JS中8大数据类型:Number、String、Boolean、Null、undefined、object、symbol(ES6新增的)、bigInt(谷歌67版本中还出现了一种 bigInt,有些人不认可)

内存空间:{

栈内存:用来存放基本数据类型

堆内存:用来存放复杂数据类型

}

检测数据类型:{

1.typeof 数据

2.typeof(数据)

返回值:{

数值:number

字符串:string

布尔:boolean

undefined:undefined

null:object

}

区别:{

typeof 变量名1 + 变量名2->数据类型+变量名2的值

typeof(n1 + n2)->n1+n2的值的数据类型

}

}

数据类型转化:{

转数字:{

1.Number(数据):将数据转化为number类型的值，从头开始进行转化遇到不是数字的值就停止，如果中断就返回NAN，可以转化小数

2.parseInt(数据):将数据转化为number类型的值，从头开始进行转化遇到不是数字的值就停止，如果开头就中断就返回NAN，只能转化整数

3.parseFloat(数据):将数据转化为number类型的值，从头开始进行转化遇到不是数字的值就停止，如果开头就中断就返回NAN，可以转化小数

4.+数据,-数据:将数据转化为number类型的值，从头开始进行转化遇到不是数字的值就停止，如果中断就返回NAN，可以转化小数

5.非加法的数学运算(-,,/):{

*使用:a*1,a-0,a/1

解析规则:和Number一样

}

转字符串:{

string():什么数据类型可以转化{

1.100:100

2.true:true
3.null:null
4.undefined:undefined
}
toString():用来变量名来.操作{
1.null.和undefined不能进行转化
}
}
转Boolean值:{
Boolean():在js中只有5个值为false:0,空字符串,null,NAN,undefined，其他全为true
}
}
}
函数:{
定义:{
1.var fun = function(){};:赋值式函数{
不可以在函数声明前调用该函数，调用此时的该变量为undefined
}
2.function fun(){};:声明式函数{
可以在函数声明前调用该函数
}
}
}
arguments:{
伪数组:{
length:长度
}
在函数中默认存在:作用可以将该函数中的所有的数据取出来
}
预解析:{
1.首先会解析所有定义的变量名，此时所有的变量名的值都为undefined
2.解析所有的声明式函数，并且还会给该变量名定义为一个函数，所有可以使用声明式函数定义的函数可以在声明式函数之前调用
3.先执行同步的代码，将异步代码放到最后执行

```
<!-- 当函数名和变量名重名时，函数名的优先级高于变量名 -->
只有在调用函数时才会对函数中的代码进行预解析，而且每调用一次函数就解析一次，在函数调用时先是形参赋值，然后
```

}
对象:{
删除:delete obj.对象名,
循环遍历:for in 循环:{
for(var key in obj){};=>key表示为该对象中的索引值
}
判断对象中有没有该索引值:in;=>'name' in obj
}
数组:{

<!-- -->
创建:{
    1.var arr = new Array(n);当括号中有一个数字时，表示该数组的长度为n
    2.var arr = [];
}
常用方法:{
    <!-- 以下都是对原数组进行操作 -->
    1.pop():删除数组的最后一个数据=>删除后的数据
    2.push():在末尾增加一个数据，也可以添加多个数据使用逗号分隔=>返回添加后数组的长度
    3.unshift():在数组的最前面添加一个数据，也可以添加多个数据使用逗号分隔=>返回添加后数组的长度 //比pu
    4.shift():删除数组的最前面一个数据=>删除后的数据
    5.reverse():翻转数组
    6.sort():数组排序{
        sort():以ASCII码继续排序
        sort(function (a,b){return a - b})||sort((a,b)=>a-b):升序排列
        sort(function (a,b){return b - a})||sort((a,b)=>b-a):降序排列
    }
    7.splice():{
        1.截取数组:splice(开始索引，[多少个]):从开始索引的位置进行截取，如果没写多少个，默认截取到末尾=
        2.替换数组:splice(开始索引，多少个，替换的值1，替换的值2，。。。):从开始索引的位置进行截取，然后
    }
    <!-- 原数组不变 -->
    1.concat(数据|数组):对数据和数组进行拼接=>返回拼接好的数组
    2.slice([开始索引位置]，[结束索引的位置]):对数组进行截取=>返回截取后的数组{
        一个参数:从开始处到末尾
        0个值:从开始到末尾
        参数可以为负数，就是从后面数
    }
    3.join():将数组按照括号中的符号进行连接成字符串，如果没有符号默认为逗号
    4.indexOf(数据):查询该数据在数组中的位置,从前向后=>返回查询后的位置{
        2个参数:第一个为数据，第二个为一个位置，表示为从什么位置开始先前查询
    }
    5.lastIndexOf(数据):查询该数据在数组中的位置,从后向前=>返回查询后的位置{
        2个参数:第一个为数据，第二个为一个位置，表示为从什么位置开始向后查询
    }
    6.foreach((item,index,arr)=>{}):一个数组的循环遍历=>无返回值:{
        item:此时索引的参数
        index:此时的索引位置
        arr:原数组
    }
    7.map((item,index,arr)=>{}):映射数组,对数组中每个元素进行修改=>返回一个新数组:{
        item:此时索引的参数
        index:此时的索引位置
        arr:原数组
    }
    8.filter((item,index,arr)=>{}):筛选数组,对数组中每个元素进行筛选，满足保留=>返回一个新数组:{
        item:此时索引的参数
        index:此时的索引位置
        arr:原数组
    }
    9.every((item,index,arr)=>{}):只有当数组中的值全部满足为true，有一个不满足返回false=>返回一个Boo

item:此时索引的参数
index:此时的索引位置
arr:原数组
}
10.some((item,index,arr)=>{}):只有当数组中的值只要有一个满足为true，全不满足返回false=>返回一个Bo
item:此时索引的参数
index:此时的索引位置
arr:原数组
}
11.copyWithIn(替换的位置，开始索引，结束索引){
将自身数组的开始索引的位置到结束索引的位置的数组替换从替换的位置开始替换
}
12.fill(要填充的数据，开始索引，结束索引):填充数组=>返回填充完的数组{
从开始索引的位置到结束索引的位置，使用第一个参数将原来的数据进行替换或者填充
第二个参数默认为0，第三个参数默认为末尾
}
13.includes(数据|Infinity):查看数组是否有这个数据=>返回一个Boolean值
14.flat(数值):拍平数组，就是将多维数组进行展开{
默认为1
Infinity:就是无论是多少维的数组都转化为一维数组
}
15.find(item=>{}):找到第一个满足条件的数据=>返回一个数据
16.findIndex(item=>{}):找到第一个满足条件的索引值=>一个索引下标
}

}

字符串:{

```
<!--  -->
1.创建:{
    var str = "dsfklafj";
    var str = new String("fdslkfj");
}
2.属性:{
    length:只读属性，用来查看字符串的长度
}
3.方法:{
    <!-- 所有的字符串方法都不会对字符串进行修改 -->
    1.charAt(数字):查询该数字下标对应的值，如果没有就为空=>返回对应的值或者为空
    2.charCodeAt(数字):查询该数字下标对应的值，如果没有就为空=>返回对应的值转化为对应ASCII码
    3.substr(开始索引位置，截取的个数):截取字符串=>返回截取后的字符串{
        截取的个数默认为末尾
    }
    4.substring(索引的开始的位置，索引结束的位置):截取字符串=>返回截取后的字符串{
            没有结束索引值时默认到结尾
            结束的所有的位置不包括
    }
    5.toLowerCase():将字符串中所有的大写字母转化为小写字母=>转化后的字符串
    6.toUpperCase():将字符串中所有的小写字母转化为大写字母=>转化后的字符串
    7.replace(要替换的字符串，新的字符串):替换字符串，只能替换第一次满足的字符串，**可以放正则表达式**=
    8.concat(字符串):拼接字符串=>返回拼接好的字符串
    9.slice(开始索引，结束索引):截取字符串=>返回截取后的字符串{
        跟substring()区别:slice可以允许出现负数
    }
    10.split(切割符号,多少个):将字符串使用切割符号进行切割成数组=>返回切割后的数组{
        第二个参数没有时默认全部返回
        第二个参数定义时，只返回数组前定义个数的数据作为返回数组
    }
    11.indexOf(字符串片段，开始索引位置):从前到后查询字符串中指定的字符串片段=>返回索引值，没有查询到就
        开始索引位置默认为开头，
    }
    12.lastIndexOf(字符串片段，开始索引位置):从后到前查询字符串中指定的字符串片段=>返回索引值，没有查询
        开始索引位置默认为结尾，
    }
    13.includes(字符串片段):查询字符串中是否包含有该字符串片段=>返回Boolean值
    14.search(字符串片段):跟indexOf()一样当时他没有第二参数，**可以放正则表达式**
    15.match(字符串片段|正则表达式)
    16.trim():去除首尾空格=>去除后的字符串
    17.trimStart():去除开始的空格，别名trimLeft()=>去除后的字符串
    18.trimEnd():去除结束的空格，别名trimRight()=>去除后的字符串
    19.padStart(目标长度，填充字符串):将字符串扩大到目标长度，空白部分在前面填充第二个参数补全
    20.padEnd(目标长度，填充字符串):将字符串扩大到目标长度，空白部分在后面填充第二个参数补全
    21.startsWith(字符串片段):字符串是否以括号中的字符串开头=>返回Boolean值
    22.endsWith(字符串片段):字符串是否以括号中的字符串结尾=>返回Boolean值
}


}
```

JSON格式:{

1.JSON.parse():将json格式的字符串转化为js中的类型
2.JSON.stringify():将JS中的格式字符串转化为json格式的字符串
}
本地存储:{

```
<!--  -->
1.localStorage:永久缓冲{
    1.setItem(key,value):设置缓存内容
    2.getItem(key):获取key对应的value值，如果获取的key没有对应的值返回null
    3.removeItem(key):删除key对应的一行信息
    4.clear():清除所有的数据
}
2.sessionStorage:会话缓存{
    1.setItem(key,value):设置缓存内容
    2.getItem(key):获取key对应的value值，如果获取的key没有对应的值返回null
    3.removeItem(key):删除key对应的一行信息
    4.clear():清除所有的数据
}
共同点:{
    只能存放字符串格式的数据
    要想存放对象要转化为json格式进行缓存
}
```

}
Math:数学方法{

```
<!--  -->
1.random():0~1的随机数
2.round(数字):四舍五入
3.ceil(数字):向上取整
4.floor(数字):向下取整
5.pow(数字,多少次幂):进行幂运算
6.sqrt(数字):平分根
7.abs(数字):绝对值
8.max(数字1,数字2,...)返回数字的最大值,不能传数组
9.min(数字1,数字2,...)返回数字的最小值,不能传数组
10.PI:得到一个近似π的值
```

}
数学进制转换:{

```
<!--  -->
十进制转其他进制:{
    数字.toString(你要转化的进制(2~32))=>以字符串的格式进行返回
}
其他进制转为十进制:{
    parseInt(数字,你把这个数字当成几进制)
}


}
```

小数点的保留:toFixed(你要保留几位小数)=>返回结果的格式为字符串，如果小数位数不够使用0站位

时间对象:{

```
<!--  -->
创建:{
    var date = new Date();得到的是当前终端的时间，就是用户电脑上设置的时间
    **参数的传递**:{
        1.**参数为数字**:{
            参数个数>=2{
                第一个参数:年(下面的参数如果超过指定的范围会自动进位)
                第二个参数:月(0~11)
                第三个参数:日
                第四个参数:时
                第五个参数:分
                第六个参数:秒
            }
            参数个数为1{
                将该参数当做一个时间戳进行设置
            }
        }
        2.**参数为字符串**:{
            'yyyy-mm-dd HH:MM:SS'
            'yyyy/mm/dd HH:MM:SS'
            此时的月份就是从1~12
        }
    }
    **方法**:{
        1.getFullYear():时间对象的年份,类型number
        2.getMonth():时间对象的月份(0~11),类型number
        3.getDate():时间对象的天数,类型number
        4.getDay():时间对象的星期(0~6),类型number
        5.getHours():时间对象的小时,类型number
        6.getMinutes():时间对象的分钟,类型number
        7.getSecondes():时间对象的秒钟,类型number
        8.getMilliSecondes():时间对象的毫秒,类型number
        7.getSecondes():时间对象的秒钟,类型number
        9.getTime():时间对象的时间戳,类型number
        10.setFullYear(N):设置时间对象的年份,类型number
        11.setMonth(N):设置时间对象的月份(0~11),类型number
        12.setDate(N):设置时间对象的天数,类型number
        13.setHours(N):设置时间对象的小时,类型number
        14.setMinutes(N):设置时间对象的分钟,类型number
        15.setSecondes(N):设置时间对象的秒钟,类型number
        16.setMilliSecondes(N):设置时间对象的毫秒,类型number
        17.setSecondes(N):设置时间对象的秒钟,类型number
        18.setTime(N):设置时间对象的时间戳,类型number
        <!-- 在该方法中添加UTC:getUTCMonth -->
    }
    **时间差**:{
        两个时间对象可以相减，但在IE低版本中不能
    }
}
```

}

定时器:最小捕获毫秒为16毫秒{

```
<!--  -->
setTimeout(函数，时间:毫秒级):炸弹定时器=>页面的第几个定时器
setInterval(函数，时间:毫秒级):间隔定时器=>页面的第几个定时器
**定时器关闭**:{
    clearInterval(第几个定时器):可以混用
    clearTimeout(第几个定时器):可以混用
}
```

}

BOM:browser object model浏览器对象模型{

```
<!--  -->
BOMA的顶级对象window
**浏览器尺寸**:{
    innerWidth:浏览器的宽度(包含滚动条)
    innerHeight:浏览器的高度(包含滚动条)
}
**浏览器的弹出层**:{
    1.alert():警告框，弹出一个提示文本和一个确定按钮=>无返回值
    2.confirm():选择框，弹出一个提示文本和一个确定按钮和一个取消按钮=>返回一个Boolean值
    3.prompt():输入框，弹出一个提示文本和一个输入框，有确定和取消两个按钮=>点确定为文本框中的内容，点取
    <!-- 三个弹出层会阻断程序进行 -->
}
**浏览器的地址栏**:{
    location:{
        1.hash:当前页面的hash值
        2.href:当前地址栏地址:可读可写，写就是跳转页面
        3.search:查询字符串中的值(queryString)
    }
    hostory:{
        length:有多少条历史记录
        back():回退到上一次历史记录
        forward():前进到下一条历史记录
        go(整数):正整数前进，负整数后退，0刷新页面
    }
    navigator:{
        userAgent:浏览器的版本以及浏览器的型号
        appName:浏览器的名字，除了IE低版本是IE自己其他都是netscape
    }
    浏览器事件:{
        onload:资源加载完毕后才会触发
        onscroll:滚动条滚动就触发
        onresize:只要屏幕大小改变就触发{
            1.scrollTop:可读可写属性，浏览器此时卷去的高度{
                在HTML或者在body标签中使用
                <!-- JS中的body:document.body -->
                <!-- JS中的html:document.documentElement -->
            }
            2.scrollLeft:可读可写属性，浏览器卷去的宽度{
                在HTML或者在body标签中使用
                <!-- JS中的body:document.body -->
                <!-- JS中的html:document.documentElement -->
            }
            3.scrollTo(x坐标,y坐标):浏览器定位到某个位置{
                通过window对象来调用
                第三个参数:决定定位方式behavior:[instant:默认|smooth:平滑滚动]一般使用较少，都是通过
            }
        }
    }
}
```

}
DOM:文档对象模型(document object model){

```
<!--  -->
标签获取:{
    1.非常规标签:{
        html:document.documentElement
        head:document.head
        body:document.body
    }
    2.常规标签:{
        getElementById():通过ID来进行查询
        querySelect():查找一个标签
        querySelectAll():查找多个标签
    }
    3.标签属性操作:{
        1.原生属性:{
            style,id等等等:直接点操作
            <!-- 但是由于class为JS中的关键字，使用操作class属性改为className -->
        }
        2.非原生属性:{
            1.getAttribute(属性名):得到属性名中的值
            2.setAttribute(属性名，值):设置属性名和属性值，得到值的类型为字符串
            3.removeAttribute(属性名):移出某个属性
        }
        3.H5自定义属性:{
            只能获取到以data-开头的属性值
            通过标签名.dataset来获取一个对象包含data-中的所有属性值，对象中的属性名去掉了data-
            删除一个对象属性:使用对象中的delete
        }
        4.类名{
            1.通过原生属性的方法
            2.classList:{
                1.add(类名):增加一个类名
                2.remove(类名):删除一个类名
                3.toggle(类名):切换类名，当该标签中包含有该类名就删除，没有就添加
            }
        }
    }
    4.操作文本内容:{
        1.innerHTML:读写属性，读，获取到文本中的所有内容，写会解析写中包含的标签元素
        2.innetText:读写属性，读，获取文本中的所有内容，写，自会将文本的内容一模一样转化到屏幕上
        3.value:只能使用在表单文本上，就是input标签
    }
    5.style属性值:{
        1.行内样式{
            通过.style只能获取和设置行内样式，获取不到非行内样式，所有一般使用.style来设置样式值，不用s
        }
        2.非行内样式:{
            getComputedStyle(标签):标准浏览器可以使用，获取属性
            currentStyle:IE低版本使用,通过标签点来实现
        }
    }
    6.节点:{
```

```
        <!-- 节点分为:元素节点=1，节点名大写，属性节点=2，属性名，文本节点=3，#text，注释节点=4，#comm
        <!-- 节点编号:通过每个节点中nodeType属性值得到的 -->
        <!-- 节点名:通过每个节点中nodeName属性值得到的 -->
        <!-- 获取元素节点 -->
        1.children:得到的是所有的子元素节点
        2.firstElementChild:获取第一个子元素节点
        3.lastElementChild:获取最后一个子元素节点
        4.previousElementSibling:获取到上一个元素兄弟节点
        5.nextElementSibling:获取到下一个元素兄弟节点
        6.parentNode:父亲元素节点
        7.parentElement:父亲元素节点
        <!-- 获取所有节点 -->
        1.childNodes:获取所有子节点
        2.firstChild:获取第一个节点
        3.lastChild:获取最后一个节点
        4.previousSibling:获取到上一个兄弟节点
        5.nextSibling:获取到下一个兄弟节点
        <!-- 获取属性节点 -->
        1.attributes:获取到所有的属性节点
        <!-- 创建元素节点 -->
        1.document.createElement(元素节点名)

        节点的增删改查:{
            <!-- 插入一个节点 -->
            1.appendChild(元素节点名):插入一个字节点
            2.insertBefore(新元素节点名，旧元素节点名):将新元素节点插入到旧元素节点前面
            <!-- 删除一个节点 -->
            1.removeChild(子元素节点):将子元素节点从父元素节点中删除
            2.remove():把自己移出
            <!-- 改|替换节点 -->
            1.replaceChild(新元素节点，旧元素节点):用新的元素节点替换旧的元素节点
            <!-- 节点的克隆 -->
            1.cloneNode():将自己克隆一份，参数:true：就是克隆子节点，false:不克隆子节点(默认为false)
        }
    }
 }

}
```

打印:{

console.log():常用，

console.dir():详细展开

console.table():一般用于对象中

}

自我认知:{

使用在书写js代码时，就相当于在window的类中写代码

}