2021 校招前端笔试题

要求

1. 题型：单选（10）+ 编程（4）
2. 编程题难度适中，代码量少

考察点分布（建议）

1. 单选（10）

- 智力、数学题（1）
- 计算机网络 （2）
- 数据结构、算法 （2）
- 浏览器 （2）
- 数据库 （1）
- Javascript 语言基础 （2）

2. 编程（4）

- 语言基础 （1）
- 数据结构、算法（2）
- 复杂问题（1）

候选题库

~~欢迎补充~~

1. 单选

题目：javascript 执行 0.3 + 0.6 等于多少？（jin-max：考计算机基本素养）
A. 0.9 B. 1 C.0.8999999999999999 D.0.8999999

（答案： C）

（解答：ieee 754 标准的浮点数，javascript 数字基本是 64位的双精度浮点数，Mantissa（fraction part）52位 用于表示小数，1 ^ (-52) 精确到小数点第16位，你用 Number.Epsilon 可以查到 JS 可表示的最小数字，对于0.1 到 0.9 的小数，仅仅 0.5 的 Mantissa 全都为 0 （ 1 / 2），基本上 小数 都为逼近数字，只是可能 0.3 和 0.7 的浮点数表示小数第16位不是 0，所以显示出来差异（参考这个链接），不像0.1 和 0.7，答案 C 与 D 只是 考 js 的数字是single 还是 double precision 可以推导出来。。基本 js 会从小数第16位后截断，所以 0.1， 0.2 等小数在 js 和其他一些语言下还是 0.1（1.0000000000000000555111512312 6E-1）和 0.2 手动 Doge ）

题目：下面哪个不是 javascript 的原生错误类型？（jin-max： 考 语言知识 和 英语）

A. RangeError B. TypeError C. SyntaxError D. RuntimeError

（答案：D）

题目：下面哪个不是优化递归 爆栈(StackOverflow) 的方法 ？（jin-max： 考计算机基本素养）

A. 改为循环B. 柯里化（Currying）C. 尾递归（Tail Recursion）D.蹦床函数（Trampoline）

（答案：B ，注：递归函数利用栈(stack)，蹦床函数相互调用利用堆(heap)，操作系统中 sizeof(heap) >> sizeof(stack) ）

题目：一棵有 2049 个节点的二叉树（Binary Tree）最少有几层 ？（jin-max： 考计算机基本素养）

A. 10B. 11C. 12D. 13

(答案：CMath.ceil(Math.log2(#nodes) + 1)) 和 简单的数学归纳法 推导)

题目: 下面哪个计算结果不为 2 ？（jin-max： 考计算机基本素养）

A. 1 ^ 1 >> 1 << 1 B. 1 ** 2 C. 1 ^ 3 D. 3 & 6

(答案：B)

题目：下面 Javascript 程序的执行结果哪一项 ？ （ jin-max： 考 js 的 workloop）

```
setTimeout(function() {
    console.log('1');
})
new Promise(function(resolve) {
    console.log('2');
    for(var i = 0; i < 1000; i++) {
         i == 99 && resolve();
    }
    console.log('3');
}).then(function() {
    console.log('4');
})
console.log('5');
```

A. 1, 2, 3, 4, 5 B. 1, 2, 5, 4, 3 C. 1, 2, 3, 5, 4 D. 2, 3, 5, 4, 1

（答案：D ）

题目：以下代码的运行结果是 (C) （chennan）

```
var x = '16'
var y = 3
[x + y, x - y, x / y, x % y]
```

A. [19, 13, 5, 1]
B. ['163', NaN, NaN, NaN]
C. ['163', 13, 5.333333333333333, 1]
D. [19, 13, 5.333333333333333, 1]

题目：以下代码的运行结果是 (B) （chennan）

```
for(var i = 0; i < 10; i++) {
 setTimeout(() => console.log(i), 1000);
}
```

A. 0-9 B. 10个10 C. 10个9 D. 无限循环

题目：给定一个正则表达式 /^[aeiou]*d$/, 下面哪项可以匹配成功（A）(chennan)

A. d B. aaa2 C. aeiou*d D. abcd

题目：快速排序在下列哪种情况下最易发挥其长处（C）（chennan）
A. 被排序的数据中含有多个相同排序码
B. 被排序的数据已基本有序
C. 被排序的数据完全无序
D. 被排序的数据中的最大值和最小值相差悬殊

题目：关于 Web 语义化的说法错误的是（D）(chennan)
A. 语义化的含义就是用正确的标签做正确的事情
B. 语义化让页面的内容结构化，结构更清晰
C. 语义化便于对浏览器、便于代码阅读、便于维护
D. 语义化便于SEO, 页面中多放置一些H1 标签，会提高网页的排名

题目：关于浏览器的同源策略，下列说法错误的是（B）（chennan）
A. 同源策略的目的是为了保证用户信息的安全，防止恶意网站窃取数据
B. 可以使用脚本将 document.domain 的值设置为其当前域或其子域，用于后续的源检查
C. 跨域资源共享（CORS）是 HTTP 协议的一部分，它允许服务端来指定哪些主机可以从这个服务端加载资源
D. 可以通过 window.postMessage 接口读写其他窗口的 LocalStorage

智力题（chennan）：

警方在一起案件的侦破过程中，抓获了甲乙丙三个犯罪嫌疑人。
甲说：“丙在说谎。”
乙说：“甲和丙都在说谎。”
丙说：“乙在说谎。”
由此可以推知，三个人中说真话的是?

A. 甲 B. 乙 C. 丙 D. 没有人

题目：后序遍历（Post-Order Traversal）结果为 2,1,1,5,4 的二叉树可能有几种？（lizhifeng）
A.28
B. 35
C. 42
D. 49
答案：C
思路：核心是转换一下就等价于节点数为 n 的二叉树共有几种的问题，然后简单地从 1 推到 5 或者直接套用 Catalan就 over 了，还可以靠猜测问题规模猜到大致范围；
两个相同的 1 是干扰项，长度为 5 也很容易诱惑人放弃思考去穷举

题目：浏览器发送常见的 HTTP 请求时，这些请求与其相应的 TCP 连接之间的关系是（lizhifeng）
A. 每个 HTTP 请求可以由多个 TCP 连接进行发送
B. 每个 TCP 连接可以发送多个 HTTP 请求
C. 每个 HTTP 请求跟每个 TCP 连接一一对应
D. 以上说法都不对
答案：B（稍微考点跟前端相关又简单点的东西…）

题目：以下函数的时间复杂度是（lizhifeng）

以下代码的时间复杂度是

```
function f(n) { // n
  if (n <= 1) {
    return n
  } else if (n % 2 === 0) { //  n
    return 2 * f(n / 2)
  } else {
    return f(n - 1) + 1
  }
}
```

A. O(log(n))
B. O(n)
C. O(n*log(n))
D. 以上说法都不对
答案：A
思路：因为是简单的尾递归，所以直接计算收敛速度就行；第二个分支收敛最快，等价于 log2(n)；第三个分支在递归的时候下一步几乎总是落在第二个分支，所以次数不会超过第二个分支的数量；综上，复杂度约为 2log2(n)；
题目的主要干扰项在于 f(n) 的返回值实际上就是 n，很容易让人认为是 >= O(n)，也有可能认为是 D

网络相关单选题（yanghanxing）：

题目：在 OSI 7 层模型中，以下哪个网络协议工作在最下层（底部）？

A. FTP B. TCP C. IP D. DNS
（答案：C ）

题目：关于 HTTP，以下说法正确的是？
A. 通过 HTTP 请求下载 1 GB 大的文件，产生的网络流量有可能小于 1 GB
B. HTTP 请求的 Content-Length 值包含了 Headers 与 Body 的大小
C. 对于 HTTPS 请求，中间人无法得到你要请求的目标服务器的 IP 信息，因此可以避免中间人攻击
D. 多条 HTTP 请求间无法共用 TCP 连接，这是发送 HTTP 请求开销比较大的主要原因
（答案：A）

2. 编程

// 基本的字符串处理 出题人：陈楠

题目：转换日期格式。输入一个日期字符串 date，格式为 Day Month Year，其中：
Day 是集合 {"1st", "2nd", "3rd", "4th", ..., "30th", "31st"} 中的一个元素。
Month 是集合 {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"} 中的一个元素。
Year 的范围在 [1900, 2100]之间。
请将字符串转变为 YYYY-MM-DD 格式。

注：输入日期保证是合法的，不需要处理异常输入。

示例 1:
输入： date = "1st Oct 2019"
输出：“2019-10-01”
示例 2:
输入： date = "27th Mar 1966"
输出：“1966-03-27”

题目：实现一个 flat()
函数，该方法会按照一个可指定的深度递归遍历数组，并将所有元素与遍历到的子数组中的元素合并为一个新数组返回。(max-jin-max)

输入/输出例子：

```
const arr1 = [0, 1, 2, [3, 4]];

console.log(arr1.flat());
// expected output: [0, 1, 2, 3, 4]

const arr2 = [0, 1, 2, [[[3, 4]]]];

console.log(arr2.flat(2));
// expected output: [0, 1, 2, [3, 4]]
```

```
 1
//  reduceconcat
var arr1 = [1,2,3,[1,2,3,4, [2,3,4]]];

function flatDeep(arr, d = 1) {
   return d > 0 ? arr.reduce((acc, val) => acc.concat(Array.isArray(val) ?
flatDeep(val, d - 1) : val), [])
                : arr.slice();
};

flatDeep(arr1, Infinity);
// [1, 2, 3, 1, 2, 3, 4, 2, 3, 4]


 2
// forEach
```

```
const eachFlat = (arr = [], depth = 1) => {
  const result = []; //
  //
  (function flat(arr, depth) {
    // forEach
    arr.forEach((item) => {
      //
      if (Array.isArray(item) && depth > 0) {
        //
        flat(item, depth - 1)
      } else {
        //
        result.push(item)
      }
    })
  })(arr, depth)
  //
  return result;
}

3
// for of
const forFlat = (arr = [], depth = 1) => {
  const result = [];
  (function flat(arr, depth) {
    for (let item of arr) {
      if (Array.isArray(item) && depth > 0) {
        flat(item, depth - 1)
      } else {
        // undefined
        item !== void 0 && result.push(item);
      }
    }
  })(arr, depth)
  return result;
}


4:
//
//
//  shift / unshift  w/o  OPs
var arr1 = [1,2,3,[1,2,3,4, [2,3,4]]];
function flatten(input) {
  const stack = [...input];
  const res = [];
  while (stack.length) {
    //  pop  stack
    const next = stack.pop();
    if (Array.isArray(next)) {
      //  push
      stack.push(...next);
    } else {
```

```
      res.push(next);
    }
  }
  //
  return res.reverse();
}
flatten(arr1);// [1, 2, 3, 1, 2, 3, 4, 2, 3, 4]
```

```
5:
function* flatten(array) {
    for (const item of array) {
        if (Array.isArray(item)) {
            yield* flatten(item);
        } else {
            yield item;
        }
    }
}

var arr = [1, 2, [3, 4, [5, 6]]];
const flattened = [...flatten(arr)];
// [1, 2, 3, 4, 5, 6]
```

flat PR
https://github.com/qbox/kodo-web/pull/1100/files#diff-e3848f9eb092d1d09302
57886c1c7872f093164662a603bed21db30c5258e9edR58

题目：已知数组 list[n][2]，这 n 组数据经过若干次排序、删减后得到 newList[m][2]，满足 newList[i][1] 等于 newList[i+1][0]，求 m 可能的最大值（lizhifeng）
要求：不限语言；只完成核心思路也能得分，不强求能运行或无
bug；允许假定问题规模不大，从而无须过多考虑时间或空间复杂度，但是优化了能加分
例子 1：
输入：[[3,8],[2,9],[4,2],[9,7],[2,5]]
输出：3（对应 [[4,2],[2,9],[9,7]]）
例子 2：
输入：[[2,8],[6,6],[8,6],[4,7],[8,6],[7,1],[8,1],[2,6],[4,8],[6,2],[4,1]]
输出：6（对应 [[4,8],[8,6],[6,6],[6,2],[2,8],[8,1]）

// ==========
考点：基本数据结构 & 递归，本质是 不带权有向图 + 邻接表存储的情况下求最长路径
耗时：10 ~ 30 min

参考答案

```
let m = 0
function dfs(arr, curr = null, len = 0) {
  if (arr.length === 0) {
    m = Math.max(m, len)
    return
  }
  for (let i = 0; i < arr.length; i++) {
    !curr || curr[1] === arr[i][0]
      ? dfs([...arr.slice(0, i), ...arr.slice(i + 1)], arr[i], len + 1) //
      : dfs([], null, len) //
  }
}
console.log(m)
```

题目：已知数组 list 和一个栈，list 元素须按顺序入栈，但是栈内元素可以任意出栈，求共有多少种不同的出栈结果（lizhifeng）
举个例子，如果用「待入栈队列，栈，已出栈队列」的三元组格式来表示每一步操作后的状态，并且入栈顺序是
1,3,5，那么一次可能的出栈过程为：
初始状态（135,,）-> 入栈（35,1,）-> 出栈（35,,1）-> 入栈（5,3,1）-> 入栈（,35,1）-> 出栈（,3,15）-> 出栈（,,153）
要求：不限语言；只完成核心思路也能得分，不强求能运行或无 bug；问题规模不大，可以不考虑时间或空间复杂度

例子 1：
输入：[1,3,5]
输出：5（对应 [1,3,5] [1,5,3][3,1,5][3,5,1][5,3,1]）
例子 2：
输入：[1,1,3]
输出：3（对应[1,1,3][1,3,1][3,1,1]）

// ==========
考点：简单模拟题，栈的基本概念 & 简单递归；注意，上面的第二个 case 是为了避免直接套 catalan，直接套公式就没意思了
耗时：10 ~ 30 min

参考答案

```
const result = new Set()
function f(input, stack = [], output = []) {
  if (input.length === 0 && stack.length === 0) {
    result.add(output.join())
  } else {
    if (input.length > 0) f(input.slice(0,-1), [...stack,input.slice(-1)],
output) //
    if (stack.length > 0) f(input, stack.slice(0,-1),
[...output,stack.slice(-1)]) //
  }
}
f([...list].reverse()) //
console.log(result.size)
```

题目：编写一个函数 deepClone(obj): obj，实现对象的深拷贝（可以使用任意熟悉的编程语言实现）。

评分标准：

- 一般：能看出来对深拷贝的原理是理解的，但是编写的程序不能正常工作
- 良好：即能看出对深拷贝对原理的理解，程序又可以正常工作；
- 优秀：即能看出对深拷贝对原理的理解，程序又可以正常工作，并且还进行了优化，比如：
  - 正则表达式对象可以直接使用 `toString()`
    进行转换为正则字符串，然后再使用正则表达式对象的构造方法传入得到的正则字符串来获得新的正则对象，而无需对正则表达式对象的每个属性值进行递归遍历
  - 如果考虑到了对象的属性中包含了 Symbol 类型的使用普通方式变量不能遍历到的属性，说明考生对 JS
    有过深入理解，也可以作为一个加分项。
  - Lodash 对 Map、Set、RegExp 等常见的对象的深拷贝都有做处理，具体实现可以参考 https://sourcegraph.com/github.com/lodash/lodash/-/blob/.internal/baseClone.js#L207

出题人：何文杰

参考资料

- 校招试卷
- 前端开发笔试出题参考

- 2020年的笔试题


2020春招前端开发...习生笔试题目.pdf