js第十天

数组快速排序

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scal
e=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
  <script>
    // 快速排序:
    // 1.每次从数组中取出一个基准值mid,
    // 2.比它小的放在左边的数组中 var left = []
    // 3.比它大的放在右边的数组中 var right = []
    // 再继续对 left数组和right数组 重复上述操作

    // splice(x,n)
    // 从索引x处开始，删除n个
    // 返回值：删除后的内容 组成的新数组
    // 修改原数组

    // 获取中间项
    // midIdx = Math.floor(arr.length / 2 )
    // mid = arr.splice(midIdx, 1)[0]
    // arr.splice(3, 1) => [12] => 12
    // var arr = [15, 3, 6, 12, 11, 38, 19]


    // 递归
    // 一定要有结束条件
    // 确定它的核心功能

    // 快速排序:
    // 1.每次从数组中取出一个基准值mid,
    // 2.比它小的放在左边的数组中 var left = []
    // 3.比它大的放在右边的数组中 var right = []
    // 4.最后通过concat将最终结果连接到一起
    function quickSort(ary) {
```

```
        // 结束条件
        if (ary.length <= 1) {
          return ary
        }
        // 从数组中获取基准值
        var midIdx = Math.floor(ary.length / 2)
        var mid = ary.splice(midIdx, 1)[0] // [12]

        var left = []
        var right = []
        // 比基准值小的放在left里面，大的放在right里面
        for (let i = 0; i < ary.length; i++) {
          ary[i] < mid ? left.push(ary[i]) : right.push(ary[i])
        }
        return quickSort(left).concat(mid, quickSort(right))
      }
      var arr = [15, 3, 6, 12, 11, 38, 19]

      console.log(quickSort(arr))
    </script>
  </body>
  </html>
```

复习

// 数组方法

```
// splice push pop shift unshift
// join slice forEach map some filter every
// reverse concat sort

// 向数组末尾添加
// push(x)
// splice(arr.length, 0, x)
// arr[arr.length] = x

// 删除数组末尾最后一项
// pop()
// splice(arr.length-1)
// splice(-1)
// arr.length--

// 向数组开头添加
```

```
// unshift
// splice(0, 0, x)

// 删除数组开头
// shift()
// splice(0, 1)

// 数组克隆、截取
// slice()/slice(0)/slice(n)

// 数组拼接
// concat()

// 数组排序
// arr.sort(function(a, b) {
//   return a - b // 从小到大
// })
// arr.sort(function (a, b) {
//   return b - a // 从大到小
// })

// 数组循环遍历
// forEach map
// for循环语句

// 数组转换
// join(连接符)/toString()
```

字符串方法

// indexOf/lastIndexOf 查找某个字符的索引位置

```
// 字符串截取
// substr substring slice

// 大小写转换
// toLowerCase() / toUpperCase()

// 将字符串拆分成数组
// split(分隔符) '1+2+3'

// 替换某个字符
// replace()
```

Math方法

```
// Math方法
// Math.round() 四舍五入
// Math.random() 随机数 [0, 1)
// Math.ceil() 向上取整
// Math.floor() 向下取整
// Math.max() 最大值
// Math.min() 最小值

// n到m之间随机正数
// Math.round(Math.random() * (m - n) + n)

// 0到m之间随机正数
// Math.round(Math.random() * m)
```

函数任意数求和

```

```

```
function sum() {
    console.log(arguments) // 实参集合
    let total = 0
    for (let i = 0; i < arguments.length; i++) {
      var n = Number(arguments[i])
      if (!isNaN(n)) {
        total += n
      }
    }
    return total
  }

  // console.log(sum(1, 2, 3))
  console.log(sum(1, '12', 10, NaN, undefined))
```

```

```

隔行变色

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scal
e=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
  <style>
   .c0 {
     background-color: aqua
   }

   .c1 {
     background-color: antiquewhite
   }
  </style>
</head>
<body>
  <ul id="list">
    <li>item1</li>
    <li>item2</li>
    <li>item3</li>
    <li>item4</li>
    <li>item5</li>
    <li>item6</li>
    <li>item7</li>
    <li>item8</li>
    <li>item9</li>
    <li>item10</li>
  </ul>
  <script>
    let list = document.querySelectorAll('#list>li')
    // i % 2 0 1   'c' + i % 2 'c0' 'c1'
    for (let i = 0; i < list.length; i++) {
      list[i].className = 'c' + (i % 2) // 'c0' 'c1'

      list[i].onmouseover = function () {
        this.style.backgroundColor = 'red'
      }

      list[i].onmouseout = function () {
        this.style.backgroundColor = ''
      }
    }

    // for (let i = 0; i < list.length; i++) {
    //   list[i].style.backgroundColor = i % 2 ? 'red' : 'pink'
```

```
    //   list[i].onmouseover = function () {
    //     this.style.backgroundColor = 'green'
    //   }

    //   list[i].onmouseout = function () {
    //     this.style.backgroundColor = i % 2 ? 'red' : 'pink'
    //   }
    // }
  </script>
</body>
</html>
```

递归求和

1+.....+100求和

```
    // 从1到100
    let total = 0
    for (let i = 1; i <= 100; i++) {
      total += i
    }
    console.log(total)

    function add(n) {
      if (n <= 1) { // 递归结束条件
        return n
      }
      return n + add(--n) // 100 + add(99)
    }
    console.log(add(100))
```

1~ ~100能被2整除的数相加

```
<meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scal
e=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
  <script>
    // 从1到100 能够被2整除 的累加求和
```

```
let total = 0
for (let i = 1; i <= 100; i++) {
  if (i % 2 === 0) {
    total += i
  }
}
console.log(total)

function sum(n) {
  if (n <= 0) { // 结束条件
    return n
  }

  // 能够被2整除
  if (n % 2 === 0) {
   return n + sum(--n)
  }

  // 不能被2整除
  return sum(--n)
}
console.log(sum(100))

// sum(100) => 100 + sum(99)
// sum(99) => 100 + sum(98)
// sum(98) => 100 + 98 + sum(97)
// sum(97) => 100 + 98 + sum(96)

// 从1到100
// let total = 0
// for (let i = 1; i <= 100; i++) {
//   total += i
// }
// console.log(total)

// function add(n) {
//   if (n <= 1) { // 递归结束条件
//     return n
//   }
//   return n + add(--n) // 100 + add(99)
// }
// console.log(add(100))



// n-- => n = n - 1
// n - 1

// 100 + add(99)
```

```
    // => 100 + 99 + add(98)
    // => 100 + 99 + 98 + add(97)
    // => 100 + 99 + 98 + 97 + add(97)
    // add(1) => 1

    // i++ ++i
    // i-- --i
  </script>
</body>
</html>
```

Dom

```
<script>
    // 获取页面元素
    // document.getElementById('#id')
    // document.getElementsByClassName('.class')
    // document.getElementsByTagName('tag')
    // document.querySelector('css选择器')
    // document.querySelectorAll('css选择器')

    // DOM操作
    var h3 = document.createElement('h3')
    h3.innerHTML = 'hello h3!'
    document.body.appendChild(h3)
    // insertBefore(newNode, oldNode)
    // removeChild() 删除子节点
    // replaceChild(newNode, oldNode)

    // cloneNode() 只是把当前节点克隆 不克隆里面后代节点
    // cloneNode(true) 深度克隆
  </script>
```