0.1+0.2问题

转成整数处理

```
function accAdd(arg1,arg2){
  var r1,r2,m;
  try{
    r1=arg1.toString().split(".")[1].length
  }catch(e){
    r1=0
  }
  try{
    r2=arg2.toString().split(".")[1].length
  }catch(e){
    r2=0
  }
  m=Math.pow(10,Math.max(r1,r2));
  return (arg1*m+arg2*m)/m;
}
var result = accAdd(0.1,0.2)
console.log(result) // 0.3
```

大数相加解决[415.字符串相加]

传两个字符串进来，返回一个字符串

```
var addStrings = function (num1, num2) {
    let result = '';
    let i = num1.length - 1, j = num2.length - 1, carry = 0;
    while (i >= 0 || j >= 0) {
        let n1 = i >= 0 ? +num1[i] : 0;
        let n2 = j >= 0 ? +num2[j] : 0;
        const temp = n1 + n2 + carry;
        carry = temp / 10 | 0;
        result = `${temp % 10}${result}`;
        i--; j--;
    }
    if (carry === 1) result = `1${result}`;
    return result;
```

```
};
```

传两个字符串进来，返回一个字符串

- 转成数字相加的问题
- 注意处理全零字符串的情况

```
var multiply = function (num1, num2) {
    let result = '0';
    let i = num1.length - 1;
    while (i >= 0) {
        let subfixZero = new Array(num1.length - 1 - i).fill('0').join('');
        let sumCount = +num1[i];
        let tempSum = '0';
        while (sumCount > 0) {
            tempSum = bigSum(tempSum, num2);
            sumCount--;
        }
        tempSum = `${tempSum}${subfixZero}`;
        result = bigSum(result, tempSum);
        i--;
    }
    // 处理一下开头的零
    for (let i = 0; i < result.length; i++) {
        if (result[i] !== '0') {
            return result.slice(i);
        }
    }
    return '0';

    function bigSum(n1, n2) {
        let result = '';
        let i = n1.length - 1, j = n2.length - 1, curry = 0;
        while (i >= 0 || j >= 0) {
            let l1 = i >= 0 ? +n1[i] : 0;
            let l2 = j >= 0 ? + n2[j] : 0;
            let sum = l1 + l2 + curry;
            curry = sum / 10 | 0;
            result = `${sum % 10}${result}`;
            i--; j--;
        }
        if (curry === 1) result = `1${result}`;
        return result;
    }
};
```

数组乱序输出

Math.random输出的结果是0-1内的小数，可以直接通过length映射

```
const randomIndex = Math.round(Math.random()*(array.length - 1 -i) + 1);
```

数组去重复（7种方法）

关键点是NaN怎么判断，对NaN进行去重，这个题目的另一个考察点是对API的灵活运用，虽然很多方法不可能用在实际的场景中，但是who care，面试官只会觉得你懂得好多～

- 1.利用Set()+Array.from()
  - 方式对NaN和undefined类型去重也是有效的，是因为NaN和undefined都可以被存储在Set中， NaN之间被视为相同的值
- 2.利用两层循环+数组的splice方法
  - 此方法对NaN是无法进行去重的，因为进行比较时NaN !== NaN
- 3.利用数组的indexOf方法
  - 新建一个空数组，遍历需要去重的数组，将数组元素存入新数组中，存放前判断数组中是否已经含有当前元素，没有则存入。此方法也无法对NaN去重
  - indexOf() 方法：返回调用它的String对象中第一次出现的指定值的索引
- 4.利用数组的includes方法
  - 此方法逻辑与indexOf方法去重异曲同工，只是用includes方法来判断是否包含重复元素。
- 5.利用数组的filter()+indexOf()
  - 输出结果中不包含NaN，是因为indexOf()无法对NaN进行判断
- 6.利用Map()
  - 使用Map()也可对NaN去重，原因是Map进行判断时认为NaN是与NaN相等的
- 7.利用对象
  - 和Map()是差不多的，主要是利用了对象的属性名不可重复这一特性。

数组扁平化flatten(6种方法)

- 递归
- reduce
- 扩展运算符

- toString,split
- es6 flat
- 正则和json，json.stringify

```
function flatten(arr) {
  let result = [];
  for (let i = 0; i < arr.length; i++) {
    if (Array.isArray(arr[i])) {
      result = result.concat(flatten(arr[i]));
    } else {
      result.push(arr[i]);
    }
  }
  return result;
}

function flatten(arr) {
  return arr.reduce((p, c) => {
    return p.concat(Array.isArray(c) ? flatten(c) : c);
  }, [])
}
```

🔥对象扁平化flatObj

多次遇到，建议背诵

```
/* 题目*/
var entryObj = {
    a: {
        b: {
            c: {
                    dd: 'abcdd'
            }
        },
        d: {
            xx: 'adxx'
        },
        e: 'ae'
    }
}

// 要求转换成如下对象
var outputObj = {
```

```
        'a.b.c.dd': 'abcdd',
        'a.d.xx': 'adxx',
        'a.e': 'ae'
}

function flat(obj, path = '', res = {}, isArray) {
  for (let [k, v] of Object.entries(obj)) {
    if (Array.isArray(v)) {
      let _k = isArray ? `${path}[${k}]` : `${path}${k}`;
      flat(v, _k, res, true);
    } else if (typeof v === 'object') {
      let _k = isArray ? `${path}[${k}].` : `${path}${k}.`;
      flat(v, _k, res, false);
    } else {
      let _k = isArray ? `${path}[${k}]` : `${path}${k}`;
      res[_k] = v;
    }
  }
  return res;
}

console.log(flat({ a: { aa: [{ aa1: 1 }] } }))
```

数字千分位分割

注意可能有小数

```
function format(number) {
  number = number.toString();
  let decimals = '';
  number.includes('.') ? decimals = number.split('.')[1] : decimals;

  let len = number.length;
  if (len < 3) {
    return number;
  } else {
    let temp = '';
    let remainder = len % 3;
    decimals ? temp = '.' + decimals : temp;
    if (remainder > 0) {
      return number.slice(0, remainder) + ',' + number.slice(remainder,
len).match(/\d{3}/g).join(',') + temp;
    } else {
      return number.slice(0, len).match(/d{3}/g).join(',') + temp;
    }
```

```
  }
}
```

js下划线转驼峰处理「快手」

正则法

```
function camelCase(str) {
  return str.replace(/_([a-z])/g, function(match, group1) {
    return group1.toUpperCase();
  });
}

console.log(camelCase("some_string"));  // "someString"
```

补充

```
function camelCase(str) {
  return str.replace(/([-_])([a-z])/g, function(match, group1, group2) {
    return group2.toUpperCase();
  });
}

console.log(camelCase("some-string_with-underscores"));
```

Hex转RGB的方法

```
function hexToRgb(val) {
  //HEX十六进制颜色值转换为RGB(A)颜色值
  // 16进制颜色值的正则
  var reg = /^#([0-9a-fA-f]{3}|[0-9a-fA-f]{6})$/;
  // 把颜色值变成小写
  var color = val.toLowerCase();
  var result = '';
  if (reg.test(color)) {
    // 如果只有三位的值，需变成六位，如：#fff => #ffffff
    if (color.length === 4) {
      var colorNew = '#';
      for (var i = 1; i < 4; i += 1) {
        colorNew += color.slice(i, i + 1).concat(color.slice(i, i + 1));
      }
```

```
      color = colorNew;
    }
    // 处理六位的颜色值，转为RGB
    var colorChange = [];
    for (var i = 1; i < 7; i += 2) {
      colorChange.push(parseInt('0x' + color.slice(i, i + 2)));
    }
    result = 'rgb(' + colorChange.join(',') + ')';
    return { rgb: result, r: colorChange[0], g: colorChange[1], b: colorChange[2] };
  } else {
    result = '无效';
    return { rgb: result };
  }
}
```

实现模版字符串解析

```
var template = `
<div>
    <% if(name){ %>
        <span>%= name =%</span>
    <% } %>
    %= age =%
<div>`
let str = rander(template, {name: '小明', age: 18})
// 解析完成 str <div> <span>小明</span>18<div>
```

```
function parseTemplateString (templateString, data) {
  // 使用正则表达式在模板字符串中查找所有 ${...} 的实例
  const regex = /${(.*?)}/g;
  // 使用 replace() 方法将每个 ${...} 的实例替换为数据对象中相应的值
  const parsedString = templateString.replace(regex, (match, key) => {
    // 使用 eval() 函数来评估 ${...} 中的表达式，并从数据对象中返回相应的值
    return eval(`data.${key}`);
  });
  return parsedString;
}
```

🔥数组转树形结构的三种方法

递归解法非常好理解，代码量也很少，题目出现概率很高

```
{
  "city": [
    { "id": 12, "parent_id": 1, "name": "朝阳区" },
    { "id": 241, "parent_id": 24, "name": "田林街道" },
    { "id": 31, "parent_id": 3, "name": "广州市" },
    { "id": 13, "parent_id": 1, "name": "昌平区" },
    { "id": 2421, "parent_id": 242, "name": "上海科技绿洲" },
    { "id": 21, "parent_id": 2, "name": "静安区" },
    { "id": 242, "parent_id": 24, "name": "漕河泾街道" },
    { "id": 22, "parent_id": 2, "name": "黄浦区" },
    { "id": 11, "parent_id": 1, "name": "顺义区" },
    { "id": 2, "parent_id": 0, "name": "上海市" },
    { "id": 24, "parent_id": 2, "name": "徐汇区" },
    { "id": 1, "parent_id": 0, "name": "北京市" },
    { "id": 2422, "parent_id": 242, "name": "漕河泾开发区" },
    { "id": 32, "parent_id": 3, "name": "深圳市" },
    { "id": 33, "parent_id": 3, "name": "东莞市" },
    { "id": 3, "parent_id": 0, "name": "广东省" }
  ]
}
```

```
function arrayToTreeV3(list, root) {
  return list
    .filter(item => item.parent_id === root)
    .map(item => ({...item, children: arrayToTreeV3(list, item.id)}))
}
```

获取URL中的参数

这里主要还是正则表达式的设计

- /?&/igm，前面是？或者&，任意字符直到遇到=，使用非贪婪模式，等号后面是非&符号的任意字符，然后去匹配就好了
- 理论上可以用matchAll，然后用迭代器去处理

```
function name(url) {
    const _url = url || window.location.href;
    const _urlParams = _url.match(/[?&](.+?=[^&]+)/igm);
    return _urlParams ? _urlParams.reduce((a,b) => {
        const value = b.slice(1).split('=');
```

```
        a[value[0]] = value[1];
        return a;
    }, {}) : {}

}
```

小结

场景题目其实很多，没办法去枚举，但是这里标记出来的是相对高频的题目