数组

数组：是按次序排列的一组值，每个值的位置都有编号（从零开始），整个数组用方括号表示 var arr = ['sxt','bsizhan','it']

位置也被称为下标

数组可以在创建时赋值也可以先定义后赋值

var names = []; names[0] = "frank"; console.log(names);

任何类型的数据都可以放进数组

var into = [20,"iwen",true];

多维数组：数组里面嵌套数组

var Arr = [100,200,["iwen","frank"]]; console.log(Arr);//[100,200,Array(2)]

下标的读取

var user = ["iwen",:ime","frank"]; console.log(user[0]);//frank

length属性

console.log(user.length);//3 console.log(user[5]);//数组越界undefined

数组的遍历:循环数组中的每一个值

```
var username = ["iwen","ime","sakuar","frank"];
for(var i = 0;i<4;i++){
console.log(username[i]); }
数不过来时
for(var i = 0;i<username.length;i++){
console.log(username[i]); }
或 var i = 0; while(i<username.length){//i=0,1,2,3
console.log(username[i]);
i++;}
for(var i in username){
console.log(username[i]);}
```

数组静态方法Array.isArray()

返回一个布尔值，标识参数是否为数组，弥补typeof运算符的不足
var arr = [10,20,30]; console.log(Array.isArray(arr));
返回true是数组，返回false不是数组

push()/pop()

push:用于在数组的末端添加一个或多个元素，并返回添加新元素后的数组长度（该方法会改变原数组）

```
var arr = ["hello"];
arr.push("hi");
console.log(arr);//['hello','hi']
console.log(arr.push("ha"));//3
```

可以加多条var arr = ["hello"];

```
var mylength = arr.push("world",100);
console.log(arr);
console.log(mylength);
```

pop:用于删除数组的最后一个元素，并返回该元素(原数组被改变)

```
var arr1 = [100,200,300,400];
arr1.pop();
console.log(arr1);
```

shift/unshift

shift方法用于删除数组的第一个元素并返回该元素（改变原数组）

```
var names = ["iwen","ime","frank"]
var result = names.shift();
console.log(result);
//清空数组元素
var arr1 = [100,200,300];
var item;
while(item = arr1.shift()){
console.log(item);//100 200 300(100为true,删到空为undefined即false,不再打印)
console.log(arr1);//[]
}
```

unshift用于在数组的第一个位置添加元素，并返回添加元素后的数组长度，该方法会改变原数组

```
var arr = [200];
arr.unshift(100,120,140);
console.log(arr);//[100,120,140,200]
```

join

以指定参数作为分隔符，将所有数组成员连接为一个字符串并返回，如果不提供参数，默认用逗号分隔，但本身还是数组

```
var arr = [10,20,30,40];
arr.join();//10,20,30,40
arr.join("|");//10|20|30|40
arr.join("");//10203040
arr.join(" ");//10 20 30 40
```

如果数组成员是undefined或null或空位，会被转换成空字符串

```
var arr = [10,20,30,40,null,undefined];
console.log(arr.join());//10,20,30,40,,
```

数组的join配合字符串的split可以实现数组与字符串的互换

```
var arr1 = ["hello","world"];
console.log(arr1.join(" "));//hello world
var result = arr1.join(" ");
console.log(result.split(" "));//['hello','world']把字符串转变为数组
```

concat()

用于多个数组的合并，它将新数组的成员添加到原数组成员的后部，然后返回一个新数组，原数组不变

```
var arr1 = ["上","学"]
var arr2 = ["sxt","itbaizhan"]
console.log(arr1+arr2);//上,学sxt,itbaizhan(变成字符串了)
console.log(arr1.concat(srr2));//['上','学','sxt','itbaizzhan']
```

除了数组作为参数，concat也接受其他类型的值作为参数，添加到目标数组尾部

```
var arr4 = [10];
console.log(arr4.concat(20,30,40));//[10,20,30,40]
```

应用场景:上拉加载，合并数据

reverse()

颠倒排列数组元素，返回改变后的数组，该方法将改变原数组

```
var arr = [1,2,3,4,5];
arr.reverse();
console.log(arr);//[5,4,3,2,1]
```

该方法可实现字符串的翻转

```
var str = "helloworld";
var result = str.split("");
result.reverse();
console.log(result.join(""));//dlrowolleh
```

indexOf()

返回给定元素在数组中第一次出现的位置，如果没有出现则返回-1

```
var arr = [10,20,30,40,50,30];
console.log(arr.indexOf(30));//2
if(arr.indexOf(30)>-1){
console.log("存在");
}else{console.log("不存在")}
```

indexOf还可以接受第二个参数，表示搜索的开始位置（包括该位置）

```
console.log(indexOf(20,3));/-1
console.log(indexOf(30,3));/5
```

slice()

截取Array的部分元素，然后返回一个新的Array

```
let arr = ['A','B','C','D','E','F','G'];
rr.slice(0,3);//从0到3，但不包括3['A','B','C']
```

arr.slice(3);//从3到结束['D','E','F','G']
slice()从头到尾截取所有元素，用于复制数组
let aCopy = arr.slice();
console.log(aCopy);//['A','B','C','D','E','F','G'];
console.log(aCopy === arr);//false

拓展运算符spread

是三个点，将一个数组转为用逗号分隔的参数序列,逗号不显示

1. 遍历

   ```
   var arr = [10,20,30,40];
   console.log(...arr);//10 20 30 40
   ```

2. 替代apply方法

   ```
   console.log(Math.max.apply(null,arr))//把max功能赋给数组
   console.log(Math.max(10,20,30,40));//数据格式可以得到40
   console.log(Math.max(..arr));//把数组变成数据格式得到40
   ```

3. 数组合并

   ```
   var arr1 = [10,20,30,40];
   var arr2 = [50,60,70,80];
   console.log([...arr1,...arr2]);//[10,20,30,40,50,60,70,80,]
   ```

Array.from()

将类数组转换为真正的数组
常见类数组：arguments读取当前参数，元素集合，类似数组的对象

1. arguments
   function add(){
   console.log(arguments);
   }
   add(10,20,30);
   function add(){
   console.log(arguments[0]);//10
   }
2. 类数组，伪数组只能使用数组的读取方式和length属性，不能使用数组方法
   function add(){
   console.log(arguments);
   var result = Array.from(arguments);
   result.push(40);

```
console.log(result);
}
add(10,20,30);
```

3. 元素集合

```
let titles = document.querySelectorAll("h3");
console.log(titles[0]);
console.log(Array.from(titles));
```

4. 伪数组对象

```
var user = {
"0":"iwen",
"1":"20",
"2":"男",
length:3 }
console.log(user["0"]);
console.log(Array.from(user));
```

Array.of()

将一组值转化为数组
Array.of(10,20,30);//[10,20,30];
Array(3);开辟三个新空间[空属性*3]

函数

是一段可以反复调用的代码块

function命令：

function命令声明的代码区块，就是一个函数，function命令后面是函数名，函数名后面是一对圆括号，里面是传入函数的参数，函数体放在大括号里面

```
function add(){
console.log("结果");
}
add();//函数调用:函数名+();有了函数调用才打印结果
add();
add();//函数可以反复调用得到复制的效果
function add(){
var x = 10;
var y = 20;
console.log(x+y);
}
add();
```

函数名的提升，可以先调用后创建

```
add();
function add(){
var x = 10;
var y = 20;
console.log(x+y);
}
```

函数参数：

函数运行的时候，有时候需要提供外部数据，不同的外部数据会得到不同的结果，这种外部数据就叫参数

```
function add(x,y){
console.log(x+y);
}
add(10,20);//调用时要传参数
```

函数返回值

：JavaScript函数提供两个接口实现与外界的交互，其中参数作为入口，接收外界信息，返回值作为出口，把运算结果反馈给外界

```
function add(x,y){
return x + y//可以利用return返回结果赋值给变量
}
var result = add(10,20);
console.log(result);
```

var 变量名 = 函数名（赋值的参数），return啥返回啥结果

在{}中return后面不能再添加任何代码，因为不会执行

```
function add(x+y){
console.log(111);//执行
return x + y
console.log(111);//不执行
}
```

箭头函数：

```
var fn3 = (x,y) => x + y(返回值)
console.log(fn3(10,20));
```

1.一个参数

```
var fn3 = x => x
```

2.无参数

```
var fn4 = () => 10
```

3.多个参数

```
var fn4 = (x,y) => x + y
```

如果箭头函数的代码块多于一条语句，就要使用大括号将它们括起来，并且使用return语句返回

```
var fn6 = (x,y) => {
var z = 10;
return x + y + z;
```

}
由于大括号被解释为代码块，所以如果箭头函数直接返回一个对象，必须在对象外面加上大括号，否则会报错
var fn8 = () => ({x:10,y:20})
箭头函数的一个用处是简化回调函数（匿名函数）
var arr = [10,20,30];
var arr = [10,20,30];
arr.map((element,index)=>{console.log(element);})
map会遍历element 打印出10 20 30
使用注意点：(没懂)对于普通函数来说，内部的this指向函数运行时所在的对象，但是这一点对箭头函数不成立，它没有自己的this对象，内部的this就是定义上层作用域的this
var name = "ime"
var user = {
name:"iwen",
age:20,
getName(){
setTimeout(() => {
console.log(this.name);
})
}
}
user.getName();//iwen

set

set是一种数据结构，用于存储唯一的值集合，类似于数组，但是成员的值都是唯一的，没有重复的值
1.add()添加
var mySet = new Set();
mySet.add(100);
console.log(mySet);//Set(1){100}
2.delete()删除
mySet.delete(100);
console.log(mySet);//Set(0){size:0}
3.has()判断该值是否为set成员
mySet.add(200);
var flag = mySet.has(200);
console.log(flag);//true
4.clear()清除所有成员
mySet.clear();
console.log(mySet);//Set(0){size:0}
例：var s = new set();
var arr = [10,20,30,40,50];
arr.forEach(x => s.add(x))//x是参数，循环向集合里添加数字，x代表这里边的每一条数据,forEach每次遍历的值再当做箭头函数参数
console.log(s);//Set(5){10,20,30,40,50}
set函数可以接受一个数组作为参数

```
var s = new Set([10,20,30]);
console.log(s);//Set(3){10,20,30}
```

数组去重的方法

```
var arr = [10,20,30,10,20];
var s = new Set(arr);
console.log([...s]);//把集合变成数组
```

s.size返回集合长度