2022前端年底面试总结

又到年底了，很多小伙伴又开始 跳槽 了，本次汇总都是 面试真题 ，来自各位小伙伴有 大厂 也有 小厂 ，还有 外包 可以说很全面了。

某外包公司 3-5年 前端开发

1.计算机CPU功能和基本结构

属实开幕雷击 这种底层知识对于前端来说可能问的比较少

CPU 它是你的电脑中最 硬核 的组件，这种说法一点不为过。 CPU 是能够让你的计算机叫 计算机 的核心组件，但是它却不能代表你的电脑， CPU 与计算机的关系就相当于大脑和人的关系

cpu的功能

主要是从程序或应用程序获取指令并执行计算。此过程可以分为三个关键阶段: 提取 ， 解码 和 执行

cpu的内部结构

以C语言为例 程序编译执行的过程 在这个流程中，CPU 负责的就是解释和运行最终转换成机器语言的内容

2.计算机的虚拟内存介绍一下

虚拟内存

虚拟内存 是内存和磁盘交互的第二个媒介。虚拟内存是指把磁盘的一部分作为 假想内存 来使用。这与磁盘缓存是假想的磁盘 (实际上是内存)相对，虚拟内存是假想的内存 (实际上是磁盘) 虚拟内存是计算机系统内存管理的一种技术。它使得应用程序认为它拥有 连续可用 的内存(一个完整的地址空间)，但是实际上，它通常被分割成多个物理碎片，还有部分存储在外部磁盘管理器上，必要时进行数据交换。

3.原生js对象的特征

我认为这个题他是想问对象的 `封装`， `继承`， `多态`，三大特征

`封装` 就是把抽象出来的数据和对数据的操作封装在一起，数据被保护在内部，程序的其它部分只有通过被授权的操作(成员方法)，才能对数据进行操作。

`继承`，继承的方式有很多种，常用的 `原型链继承`， `寄生继承`， `call apply借用法`.

`多态`，函数重载，很多语言都支持函数重载，js是不支持函数重载的,如果js想实现函数重载只能通过判断参数的个数或者类型来实现，或者使用ts去支持。

4.vue组件的传值方法都有哪些

传参的方式就很多了 `只包括组件不涉及路由传参`

1 父子 父给子 使用props 自给父使用emit

2 全局状态管理工具 Vuex pinia 等

3 发布订阅模式 eventBus mitt 等

4 借助 `$children` / `$parent`

5 provide inject

6 ref / refs

7 `$attrs`

5.vue3新特性

说起3的新特性也是挺多的

最主要的区别就是 `Composition API`

新增两个内置组件 `Teleport` `Suspense`

`Vue.prototype` 替换为 `config.globalProperties`

去掉sync修饰符 并且v-model名称改变 `modelValue` 作为 prop， `update:modelValue`

style中使用变量 `color: v-bind('theme.color')`

支持 `Fragments`

响应式原理的变化 `defineProperty` 换成 `proxy`

生命周期钩子

```js
Vue2--------------vue3
beforeCreate  -> setup()
created       -> setup()
beforeMount   -> onBeforeMount
mounted       -> onMounted
beforeUpdate  -> onBeforeUpdate
updated       -> onUpdated
beforeDestroy -> onBeforeUnmount
destroyed     -> onUnmounted
activated     -> onActivated
deactivated   -> onDeactivated
```


6.哪个大学毕业的，学的是什么专业，学历学信网可不可查

答...

京东 3-5 年 前端

1.深拷贝浅拷贝的方法都有哪些

这道题主要考对引用类型的理解

1. `深拷贝` 偷懒方法 这种方法 可以深拷贝 注意 `function和undefined` 是会丢掉的 没有这两种类型可以用

```js
const obj2 = JSON.parse(JSON.stringify(obj1))
```


2. `深拷贝` 递归解法

```js 复制代码
function DeepClone(data) {
    const newData = Array.isArray(data) ? [] : {}
    for (let key in data) {
      if (data[key] && typeof data[key] === 'object') {
        newData[key] = DeepClone(data[key])
      } else {
        newData[key] = data[key]
      }
    }
    return newData
  }
```

3. 浅拷贝 Object.assign 这个最常用了 注意这个方法是浅拷贝

```js 复制代码
  Object.assign({},{},...)
```

4. 浅拷贝 es6扩展运算符

```js 复制代码
const obj2 = {...obj1}
```

这道题相信大家还是都会的

2.async和await

`async` 是一个加在函数前的修饰符，被async定义的函数会默认返回一个Promise对象resolve的值。因此对async函数可以直接then，返回值就是then方法传入的函数。

`await` `await` 也是一个修饰符，只能放在async定义的函数内。可以理解为等待。 `awai` t 修饰的如果是 `Promise` 对象：可以获取Promise中返回的内容（resolve或reject的参数），且取到值后语句才会往下执行； 如果不是 `Promise` 对象：把这个非 `promise` 的东西当做 `await` 表达式的结果。

```js 复制代码
async function some() {
   await Promise.xxxxxx
}
```

3.JS事件循环event loop 怎么理解

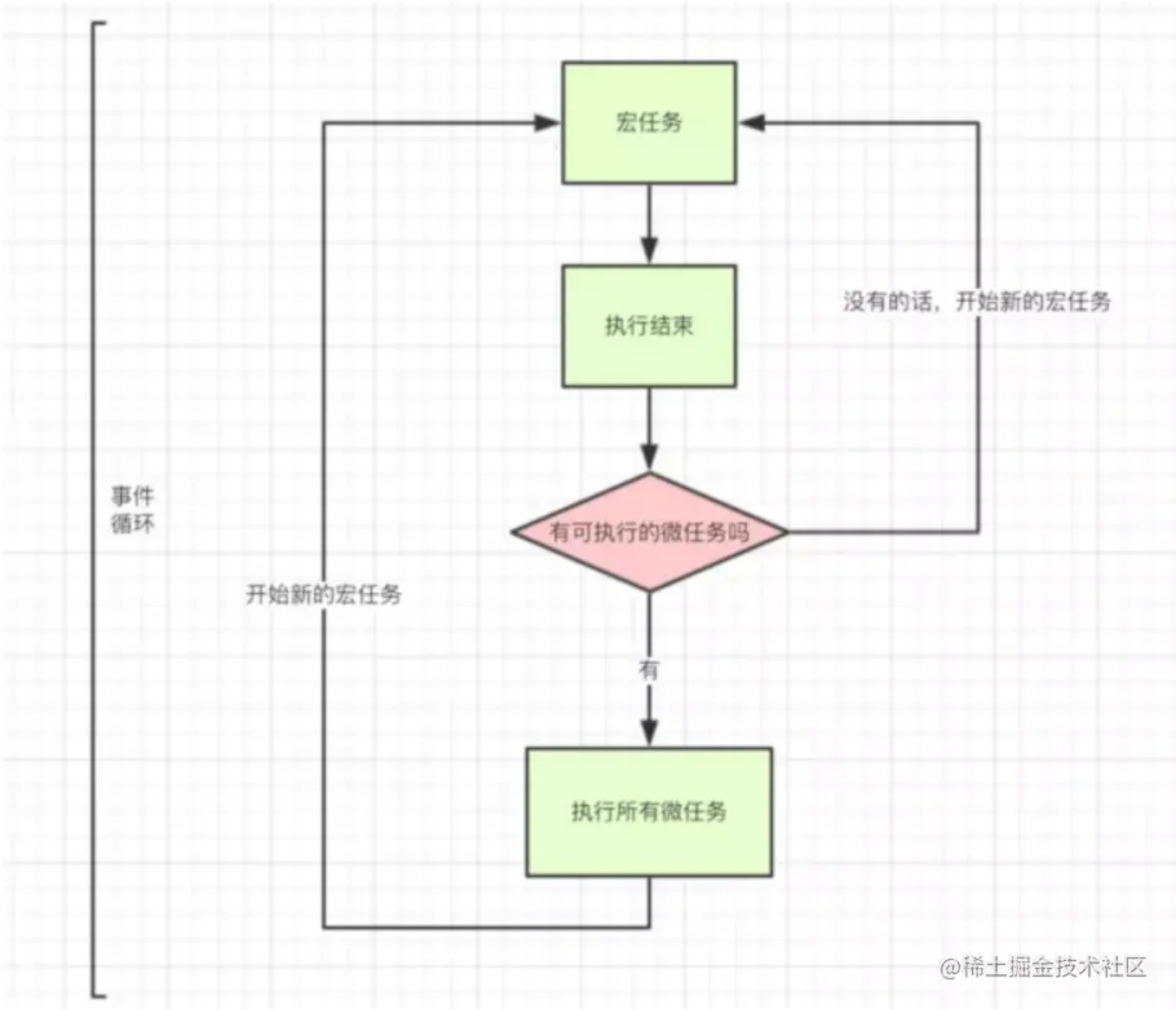
果然微任务 宏任务又来了

js宏任务有：`<script>整体代码`、`setTimeout`、`setInterval`、`setImmediate`、`Ajax`、`DOM事件`

js微任务有：`nodejs process.nextTick`、`MutationObserver`、`Promise.then catch finally`

执行顺序是宏任务先走应为script标签要先走，然后清空所有的同步任务，碰到异步任务都放到异步队列里面，

在异步队列里面先执行宏任务，然后执行本次宏任务里面的所有微任务，然后继续下一个tick一直循环直到全部执行完成



4.vue-router的方法都有哪些?

`redirect` 可以重定向

`push` 跳转页面并且有历史记录

`replace` 跳转且没有历史记录

`go` 前进后退

`back` 返回

`beforeEach` 前置守卫

`afterEach` 后置守卫

`addRoute` 动态路由

`removeRoute` 删除路由

`getRoutes` 查看路由表

5.HTTP状态码206是干什么的

这个应该问的是断点续传技术

首先就是前端拿到file 然后通过 `slice` 可以切片 然后使用 `promise.all` 并发上传 中间可以暂停也可以恢复上传 后端可以读到碎片最后上传完成之后后端会合成一个完整的文件 那如何记录当前上传到哪块了 此时需要通过请求头的Range 属性 提供该信息

```
复制代码
GET /file.zip HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-
excel, application/msword, application/vnd.ms-powerpointRange:
bytes=200000- //告诉服务器 file.zip 这个文件从200000字节开始传，前面的字节不用传了
Accept-Language: zh-cn
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)Connection: Keep-Alive
```

6.算法题环节

相交链表 跟LeetCode 基本一样 这边直接拿过来了

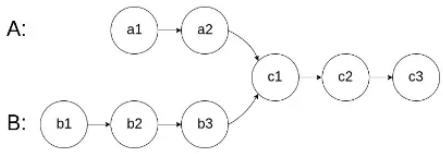
160. 相交链表

简单 1.9K

相关企业

给你两个单链表的头节点 `headA` 和 `headB` ，请你找出并返回两个单链表相交的起始节点。如果两个链表不存在相交节点，返回 `null` 。

图示两个链表在节点 `c1` 开始相交：



题目数据 **保证** 整个链式结构中不存在环。

注意，函数返回结果后，链表必须 **保持其原始结构** 。

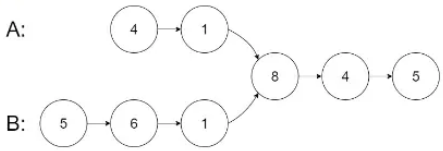
自定义评测：

评测系统 的输入如下（你设计的程序 **不适用** 此输入）：

- `intersectVal` - 相交的起始节点的值。如果不存在相交节点，这一值为 `0`
- `listA` - 第一个链表
- `listB` - 第二个链表
- `skipA` - 在 `listA` 中（从头节点开始）跳到交叉节点的节点数
- `skipB` - 在 `listB` 中（从头节点开始）跳到交叉节点的节点数

评测系统将根据这些输入创建链式数据结构，并将两个头节点 `headA` 和 `headB` 传递给你的程序。如果程序能够正确返回相交节点，那么你的解决方案将被 **视作正确答案** 。

示例 1：




```
输入：intersectVal = 8, listA = [4,1,8,4,5], listB = [5,6,1,8,4,5], skipA = 2, skipB = 3
输出：Intersected at '8'
解释：相交节点的值为 8 （注意，如果两个链表相交则不能为 0）。
从各自的表头开始算起，链表 A 为 [4,1,8,4,5]，链表 B 为 [5,6,1,8,4,5]。
在 A 中，相交节点前有 2 个节点；在 B 中，相交节点前有 3 个节点。
— 请注意相交节点的值不为 1，因为在链表 A 和链表 B 之中值为 1 的节点（A 中第二个节点和 B 中第三个节点）是不同的节点。换句话说，它们在内存中指向两个不同的位置，而链表
A 和链表 B 中值为 8 的节点（A 中第三个节点，B 中第四个节点）在内存中指向相同的位置。
```


我是这么写的

ts 复制代码

```ts
/**
 * Definition for singly-linked list.
 * class ListNode {
 *     val: number
 *     next: ListNode | null
 *     constructor(val?: number, next?: ListNode | null) {
 *         this.val = (val===undefined ? 0 : val)
 *         this.next = (next===undefined ? null : next)
 *     }
 * }
 */
let set = new Set()
function getIntersectionNode(headA: ListNode | null, headB: ListNode | null): ListNode | null {
        while (headA!=null) {
          set.add(headA)
          headA = headA.next
```

```
        }
        while (headB!=null) {
            if(set.has(headB)){
               return headB
            }
           headB = headB.next
        }

};
```

思路就是 在第一个循环里面疯狂添加 A链表的值 在第二个循环里面去检测 如果set里面的值 与B链表的值一样 那说明有相交