JavaScript

基础操作

- 创建对象

```
let obj= {}
```

- 新增属性

```
obj.a = 1
```

- 修改属性

```
obj.a = 2
```

- 查询属性

```
obj.a
```

- 删除属性

```
delete obj.a
```

- 包含includes

```

let user = {name:'张三',age:20}
user.name.includes('张') //返回true
```

数组操作

- 数组创建

```
// 常见方式
let  arr = []
// new一个对象的方式
let arr1 = neww Array()
```

- 数组赋值

```
 let  arr = []

 // 数组内的值的类型可以是任何东西，且可以混合
 arr[1] = 'abc'
 // 注意，数组arr从下标0开始，所以此时会自动补全arr[0]的值，使其为undefined（空）

 // 也可以用这种方式插入
 arr.[0] = 1

 // 或者push,在尾部添加值
 arr.push('xyz')
```

- 数组长度

```
arr.length
```

- 数组元素删除

```
 // 方法1
  // 语法：arr.splice(a,b) ,从a位置开始，删除b个元素
  // 比如下面这个就是从0开始，删除1个元素，也就是删除index = 0处的元素
 arr.splice(0,1)

 // 方法2
  // 直接从头部（第一个元素）抛出
 arr.shift()

 // 方法3
  // 直接从尾部（最后一个元素）抛出
 arr.pop()
```

- 截取数组

```
// 语法：arr.slice(a,b) ,从a位置开始，b位置结束，左闭右开
// 比如下面这个就是从0开始，截取到index = 1的元素
arr.splice(0,2)

// arr.splice(a) 也可以这样，从a开始，切片完整个数组
arr.splice(1)
```

- 合并数组

```
// 语法结构：arr1.concat(arr2)
let  arr1 = [1,2]
let  arr2 = [3]
let  arr3 = arr1.concat(arr2)

// 结果为 [1,2,3] ,把第一个数组的元素放在前面
```

- 字符串转数组

```
// 语法结构：str.split('') ,其中''之间放的是分割的方式，和python的split相同
let str = 'abcde'
str.split('')  // 按照空字符来分割，也就是分割每一个单词（或者汉字），结果为['a','b','c','d','e']

let str = 'abc,de'
str.split(',')  // 按照','来分割，结果为['abc','de']
```

- 数组转字符串

```
// 语法结构：arr.jion()  ,其中括号内之间放的是连接字符的方式，默认是逗号','
let arr = ['a','b','c','d','e']
arr.jion()  // 结果为 a,b,c,d,e
arr.jion('|') // 结果为 a|b|c|d|e
```

- 数组排序

```
 // 数组排序，也就是sort功能存在问题，该方法是按照unicode编码进行排序的
 // 比如下面这个结果就不是正常的
 let arr = [1,2,10]
 let arr1 = arr.sort()
 arr1 // [1,2,10] -> [1,10,2]

 //为了实现正序功能，需要添加回调函数  arr.sort((a,b) => a-b)
 let arr = [1,2,10,7]
 let arr1 = arr.sort((a,b) => a-b)
 arr1 // [1,2,10,7] -> [1,2,7,10]

 //为了实现逆序功能，需要添加回调函数  arr.sort((a,b) => b-a)
 let arr = [1,2,10,7]
 let arr1 = arr.sort((a,b) => b-a)
 arr1 // [1,2,10,7] -> [10,7,2,1]
```

- 数组倒转

```
// 注意，数组倒转，不是逆序排序，而是完全将数组元素颠倒
// arr.reverse()
let arr = [1,2,10,7]
let arr1 = arr.reverse()
arr1 // [1,2,10,7] -> [7,10,2,1]
```

- 获取index

```
// ARR.indexOf(x) ，获取数组中x第一次出现的位置
let arr = ['a','b','c','d','e']
arr.indexOF('b') // 结果为1（从0开始）

// 若没有该元素，则返回-1，不会报错

// ARR.lastindexOf(x) ，获取数组中x最后一次出现的位置，和indexOf同理
```

- 筛选

```
// 语法结构：arr.filter(condition)，其中condition为筛选函数

// 前面说过，数组中可存放任何元素，那么数组[]中可以存放对象{}
// 比如：
let users = [{name:'张三',age:20},{name:'李四',age:21}]

// 此时我想要进行筛选操作，将我需要的对象筛选出来，比如要求年龄20岁以上的
users.forEach(
    user=>{ if (user.age>20){ console.log(user) } }
)
// 结果能够输出{name:'李四',age:21}，但是显然很麻烦

//这里使用filter，可以轻松完成上述要求
users.filter(user => user.age > 20)
```

- 数组对象元素查询

```
// 想要通过一些条件找到数组中某个元素或者对象
// 使用arr.find(condition)
let users = [{name:'张三',age:20},{name:'李四',age:21}]
user.find(user => user.name === '李四') // 结果为{name:'李四',age:21}
```

- 数组对象元素下标查询

```
// 想要通过一些条件找到数组中某个元素或者对象的下标
// 使用arr.findIndex(condition)
let users = [{name:'张三',age:20},{name:'李四',age:21}]
user.findIndex(user => user.name === '李四') // 结果为 1
```

- 多对象内元素读取 Map

```
let users = [{name:'张三',age:20},{name:'李四',age:21}]
users.map(user =>  user.name+'hhh')  //结果为{'张三hhh','李四hhh'}
```

- 总和操作 reduce

```
let users = [{name:'张三',age:20},{name:'李四',age:21}]

// 求和实现 ， 计算所有人的年龄之和
let sum = users.reduce((pre,current) => {
    return pre + current.age  // pre会保存每一次计算所得到的值，current代表当前传进来的对象
},0) // 这个0表示初始值以及数据类型，此处表示pre从0开始


// 统计每个人出现的次数
let countUser = users.reduce((pre,current) => {
    if (current.name in  pre){
        pre[current.name]++
    } else {
        pre[current.name] = 1  //这里是对象的赋值操作 obj['x'] = a
    }
    return pre  // pre会保存每一次计算所得到的值，current代表当前传进来的对象
},{})  //这一行的{}表示pre初始为一个空对象
```

其他操作

- 属性赋值

```
// 使用中括号赋值时，需要使a为字符串
obj['a'] = 1

// 若 obj中存在数字属性，比如
obj = {1:'a',2:'b'}
//需要通过如下方式访问
obj[1] //不要添加引号
```

- 浅拷贝 (对赋值变量的操作会传递给原始变量)

```
let obj1 = obj
```

- 深拷贝（对赋值变量的操作不会传递给原始变量）

```
// 方法1
let obj1 = {}
Object.assign(obj1,obj)

// 方法2
let objStr = JSON.stringify(obj) //将obj转换为字符串
let obj1 = JSON.parse(objStr) //赋值
```

- 保护性赋值

```
// 当从数据库读取值时，若值为null或者undefined，就可能会引发报错，为了避免这种情况，我们使用 ?.语法
// 语法结构：b = a?.x ，若a为null或者undefined，b不报错
let obj = undefined //obj的值为null也是同样的效果
let obj1 = obj['a']  //报错
let obj2 = obj.a //报错
let obj3 = obj ?.a //不会报错
```

```
// 当从数据库读取值时，若值为null或者undefined，就可能会引发报错，为了保证赋值，我们使用 ?? 语法
// 语法结构：b = a??c ，若a为null或者undefined，c赋值给b
let obj = undefined //obj的值为null也是同样的效果
let obj1 = obj  //报错
let obj2 = obj //报错
let obj3 = obj ?? obj4 //不会报错，obj4赋值给obj3


// ??语法还有一种操作方式：  a??=b
// 这种方式用于不赋值直接输出，即a为null或者undefined，输出b
console.log(obj ??= obj4)
//结果为 obj4

```

```
// 还有一种实现和上面的类似，但是更广泛。
// 语法结构  a||b
// 这种语法区别于上一个的点在于，若a为false，则返回b，若为true，则返回a。判定范围比??广泛，不局限于null和undefined
```

- 循环访问对象的属性

```
obj = {1:'a',2:'b'}
for (const key in obj){
    // 此时key会将数字变为字符串
    console.log(key)
}
```

- 断点/debubg

```
obj = {1:'a',2:'b'}
for (const key in obj){
    debugger  //用于实现断点操作，程序运行到此处会停止且在网页中展示变量
    console.log(key)
}
```

JSON

可用如下网页验证是否为合法json，或者用该网页查看json结构：JSON在线解析及格式化验证 - JSON.cn

json和JavaScript很相似，区别在于json对象的key必须要用双引号包裹

```
obj = {"1":"a","2":"b"}
```

此外，json的对象可以无限套娃

```
{"name":"张三","child":[
    {"name":"小张三","child":[
        "name":"小小张三","child":[
            "name":"小小张三","child":[]}
        ]}
    ]}
]}
```

还有一点需要注意，json数组中最好包含元素