6）。如果result=str.search(“abcd”);result输出结果为-1。他可以接受正则表达式作为参数，然后根据正则表达式去检索字符串。如：str=”hello abc gello abc”；result=str.search(/a[bc]d/);返回-1，正则表达式更灵活。
31. Match():可以根据正则表达式从一个字符串中将符合条件的内容提取出来，找不到则返回null。如str=”1a2b3c4d5e6f”;result=str.match(/[A-z]/);result输出结果为a。注意：1默认情况下match只会找到第一个符合要求的内容，找到第一个以后就停止检索。解决方法：将匹配模式设置为全局匹配模式，这样就会匹配到所有的内容。result=str.match(/[A-z]/g)。2可以为match设置多个匹配模式，可以同时全局模式忽略大小写，顺序无所谓。如：result=str.match(/[A-z]/gi)。3Match()会将匹配到的内容封装到一个数组总返回，即使只查询到一个结果。如：console.log(result[2]);输出结果为”c”
32. Replace():可以将字符串中指定的内容替换成新的内容。需要两个参数：1被替换的内容2新的内容。但是注意：默认只替换第一个，可以通过全局和忽略大小写来改变这个默认。如：str=”1a2b3c4d5e6f”;result=str.replace(/[a-z]/ig,”--”);输出结果为”1--2--3--4--5--6”。result=str.replace(/[a-z]/ig,””);结果则为”123456”
33. Match只能用来获取并显示匹配到的，但是将匹配到的用作其它作用可以用replace来将获取到的匹配内容用做其他内容。如xxx.replace(neg,($0)=>{$0.toUpperCase()})。注意这里的$0表示的就是匹配到的内容
34. 量词：在正则表达式中，可以根据量词设置一个内容出现的次数。1{n}:正好出现n次，如reg=/a{2}/;console.log(reg.test(“aabbcc”))；输出结果为true。注意：量词只对它前边的一个内容起作用。如reg=/ac{2}/;量词2只对前边的字母c起作用。又如reg=/(ac){2}/;加了括号则2对ac起作用，又如reg=/ab{2}c/，则应该为console.log(reg.test(“abbc”))结果才会为true。2{m,n}出现m~n次。3{m,}出现m次以上，4+匹配前面的表达式1次或多次，相当于{1,}，5\*匹配前面的表达式0次或多次，相当于{0,}，6？匹配前面的表达式0次或1次，相当于{0,1}，7^表示开头，匹配后面的表达式，如reg=/^a/表示以a开头的正则表达式，8$表示结尾，匹配前面的表达式，如reg=/a$/表示以a结尾的正则表达式。以a开头或者以a结尾：reg=/^a|a$/;。9.(点)表示除换**行符\n之外的单元字符。9()**标记一个子表达式的开始和结束位置。子表达式可以获取供以后使用。10replace里的$0$1表示第一组第二组。如/()()/g,这里面有两个()则表示有2组表达式，$0表示第一组表达式

1.(?=a) 表示我们需要匹配某样东西的前面。

2.(?!a) 表示我们需要不匹配某样东西。

3.(?:a) 表示我们需要匹配某样东西本身。

4.(?<=a) 表示我们需要匹配某样东西的后面。

**如：**console.log("我是中国人".replace(/我是(?=中国)/, "rr"))//打印出rr中国人

console.log("我是中国人".replace(/(?<=中国)人/, "rr")) //打印出我是中国rr

1. 测试某个手机号例子：var str="15245677988";var reg=/^1[3-9][0-9]{9}$/;console.log(reg.test(str));其中^1[3-9][0-9]{9}$为正则表达式
2. 正则表达式的.表示任意字符，如/./ 。本意为检查一个字符串中是否有. ，但实际上意思变成任意字符。如var reg=/./；console.log(reg.test(“b”))不是.也返回true。解决方法：使用转义字符：\.表示. \\表示\ 如：var reg=/\./；正则表达式则可以表示.
3. 正则表达式中：\w任意字母、数字、下划线。 \W除了字母、数字、下划线。 \d任意的数字 \D除了数字 \s空格 \S除了空格 \b单词边界 \B除了单词边界
4. 测试邮箱号：分析邮件形式：任意字母下划线(.任意字母下划线)@任意字母下划线.任意字母(2-5位)（例子没看完）
5. 宿主对象（DOM/WINAPI）
6. dom（文档流对象模型）通过js操作网页（html、css

）：DOM:document object model：js中通过dom来对html、css文档进行操作，只要理解了dom就可以随心所欲操作web页面了

1. 模型（dom树）：表示节点与节点之间的关系
2. 节点（Node）：构成html文档最基本的单元。文档节点：整个html文档、元素节点：html中的html标签、属性节点：元素的属性、文本节点：html标签中的文本内容
3. 浏览器在加载一个页面时，是按照自上而下的顺序加载的，读取到一行就运行一行，如果script标签写到页面的上边。在执行代码时，页面还没有加载
4. id对应为getelembyid() ,,,class对应为getelemtbyclassname()
5. onload函数（对象为window）：onload事件会在整个页面（dom）加载完成后才触发
6. 获取元素节点：通过document对象调用。1getElementById(),通过id属性获取一个元素节点对象。2通过getElementsByTagName()，通过标签名获取一组元素节点对象，这个方法会给我们返回一个类数组对象，所有查询到的元素都会封装到对象中，即使查询到的元素只有一个，也会封装到数组中返回。3getElementsByName()，通过name属性获取一组元素节点对象（常用于input）
7. innerHtml：通过这个属性可以获取元素内部的html代码。注意：对于自结束标签，这个属性是没有意义（作用）的。如果需要读取元素节点属性，直接使用 元素.属性名。如.name、.gender。但是没有.class，因此可以使用.classname代替
8. Innertext：也可以获取元素内部的文本内容，和innerHTML区别是innertext会自动将html去除
9. 获取元素节点的子节点（通过具体的元素节点调用）：1getelementbytagname()，返回当前节点的指定标签名后代节点。2childnodes,属性，当前节点的所有子节点，注意：childnodes属性会包括文本节点在内的所有节点，根据dom标签，标签间空白文本节点（text）也会当成文本节点，但是childnodes属性在ie8不会将空白文档当成子节点。3firstchild，属性，表示当前节点的第一个子节点（包含空白文本节点）。4lastchild表示当前节点的最后一个子节点。4Children属性获取当前节点的所有子节点（不会将空白文档当成子节点）
10. 获取父节点和兄弟节点（通过具体的元素节点调用）：1parentnode，该属性表示当前节点的父节点。2previoussibling，该属性表示当前节点的前一个兄弟节点。3nextsibling，该属性表示当前节点的后一个兄弟节点
11. 97的自定义函数没怎么看懂！！！！
12. 97的很多个按钮没有完成！！！
13. 按钮选中：xx.checked=true;不选中：xx.checked=false;
14. 在事件的响应函数中，响应函数是给谁绑定的，this就是谁
15. Dom其他的查询方法：
16. Document.body：在document中有一个属性body，它保存的是body的引用。不用再var body=document.getelementbytagname(“body”);直接var body=document.body
17. Documnet.documentelement中保存的是html根标签
18. Document.all：代表页面中所有的元素（标签）。注意：只有长度和元素有用，如var all=document.all;console.log(all.length)，，如果为console.log(all)则返回undefined。又如all[1]（可以为html、title、script、body等等标签）
19. getelementbyclassname()根据class属性值，来查询一组元素点对象。注意：ie8以下不支持
20. Document.queryselector():很强大！！！（同时覆盖了id、class，可以代替getelemtbyname和getelemtbyclassname）。1需要一个选择器的字符串作为参数，可以根据一个css选择器来查询一个元素节点对象如：Document.queryselector(“.box1 div”)。2ie8没有getelementbyclassname，可以用queryselector代替。3使用该方法总会返回唯一的一个元素，如果满足条件的元素有多个，name它只会返回第一个。
21. Document.queryselectorall()：更强大！！该方法和queryselector类似，不同的是它会将符合条件的元素封装到一个数组中返回。即使符合条件的元素只有一个，它也会返回数组。
22. Dom之增删查改
23. 增。1创建元素节点对象：document.createelement()。注意：需要一个标签/元素名作为对象（如div、li等），根据该标签名来创建元素节点对象并将创建好的对象作为返回值返回。。如var li=document.createelement(“li”) 。 2创建文本节点对象：document.createtextnode()。注意：需要一个文本内容作为参数，将会根据该内容创建文本节点并将新的节点返回。如var gztext=document.createtextnode(“广州”)。3appendchild()：向父节点中添加一个新的子节点（只添加，没有说添加到哪儿）。用法：父节点.appendchild(子节点)。如：li.appendchild(gztext)。又如：将该li节点添加到div中的class=”city”中：city.appendchild(li);此时div中会生成<li>广州</li>。4插入节点对象：document.insertbefore()：可以在指定的子节点钱插入新的子节点。语法：父节点.insertbefore(新节点，旧节点)。如将广州节点插入到北京前(class=”bj”)：city.insertbefore(li,bj);。5替换节点对象：replacechild()，语法：父节点.replacechild(新节点，旧节点)。如：将广州替换北京：city.replacechild(li,bj);
24. 向超链接中添加href'属性：xxx.href=”javascript:;”;
25. 删。1有父节点（父元素）：Removechild()。语法：父节点.removechild(要删除的子节点)。如：删除北京节点，city.removechild(bj)。2没有父节点：要删除的子节点.parentnode.removechild(要删除的子节点)
26. 查：查找（读取）元素节点内的html内容：innerhtml 如：var city=document.getelementbyid(“city”);alert(city.innerhtml);
27. 改：1 .innerhtml=xxx 如：var bj=document.getelementbyid(“bj”);bj.innerhtml=”南昌”;
28. 用innerHTML增：city=document.getelementbyid(“city”);city.innerhtml +=”<li>广州</li>”;注意：尽量不要用+=这种方式去写，会覆盖原来的内容（增加了时间），而且有很大隐患
29. 但是一般不会单单使用63这种增或者单单使用67这种增，而是采用折中的方法增：var li=document.getelementbyid(“li”);li.innerhtml=”广州”;city.appentchild(li);
30. tr、td等标签写进table时，尽量写进tbody中，不要写进table里。原因：tr、td和table中间隐藏了tbody。注意要加[0]
31. 使用dom操作css（内联样式）：
32. 改只能通过style，不然js操作css基本都是只读
33. 通过js修改元素的内联样式：语法：yuansu.style.样式名=样式值。注意：1样式名为字符串。如var box1=document.getelementbyid(“box1”);box1.style.width=”200px”;2遇到有减号的要使用驼峰法（去掉减号然后减号后的内容变成大写）：如修改background-color:box1.style.backgroundColor=”yellow”;
34. 通过style属性设置的样式都是内联样式，而内联样式具有较高的优先级，所以通过js修改的样式往往会立即显示。但是如果在样式表（css）中写了！Important，则此时样式会有最高级的优先级，即使通过js也不能覆盖该样式。
35. 通过js读取元素的内联样式：语法：元素.style.样式名。如alert(box1.style.width);
36. 通过style属性设置和读取的都是内联样式，但是无法读取样式表中（css）的样式
37. 获取js中元素当前显示的样式。1语法：元素.currentStyle.样式名。它可以用来读取当前元素正在显示的样式（只有ie有用！！）2在其他浏览器中使用：getcomputedStyle()这个方法来获取js中元素当前显示的样式。这个方法是window的方法，可以直接使用。需要两个参数：第一个参数：要获取样式的元素，第二个参数：可以传递一个伪元素，一般传null。该方法会返回一个对象，对象中封装了当前元素对应的样式。如var obj=getcomputedstyle(box1,null);alert(“obj.width”)，也可以直接应用：getcomputedstyle(box1,null).width;
38. 自定义函数同时兼容正常浏览器和ie浏览器来获取元素当前显示的样式：如 function getStyle(obj,name){if(window.getComputedStyle){return getComputedStyle(obj,null)[name];}else{return obj.currentStyle[name];}}
39. js其它样式相关属性：
40. clientwidth（获取元素可见宽度）、Clientheight（获取元素可见高度）：这两个属性都是只读，返回的是数字，并且不带px，包含的范围是内容区和内边距（padding）。
41. Offsetwidth（获取元素整个宽度）、offsetheight（获取元素整个高度）：只读，返回值为数字，不带px，，包含范围是内容区、内边距、边框
42. Offsetparent：1用来获取当前元素的定位父元素，2用来获取到离当前元素最近的开启了定位（fixed、relative、absolutive等），如果所有的祖先元素都没有开启定位，则返回body
43. Offsetleft，当前元素相对于其定位父元素的水平偏移量。Offsettop，当前元素相对于其定位父元素的垂直偏移量。
44. Scrollwidth:滚动条的滚动宽度。Scrollheight：滚动条的滚动长度
45. Scrollleft：可以获取水平滚动条滚动过的距离。Scrolltop：可以获取垂直滚动条滚动过的距离
46. 如果为表单项添加disabled=”disabled”时则表单项变成不可选用状态。如<input type=”checkbox” disabled=”disabled” />则该点击按钮不能被点击。Disabled的值只有true和false。在js中true设置为不能被点击，false设置为可以被点击。如xxx.disabled=false;
47. Onscroll：该事件会在元素的滚动条滚动时触发。
48. 当满足scrollheight-scrolltop==clientheight;说明垂直滚动条滚到底了
49. 当满足scrollwidth-scrolllef==clientwidth;说明水平滚动条滚到底了
50. Onmousemove:该事件将会在鼠标在元素中移动时被触发
51. 事件函数（将event作为实参）：当事件的响应函数被触发时，浏览器每次都会将一个事件对象作为实参传递进响应函数。在事件对象中封装了当前事件相关的一切信息，比如：鼠标的坐标，键盘哪个按键被按下，鼠标滚动轮滚动的方向
52. 正常浏览器会有event，但是ie8及以下版本没有event，要用window.event表示。记住解决兼容：event=event||window.evnet;
53. ClientX：获取鼠标的x坐标。ClientY：获取鼠标的y坐标。如：event.clientx、event.clienty。注意：Ie8以下版本表示方法为：window.event.ClientX、window.event.clienty
54. PageX、pageY:获取鼠标相对于当前页面的坐标
55. 滚动条滚动的高度兼容问题：chrome浏览器认为滚动条是body的，可以通过body.srcolltop来获取，而火狐等浏览器认为浏览器的滚动条是html的，可以通过documentelement.srcolltop来获取。兼容：var st=document.body.srcolltop||document.documentelement.srcolltop;
56. 事件的冒泡（bubble）(在事件event的基础上发生)：所谓的冒泡指的就是事件定位向上传导，当后代元素上的事件（event）被触发时，其祖先元素所谓相同事件也会被触发，在开发中大部分冒泡都是有用的，如果不希望发生事件冒泡可以通过事件对象来取消冒泡。取消冒泡：event.cancelbubble=true;
57. 事件的委派（利用冒泡）(day07事件的委派) ：将事件统一绑定给元素共同的祖先元素，这样当后代元素上的事件触发时，会一直冒泡到祖先元素，从而通过祖先元素的响应函数来处理事件。事件委派时利用了冒泡，通过委派可以减少事件绑定的次数，提高程序的性能。如day06增删查改的delete删除事件。当添加新员工信息时，还要再添加delete事件。此时可以用事件的委派统一。
58. Event中的target：表示触发事件的对象。如：event.target.classname=”link”;
59. 事件的绑定 （addeventlistener）（同时绑定多个事件）:用xxx.onclick这样的响应函数会覆盖。如：btn1.onclick=function(){alert(1)};btn1.onclick=function(){alert(2)};结果：alert(2)会把alert(1)覆盖，alert(1)不会显示。如果我都要连续显示呢？解决办法：使用addeventlistener()（事件的绑定）。Addeventlistener(),通过这个函数可以为元素绑定响应函数，3个参数：参数1,事件的字符串（如click、dbclick，不要on），参数2,回调函数，当事件触发时该函数会被调用，参数3,是否在补货阶段触发事件，需要一个布尔值，一般都传false。如：btn1.addeventlistener(“click”,function(){alert(1);},false);btn1.addeventlistener(“click”,function(){alert(2);},false);结果：先输出alert(1)，再输出alert(2)，不会覆盖。注意：ie8不支持addeventlistener。因此要用attachevent()来代替。2个参数：参数1,事件的字符串非（如onclick、ondbclick，要on）,参数2，回调函数，当事件触发时该函数会被调用。如attachevent(“onclick”,function(){alert(1);})；attachevent(“onclick”,function(){alert(2q);})
60. addeventlistener()中的this是绑定事件的对象，attachevent()中的this是window。兼容很正常浏览器和ie8：自定义函数，参数：obj（绑定事件的对象）、eventstr（事件的字符串）、callback（回调函数）。自定义函数如：function bind(obj,eventstr,callback){if(obj.addeventlistener){obj.addeventlistener(eventstr,callback,false);}else{obj.attachevent(“on”+enentstr,callback);}}
61. 事件的传播：微软认为从内而外传播（先出现box1，在出现box2，。。。）。传播的3个阶段：捕获阶段、目标阶段、冒泡阶段s
62. Onmousemove：鼠标移动，onmousedown：鼠标按下，onmouseup：鼠标松开。取消这些事件：如xxx.onmousedown=null;xxx.onmouseup;
63. Setcapture：对鼠标相关事件进行捕获。当调用一个元素的setcapture()方法，这个元素将会把下一次所有的鼠标按下相关的事件捕获到自身上
64. Releasecapture()：取消捕获
65. Onmousewheel(),滚轮事件。要求：当鼠标滚轮放在元素上往上滚时元素变短，当鼠标滚轮放在元素上往下滚时元素变长。注意：火狐不支持onmousewheel()，火狐中要使用DOMMousescroll来绑定滚轮事件，注意该事件需要通过addeventlistener()函数来绑定 。
66. Event.wheeldelta：判断鼠标滚轮滚动的方向。注意：1不是wheeldata！！:2火狐不支持wheeldelta事件，火狐用的是detail。如event.detail.3wheeldelta大于0为向上滚，小于0为向下滚
67. 键盘事件：键盘事件一般都绑定给可以获取到焦点的对象或者document
68. Onkeydown：某个键盘被按下。1对于onkeydown如果一直按着不放手，则事件会一直被触发.。2挡onkeydown连续触发时，第一次和第二次之间会间隔稍微长一些，其他的会非常快，这种设计是为了误操作发生
69. onkeyup：某个键盘被松开。需要搭配keycode来使用。Keycode作用为获取按键的编码，通过这个编码可以知道是哪个按键按下。如：if(event.keycode==89){alert(“y按键被按下了”)};当按下y按键时就会弹出框。注意当keycode=0；时表示方向为0，不在移动
70. Altkey、ctrlkey、shiftkey分别实现按下按键alt、ctrl、shift。
71. 实现同时按下ctrl+y：event.ctrlkey&&event.keycode==89。实现shift+y：event.shiftkey&&event.keycode==89。
72. Bom（浏览器对象模型）：通过js操作浏览器。
73. Bom为我们提供了一组对象，用来完成对浏览器的操作。Bom对象：window、navigator、location、history、screen（移动端）
74. Window：代表的是整个浏览器的窗口，同时window也是网页中的全局对象
75. Location：代表浏览器的地址信息，通过location可以获取地址栏信息，或者操作浏览器跳转页面。1直接打印location，则可以获取到地址栏的信息（当前页面的完整路径）。2修改当前页面的路径信息（会自动跳转到改了后的路径）：location=”www.baidu.com”；3assign()：用来跳转到其他页面，作用和直接修改location一样。如location.assign(www.baidu.com”)。3reload()：用于重新加载当前页面，作用和刷新按钮一样，如果在该方法中传递一个true作为参数，则会强制清空缓存刷新页面，如location.reload(true);4replace()：可以使用新的页面来替换当前页面，调用完毕也会跳转页面，注意不会生成历史记录，不可以回退。
76. Navigator：代表当前浏览器的信息，通过该对象可以来识别不同的浏览器。但是我们一般使用useragent来判断浏览器的信息。Useragent是一个字符串，这个字符串中包含用来描述浏览器信息的内容，不同浏览器会有不同的useragent
77. History：代表浏览器的历史记录，可以通过该对象来操作浏览器的历史记录。由于隐私原因，该对象不能获取到具体的历史记录，只能操作浏览器向前或向后翻页，而且该操作只在当次访问时有效。1length属性：可以获取到当次访问的链接数量。如alert(history.length);2back()方法：可以用来回退到上一个页面，作用和浏览器的回退按钮一样。如history.back();3forward()：可以用来跳转到下一个页面，作用和浏览器的前进按钮一样。如history.forword();4go(0：可以用来跳转到指定页面，需要一个整数作为参数。整数为1代表向前跳转一个页面，相当于forword()，整数为2代表向前跳转2个页面，整数为-2代表向后跳转2个页面。如histroy.go(2)意思为向后跳转2个页面，history.go(-2)意思为向前跳转2个页面
78. 定时器（动画效果基本都是用定时器）
79. 定时器之定时调用（可以执行多次）：setinterval()：1可以将一个函数每隔一段时间就执行一次。参数1：回调函数，该函数会每隔一段时间被调用一次。参数2：每次调用的间隔时间，单位是毫秒。2可以将setinterval()作为返回值（数字）使用，类型为number，这个返回值用来作为定时器的唯一标识
80. 停止定时器：clearinterval()（随便传参，不会报错！）：用来关闭定时器，该方法需要一个定时器的标识/返回值作为参数。注意：clearinterval可以接受任意参数（可以使null、undefined），如果参数是一个有效的定时器标识则停止定时器，如果不是一个有效的标识则什么也不做且不会报错。
81. 定时器注意问题：如果多按几次定时器，那么就相当于打开了很多定时器。此时会变得非常快。因为你打开一个新的定时器，定时器标识符会重新获得，你按掉取消暂停慧慧关掉最新的那一个定时器，其他定时器不会关闭。解决办法：在开启定时器之前，先关闭上一个定时器，即在代码执行之前先执行clearinterval()函数。
82. 定时器之延时调用（只会执行一次）：settimeout()。不是马上执行，隔一段时间后再执行。参数：和setinerval()的参数一样。
83. 可以自定义对象的属性。如：box.mynum; box.timer;等等都是自己定义的
84. Js中改变css的属性时，当用传进来的形参代替css属性时不能直接写xx.style.属性，要写成xx.style[属性]。如传进来的实参”left”给形参attr，则写成xx.style[attr];
85. 通过js来修改css的样式，每修改style一次，浏览器就要重新渲染一次。如box.style.width=xxx;box.style.height=xxx;浏览器需要重新渲染2次。这样子性能会比较差，也浪费时间。此时就要通过修改类类增强性能，减少时间
86. 类的操作：修改某元素的class（一类）属性。我们可以通过修改元素的class属性间接修改样式（如宽高背景颜色），这样一来我们只需要修改一次即可同时修改多个样式，浏览器只需要重新渲染页面一次，性能比较好，并且这种方式可以使表现和行为进一步分离。1新的class属性覆盖旧的class属性，如html中有<div class=”b1”><>，在css中有.b1{width:20px,height:20px};再创建一个class：.b2{width:100px,height:100px};js中直接调用box1.className=”b2”,此时box1的class不再是b1,已经变成b2。2新的class属性继承旧的class属性（在旧的属性上增加新的属性，如果新的属性上有旧的属性的样式，则会覆盖旧的属性，没有就增加到旧的属性上）：box1.className +=” b2”。注意b2前要有空格，否则b1b2就合在一起了，要分开！！！
87. 能在class类中改类就尽量在类中改类，尽量不要通过js修改css的style。
88. JSON：JSON就是一个特殊格式的字符串，这个字符串可以被任意语言解析（C/JAVA等）识别，并且可以转换成任意语言中的对象。作用：主要用于数据的交互。Json字符串中的属性名必须加引号(单双都行)。如：var json1=’{“name”:孙悟空,”age”:18}’;
89. Json分类：1对象{} 2数组[]
90. 将json转换成js中的对象，js提供了一个工具类，叫做json，这个对象可以帮助我们将一个json转换成js对象，也可以将一个js对象转换成json。
91. Json转js对象：json.parse()，返回一个js对象。需要一个json字符串作为参数。如var o1=json.parse( json1);console.log(o1.name)输出结果为孙悟空
92. Js对象转json：json.stringify()，返回一个json字符串。需要一个js对象作为参数。如var js1={“name”:孙悟空,”age”:18}；var sr=json.stringify(js1);console.log(str);输出结果为”{“name”:孙悟空,”age”:18}”
93. Ie7及以下不认识json。兼容ie7：使用eval(),这个函数可以用来执行一段字符串形式的js代码，并将结果返回。如：var str=”alert(“hello”)”;eval(str);输出结果为弹出hello框。注意：如果使用eval()执行的字符串中含有{}，它会将{}当成代码块，如果不希望将{}当成代码块解析，则需要在字符串前后各家一个（）。如eval(“(”+str+”)”);

练习中增加的知识点:

1. getElementById():这个方法将返回一个与那个有着给定id属性值的元素节点对应的对象。
2. getElementsByName()(没见过，基本不用):方法可返回带有指定名称的对象的集合。如：获取无序列表中名字name叫“t”的li 的长度：<ul id="purchases">

<li name="t">A tin of beans</li>

<li name="t" class="sale">Cheese</li>

<li name="t" class="sale important">milk</li>

</ul>

alert(document.getElementsByName("t").length);输出结果为3

1. getElementsByTagName():方法返回一个对象数组，每个对象分别对应着文档里有着给定标签的一个元素。
2. getElementsByClassName()(没见过，基本不用):方法返回文档中所有指定类名的元素集合，作为 NodeList 对象。
3. Input中value和placeholder的区别：value直接确定了text的值，不能再改，而placeholder只是text页面上显示了一个值，可以再修改。
4. 表单form中的action，用于指定页面跳转的去处。如点击登录、提交时会调到某个页面
5. Onclick：当鼠标点击时发生事件，onsubmit：当提交时发生事件，onblur：当失去焦点是发生事件，onout：在鼠标移出指定的对象时执行
6. Event.preventdefault()：阻止事件的默认行为
7. Addeventlistener():html操作dom绑定事件
8. Filter():筛选事件
9. math.random()：系统随机生成大于等于0.0且小于1.0的随机double值
10. 判断两个数组是否相等（无论数组里的内容是什么数字字符串啥的都通用），用JSON.stringify：arr1,arr2:if(JSON.stringify(arr1)===JSON.stringify(arr2))
11. 判断一个数不是number类型：1、typeof(num) != 'number' 2、isNaN(num) 为number类型：typeof(num) == 'number'
12. 将数组 arr 中的元素作为调用函数 fn 的参数将数组 arr 中的元素作为调用函数 fn 的参数:fn(...arr)
13. 创建一个函数数组 arr：let arr=()=>{....}（创建一个匿名函数，创建完后赋值给arr）
14. 假设函数数组arr长度为10，运行函数数组arr中的i个函数，结果与i相同：for(let i=0;i<10;i++){arr[i]=()=>(i)
15. 数组方法之reduce：使用语法：

数组.reduce((accumulator,currentvalue,currentindex,array)=>{....（一般函数数体内容为return(accumulator+currentvalue)）},初始值)。解析：初始值为自定义值，作为第一次的accumulator使用，但是可以不设置（为空）。accumulator为相加值，当有初始值时accumulator第一次为初始值，当没有初始值时，accumulator为数组的第一个值，对应的accumulator第二个值为上一个accumulator+数组第二个值的和，以此类推第三个第四个和。currentindex为数组每个值的索引，array为当前执行的数组。例子：function myreduce() {

        [0, 1, 2, 3, 4].reduce((a, b, c, d) => {

          console.log(a + b);

          return a + b;

        }, 10);

      }//输出结果为10,11,13,16,20

1. Array.prototype.slice.call(xxx，bbb)，可以理解为：先调用call改变数组xxx的slice方法的作用域，在特定作用域中去调用slice方法，call（）方法的第二个参数表示传递给slice的参数即截取数组的起始位置。类似的还可以有Array.prototype.slice.apply()、Array.prototype.reverse.call()、Array.prototype.splice.apply()等
2. Tostring（2）,转换成内容为2进制的字符串，tostring（10）：转换成内容为10进制的字符串，。。。
3. 判断字符串的长度：xxx.length();如let str=”10101”,console.log(str.length),输出结果为5
4. 解决两个小数相乘的精度问题：把需要计算的数据乘以10的n次幂，换算成计算机能够精确识别的整数，然后再除以10的n次幂：

function multiply(a, b) {

    a = a.toString();

    b = b.toString();

    var aLen = a.substring(a.indexOf(".")+1).length;

    var bLen = a.substring(a.indexOf(".")+1).length;

    return (a \* Math.pow(10,aLen)) \* (b \* Math.pow(10,bLen)) / Math.pow(10,aLen+bLen);

}

1. 检查一个字符串str是否有数字存在，有则返回true，否则返回false。for(let i=0;i<str.length;i++){if(Number(str[i])){return true}};return false;
2. void 0相当于undefined，但是undefined不一定等于void 0，因为undefined会被重写
3. 数组查找重复的数字和去重：

let findNum = [];

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

          //去重

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

             findNum.push(arr[i]);

           }

          //找出重复值

          if (

            arr.indexOf(arr[i]) != arr.lastIndexOf(arr[i]) &&

            findNum.indexOf(arr[i]) === -1

          ) {

            findNum.push(arr[i]);

          }

        }

1. 递归实现斐波那契：function fibonacci(n) {

//0 1 ,0+1=1,1+1=2,2+1=3,3+2=5

return n<=2?1:fibonacci(n-1)+fibonacci(n-2)

}

1. 按所给的时间格式输出指定的时间：function formatDate(t,str){

var obj = {

yyyy:t.getFullYear(),

yy:(""+ t.getFullYear()).slice(-2),

M:t.getMonth()+1,

MM:("0"+ (t.getMonth()+1)).slice(-2),

d:t.getDate(),

dd:("0" + t.getDate()).slice(-2),

H:t.getHours(),

HH:("0" + t.getHours()).slice(-2),

h:t.getHours() % 12,

hh:("0"+t.getHours() % 12).slice(-2),

m:t.getMinutes(),

mm:("0" + t.getMinutes()).slice(-2),

s:t.getSeconds(),

ss:("0" + t.getSeconds()).slice(-2),

w:['日', '一', '二', '三', '四', '五', '六'][t.getDay()]

};

return str.replace(/([a-z]+)/ig,function($1){return obj[$1]});

}

1. 邮箱格式的正则表达式： reg=/^([\w+\.])+@\w+([.]\w+)+$/
2. 正则表达式中的\s表示空字符串（空白符），\w表示匹配字母或数字或下划线或汉字，\b 匹配单词的开始或结束

1.丢弃小数部分,保留整数部分

parseInt(5/2)

2.向上取整,有小数就整数部分加1

 Math.ceil(5/2)

3,四舍五入.

Math.round(5/2)

4,向下取整

 Math.floor(5/2)

1. 实现数字逆序输出（负数、0、正数任何情况下）：

function myReverse(num) {

        //如果为小数则转成整数；

        num = parseInt(num);

        let newNum = 0;

        let indexNum;

        if (num === 0) {

          console.log("input error！");

        } else {

          if (num > 0) {

            while (num > 0) {

              indexNum = num % 10;

              newNum = newNum \* 10 + indexNum;

              num = Math.floor(num / 10);

            }

            console.log(newNum);

          } else {

            num = Math.abs(num);

            while (num > 0) {

              indexNum = num % 10;

              newNum = newNum \* 10 + indexNum;

              num = Math.floor(num / 10);

            }

            console.log(parseInt("-" + newNum));

          }

        }

      }

1. 原生object.keys()。1传入数组则返回索引：如var arr = ['a', 'b', 'c']; console.log(Object.keys(arr)); 输出结果为'0', '1', '2'。2传入对象则返回属性名：如var obj = { a: 'alive', b: 'bike', c: 'color' }; console.log(Object.keys(obj)); 输出结果为 ['a', 'b', 'c']
2. 获取数组中重复出现次数最多的数字

function findMoreNum(arr) {

        let result = {};

        // 先循环数组，把元素作为key值，元素出现的次数为value值，塞进result对象中

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

          if (!result[arr[i]]) {

            result[arr[i]] = 1;

          } else {

            result[arr[i]]++;

          }

        }

        //此时result结果为{1: 1, 2: 4, 3: 1, 23: 2}

        let keys = Object.keys(result);

        //此时keys的值为result的属性名数组["1", "2", "3", "23"]

        let maxNum = 0; //属性名对应的value值

        let maxVal = 0; //属性名

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

          //result[keys[i]]获取到当前属性名的value值

          if (result[keys[i]] > maxNum) {

            maxNum = result[keys[i]];

            maxVal = keys[i];

          }

        }

        console.log(maxVal);

      }

1. 将-形式字符串变成驼峰形式。思路：遇到-x的就将其获取，然后将-x中的x截取出来再变成大写，再替换掉-x

function toTuoFeng(str) {

        let neg2 = /-\w/g;

        console.log(

          //$0表示根据neg2匹配到的内容，然后将-后面的内容截取用来当做替换的字符

          str.replace(neg2, ($0) => {

            return $0.slice(1).toUpperCase();

          })

        );

      }

      toTuoFeng("get-element-by-id");

1. 判断一个字符串是否为二进制：思路：开头为0或1（^[0-1]），中间为0或1（[0-1]+）,结尾为0或1（[0-1]$）

function matchBit() {

        var str = "10101111";

        var reg = /^[01]+$/g; //或者reg = /^[01]+$/g

        console.log(reg.test(str));

      }

1. 判断一个字符是否为十进制。思路：开头为1-9（^[1-9]），中间判断0-9（[0-9]+），最后判断最后一个字符（[0-9]$）

var str = "21";

        var reg = /^[1-9][0-9]+$/g;

        console.log(reg.test(str));

1. 判断月份（1-12月份）。思路：单数月份为1-9月份（[1-9]）,双数月份为（1[0-2]），中间用|或区分

var str = "10";

        var reg = /[1-9]|1[0-2]/g;

        console.log(reg.test(str));

1. 判断qq号格式，并且最长为13位，最短为6(注意输入的qq号超过13位数也适用)。思路：开头第一位数为1或9（^[1-9]），中间内容为0或9并且最长为13位最短为6位（减掉开头第一位则为12和5）（[0-9]{5,12}），最后一位数为任意数（[0-9]$）

var str = "169426";

        var reg = /^[1-9][0-9]{5,12}$/g;

        console.log(reg.test(str));

1. 判断座机号。前面四位第一位为0后面三位随意，后面八位随意。如0123-01234567。

let str = "7123-02345678";

        let reg = /^0\d{3}-?\d{8}/g;

        console.log(reg.test(str));

1. 强制转换类型：parseInt、parseFloat、number、toString。隐式类型转换：==、===
2. 函数内部声明变量的时候，一定要使用var命令。如果不用的话，你实际上声明了一个全局变量
3. 如何从外部读取函数内的局部变量。在函数的内部，再定义一个子函数。该子函数会获取函数所有属性，再将其通过自函数返回给外部
4. 闭包：39的子函数就是闭包。闭包就是能够读取其他函数内部变量的函数
5. 事件冒泡：从当前触发的事件目标一级一级往上传递，一次触发，知道document位置
6. 事件捕获：会从document开始触发，一级一级往下传递，知道真正事件目标为止。
7. 浮点数的精度问题：原因：js是弱语言，当把数据转成二进制时，js会把无限循环的二进制进行四舍五入，四舍五入后的数据再相加就产生了精度问题。解决方法：将要计算的数据进行升级（乘以10的n次方知道变成整数），最后再除以10的n次方（复原）
8. 计算机网络：osi（7层）：物理层、数据链路层、网络层（ip）、传输层（tcp/ip）、会话层、表示层、应用层（http）。五层协议：物理层、数据链路层、网络层、运输层、应用层。其中物理层主要定义物理设备标准，比如网线、光纤类型，作用是传输比特流；数据链路层将物理层接收的数据进行mac地址封装和解封，这一层主要工作的设备是交换机；网络层主要讲数据进行ip地址的封装和解封，这一层工作的设备是路由器；传输层定义传输数据的协议和端口号；应用层：一些终端的应用（在屏幕上可以看到的东西）
9. Tcp和udp区别：1tcp面向连接，udp不需要；2tcp保证数据的正确性，udp可能丢包；3udp比tcp快（因为udp没有tcp的握手、拥塞等机制）；udp程序结构比较简单（没有握手、拥塞等机制）；4udp用于传输大量数据，tcp用于传输少量数据

普通函数和箭头函数的最大区别：1、this的指向。普通函数绑定自己的this，箭头函数不绑定自己的this，会捕获上下文的this作为自己的this值；2、arguments，每一个普通函数调用后都具有一个arguments对象，用来存储实际传递的参数，但是箭头函数并没有此对象

1. 异步有哪些：1fs文件操作、2数据库操作、ajax请求、定时器
2. Js数据类型（存在于栈中）：6种，number、null、undefined、string、bool、symbol
3. Js引用类型（存在于堆中）：3种，对象、函数、数组
4. 函数防抖（debounce）：事件响应函数在一段时间后才会执行，如果在这段时间内再次调用，则重新计算执行时间；当预定时间内没有再次调用该函数，则执行响应逻辑。
5. 函数节流（throttle）：可以理解为在函数防抖上多加了一个功能：函数节流会预定一个自动执行时间，到时自动执行一次。
6. throttle 和 debounce 是解决请求和响应速度不匹配问题的两个方案。
7. ajax,fetch,axios的区别：1、Ajax 指的是 XMLHttpRequest（XHR），多个请求之间如果有先后关系的话，就会出现回调地狱2、axios用于替换JQuery ajax，支持 Promise API，自动转换JSON数据3、fetch是原生js，没有使用XMLHttpRequest对象。语法简洁，更加语义化， 基于标准 Promise 实现，支持 async/await
8. React性能优化：1、拆分组件是有利于复用和组件优化的。2、{...this.props} (不要滥用，请只传递component需要的props，传得太多，或者层次传得太深，都会加重shouldComponentUpdate里面的数据比较负担。3、绑定this的方式，尽量使用bind在构造器中绑定而不是使用箭头函数
9. React新的生命周期：getDerivedStateFromProps代替componentWillReceiveProps，getSnapshotBeforeUpdate代替componentDidUpdate
10. 基本数据类型不会改变原有的，如

var a = 3;var b = a;

b = 5;

console.log(a); // 3

console.log(b); // 5

1. 引用数据类型会改变原有的，如

var obj1 = {

a: 1,

b: 2,

c: 3}

var obj2 = obj1;

obj2.a = 5;

console.log(obj1.a); // 5

console.log(obj2.a); // 5

1. 浅拷贝：对于浅拷贝而言，就是只拷贝对象的引用，而不深层次的拷贝对象的值，多个对象指向堆内存中的同一对象，任何一个修改都会使得所有对象的值修改，因为它们公用一条数据
2. 深拷贝：深拷贝不会拷贝引用类型的引用，而是将引用类型的值全部拷贝一份，形成一个新的引用类型，使得我们可以多次使用同样的数据
3. 乞丐版的深拷贝（缺陷：不支持undefined、function等类型）：JSON.stringify()+JSON.parse()：

var obj1 = {

a: 1,

b: 2,

c: 3}

var objString = JSON.stringify(obj1);var obj2 = JSON.parse(objString);

obj2.a = 5;

console.log(obj1.a); // 1

console.log(obj2.a); // 5

升级版深拷贝：Object.assign（target，数据）：

var obj1 = {

a: 1,

b: 2,

c: 3}var obj2 = Object.assign({}, obj1);

obj2.b = 5;

console.log(obj1.b); // 2

console.log(obj2.b); // 5

1. http的一些状态码：200（请求成功），202（已接受请求但未处理），204（已接受请求但未返回数据），206（处理了部分请求，返回了部分数据），300（多种选择），301（永久移动到新的url），302（临时移动到新的url），305（使用代理），404（请求出错，服务器无法理解），403（请求成功，但服务器拒绝执行），404（找不到资源），405（请求被禁止），500（服务器出错），501（服务器不支持此请求）
2. 跨域问题：1、通过jsonp跨域。2、document.domain + iframe跨域。3、 location.hash + iframe。4、 nodejs中间件代理跨域（利用node + express + http-proxy-middleware搭建一个proxy服务器。）
3. Cookie、localStorage、sessionStorage三种缓存方式的区别：Cookie顾名思义为小饼干，适合数据小的文件，每次都会携带在http头中；localStorage仅在当前会话有效，当关闭页面或者浏览器后就会清楚；sessionStorage除非手动清楚，不然一直存在
4. 原生js实现ajax+promise：

const ajaxPromise =  (param) => {

  return new Promise((resovle, reject) => {

    var xhr = new XMLHttpRequest();

    xhr.open(param.type || "get", param.url, true);

    xhr.send(param.data || null);

    xhr.onreadystatechange = () => {

     if(xhr.readyState === 4){

      if(xhr.status === 200){

        resovle(JSON.parse(xhr.responseText));

      } else{

        reject(JSON.parse(xhr.responseText));

      }

     }

    }

  })

}

1. js原生循环方法：for、while、for in、for of、
2. Map和foreach区别：map有返回值，foreach没有返回值
3. 相同点：

1都是循环遍历数组中的每一项

2forEach和map方法里每次执行匿名函数都支持3个参数，参数分别是item（当前每一项）、index（索引值）、arr（原数组）

3匿名函数中的this都是指向window

4只能遍历数组

1. 宏任务是由宿主（浏览、node等）发起的，而微任务由JavaScript自身发起。先执行微任务，再执行宏任务
2. 宏任务：定时器等
3. 微任务：Promise等
4. [withRouter（白皮书）的作用和一个简单应用](https://www.cnblogs.com/luowenshuai/p/9526341.html)。作用：把不是通过路由切换过来的组件中，将react-router 的 history、location、match 三个对象传入props对象上
5. Hook组件：用于在函数组件中引入状态管理和生命周期方法、取代高阶组件和render props来实现抽象和可重用性
6. 高阶组件：就是一个函数，且该函数接受一个组件作为参数，并返回一个新的组件。react-redux中的connect就是一个高阶组件
7. Render Props 组件和高阶组件主要用来实现抽象和可重用性。弊端就是高阶组件和 Render Props 本质上都是将复用逻辑提升到父组件中，很容易产生很多包装组件，带来的「嵌套地域」。render props模式可以把特定行为或功能封装成一个组件，提供给其他组件使用
8. Component和PureComponent的区别：React.PureComponent 通过props和state的浅对比（[shallowEqual](https://github.com/facebook/fbjs/blob/c69904a511b900266935168223063dd8772dfc40/packages/fbjs/src/core/shallowEqual.js" \l "L39)）来实现 shouldComponentUpate()，但 React.Component组件有shouldComponentUpdate(nextProps,nextState)生命周期,可以手动比较是否想要的state或props发生了改变，如果是的话返回true，生命周期继续走，不是的话返回false，生命周期停止不更新。
9. 浅对比：通过遍历对象上的键执行相等性，并在任何键具有参数之间不严格相等的值时返回false。 当所有键的值严格相等时返回true。
10. Setstate是异步的。它的第二个参数是一个回调函数，可以用来监听是否设置成功，是否成功渲染
11. Yield：yield只执行右边的语句，不执行左边的语句。