【前端面经】2023面试复盘之快手

结论

✅通过

市场寒冬呀~ 看了一圈，发现快手跟其他几个大厂类似，高阶的岗位非常非常少，所以就在官网上找了一个瞅着差不多的岗位让朋友内推了。

整体面完没想当快手是我觉得这几家里面试体验最舒服的，无论是面试官的水平还是跟其沟通的感觉都挺让人有好感的。如果不是当前互联网环境太差，快手又被抖音压制的厉害，我觉得快手真的算是一个不错的选择（当然，这单纯是我面试的感觉，不一定准😆）。不过快手的HR我还是要批评一下，前面面试环节老是催我，说业务有压力要快速推进，以至于我还约了一个晚上九点的面试。结果都面完收集薪资流水之后就慢吞吞，迟迟给不出结果😒。

一面

总时长：50min

一面聊整体的难度和节奏都还OK，能感觉到面试官在技术和管理上是都比较有经验，后来问的时候了解到是该部门下技术团队的负责人。不过一面没有写代码还是有些出乎意料的，估计是看要超时了或者有其他事情要忙。

聊项目和聊经历

> 自我介绍之后就是讲了一下我的一些项目情况，以及我的一些工作经历，包括个人绩效、架构能力、管理能力等等。与其他家聊的大差不差，主要就是对我有一个大概的了解，同时可能会考察一下沟通表达能力吧。

富文本编辑器的复制粘贴功能是如何实现的

> 这个问题的背景是用户经常从其他地方复制粘贴内容到富文本编辑器中，粘贴过来的内容其实是一个完整的 `HTML` ，那如何保证粘贴过来的内容能够在我们的富文本编辑器中友好的展示？
>
> 我们的解决方案为用户粘贴到编辑器中时，会弹窗提示，让用户自行选择粘贴内容的处理方式，主要有以下三种：
>
> 1. 不处理 —— 这种情况下我们不对用户粘贴的内容做任何处理，原封不动的将粘贴过来的 `HTML` 展示在编辑器中；即使后续在展示时出现样式异常，我们也不进行额外处理；
> 2. 处理成纯文本 —— 纯文本的情况是将 `HTML` 通过 `new DOMParser()` 解析变成DOM之后，再通过 `innerText` 属性获取到里面的纯文本信息；这种方式带来的问题是所有的文本都会变成一行，没有任何换行与空格；

3. 处理成纯文本但保留换行和空格 —— 这种方式也是先将 `HTML` 转换成DOM，之后通过 `textContent` 属性获取到带换行符和空格的文本内容。

有更好的实现方式吗

> 我能想到的更好的方式就是对粘贴过来的 `HTML` 进行逐行解析，拿到每一段文本的标签之后，跟我们自身富文本编辑器的标签做个映射，将其转换成富文本编辑器支持的标签。这样就可以实现文本粘贴过来样式变化不大，同时也能够比较好的在我们系统中展示。
>
> 但是这种方案有一定的复杂度，也需要一定的人力成本，对于我们来说投入产出比不高，因此没有采用这种方式实现。

小程序性能优化做了哪些事情

> 先讲了一下小程序的架构和渲染原理，阐述小程序性能的影响因素，之后则介绍对应的性能优化手段有哪些。
>
> 主要包含以下：
>
> 1. 使用小程序原生语法而不是类React或者Vue框架
> 2. 减少 `setData` 次数，同时优化 `setData` 的数据量大小
> 3. 请求预加载，重写路由方法，将下一个页面的请求提前到路由方法里调用
> 4. 减少 `wxml` 的嵌套深度和节点数量，同时对 `wxss` 相同样式做合并处理
> 5. 一些常规的优化手段：骨架屏、首屏数据缓存、分包、子包预加载、首屏接口合并、懒加载等方式

如何开发一个eslint的插件

> 插件的主入口暴露出一个对象，key为当前插件的规则名称，value值为该插件的具体内容。value也是一个对象，包含了 `meta` 字段用于声明当前插件的基础信息，如名称、版本号、文档等。另一个核心字段是 `create` ，该字段类型是一个方法，用于遍历代码的 `AST` 并做代码转换。

babel的转换流程是什么样的

> 首先读取字符串，然后通过 `babel-parser` 将字符串代码转换成 `抽象语法树AST` ，之后对该 `AST` 进行节点遍历和转换，生成新的 `AST` ，最后通过 `babel-generator` 将新的 `AST` 再转换成新的代码字符串。

babel包含哪几个部分，核心包有哪些

> 包含脚手架 `cli` 、一些预设转换规则 `preset` 、语法兼容模块 `polyfill` 和插件 `plugin` 等。核心包主要有 `@babel/core` 、 `@babel/parser` 、 `@babel/traverse` 、 `@babel/generator` 等。

二面

总时长：90min

果然该来的躲不掉，一面没有写代码，所以二面写了三道代码题，真是给人写麻了。三道题都没有写的很顺利，只能说思路大致OK，不过对于能否通过而言感觉还是比较危险的。

不过二面面试官挺和蔼的，笑呵呵的，面试过程中没有任何挑战的言辞或问题，整体聊下来还是比较舒服。

聊项目

> 聊项目的过程跟前面以及其他家的面试过程差不多，这里就不详细介绍了，基本上没有问到具体的技术细节点，项目聊完之后就开始写代码了。

【代码题】大数相加

> 输入：num1 = '1234567890', num2 = '987654321'
>
> 输出：'2222222211'

```
1 js
2 复制代码
3 //  这是一道比较简单的题目，主要就是模拟加法进位的实现const add = (num1, num2) => {  const n = Math.max(num1.length, num2.length);    //  逆序一下，从最后一位开始相加，同时前面位数不够的补0    const arr1 = num1.split('').reverse();    const arr2 = num2.split('').reverse();    const result = [];    //  进位的标识，只有进1位和不进位两种情况    let temp = 0;    for (let i = 0; i < n; i++) {        const a = +(arr1[i] || 0);        const b = +(arr2[i] || 0);        let current = a + b + temp;        if (current >= 10) {            temp = 1;            current -= 10;        } else {            temp = 0;        }        result.push(current);    }    //  考虑到进位导致的量级增加情况，需要额外处理    if (temp) {  result.push(temp);    }    return result.reverse().join('');}
```

【代码题】实现一个同步的sleep方法

> 调用方式：(new LazyLog()).log(1).sleep(1000).log(2)
>
> 输出：先输出1，延迟1秒后输出2

```
1 js
2 复制代码
3 //  一开始我的想法是通过Promise去实现sleep，后来发现Promise的话无法满足直接的链式调用方式//  面完之后下来查了一下，发现可以通过死循环去实现同步的sleep，但是这种方式对性能有极大的影响//  在某些环境下会执行报错，后来又去查了一下开源的sleep库
```

```
https://www.npmjs.com/package/sleep//  发现它的最终也是通过c++原生代码编译成node模块来实现的，所以这个问题有没有啥更好的答案呢🤔class LazyLog {  log(str) {  console.log(str)    return this;  }    // async sleep(delay) {  //   await new Promise(resolve => setTimeout(() => resolve(), delay));  //   return this;  //  }    sleep(delay) {    const current = Date.now();    while (Date.now() - current < delay) {      // 什么都不做    }    return this;  }}
```

【代码题】按照Z字型打印矩阵

> 输入：[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]
>
> 输出：1 2 5 9 6 3 4 7 10 13 14 11 8 12 15 16

```
1 js
2 复制代码
3 //  这个题我是没有想到更好的思路，感觉就是纯暴力破解//  从左上角的点开始，先往右上角走，走到边界的时候就开始往左下角走，到边界再换方向//  核心的点就是对于边界的判断要准确，这个我在面试的时候没写出来，复盘也试了好久才正确🤦const print = arr => {  let i = 0;  let j = 0;  let rows = arr.length;  let cols = arr[0].length;  let gotoRightTop = true;  const result = [];  //  最后一定是超出行或者列了  while (arr[i] && arr[i][j]) {    result.push(arr[i][j]);        //  往右上角走的时候，列要加1，行要减1  if (gotoRightTop) {      j++;      //  如果列出界了，说明该往下移动一个并且更换方向    if (j > cols - 1) {        gotoRightTop = false;        i++;        j = cols - 1;        continue;      }      i--;      //  如果行出界了，那么需要回到界内且更换方向      if (i < 0) {        gotoRightTop = false;        i = 0;      }    //  这是往左下走的情况，逻辑是一样的    } else {      i++;            if (i > rows - 1) {        gotoRightTop = true;        j++;        i = rows - 1;        continue;      }            j--;      if (j < 0) {        gotoRightTop = true;        j = 0;      }    }  }  return result;}
```

三面

总时长：80min

三面的级别高一些，整体聊的感觉就严厉一点，不过也问了好多具体的技术细节问题和写了两道代码题。问题的难度都不大，但是自己的回答可能会让面试官觉得不够深入，通过应该没什么问题，就是担心评价估计不是特别高。

在这家公司时在架构上做了什么事情

> 脚手架
>
> 1. 提供前端脚手架工具，支持一行代码自动完成项目创建，同时调用gitlab的API完成远程仓库的创建，最后自动生成相应的CI/CD脚本实现自动化部署

2. 将项目的配置项进行收敛，包含 `ESLint`、`Prettier`、`TSConfig`、`Vite` 的等，将标准的配置文件全部内置在脚手架当中，只提供部分配置项以单独的脚手架配置项透出
3. 提供自动化命令，包含代码格式化、质量检测、本地开发、生产打包等

框架

1. 前端框架部分主要是对一些公共模块和服务进行了单独的封装，包含请求模块、状态管理、路由等，所有的功能都由框架导出给开发者直接调用
2. 提供了业务通用能力的封装，如PDF预览、统一图表展示、富文本编辑器等

组件

1. 组件部分主要是基于 `Antd` 去做一些样式和改造以及更上层的组件封装
2. 对常见的CRUD页面封装成模板，并提供 `JSON2Page` 的使用方式，以实现通过 `JSON` 配置直接生成页面

在团队提效上做了哪些事情

从项目的全部生命周期来看：

1. 创建项目 —— 提供了脚手架工具，能够秒级完成新项目创建
2. 开发项目 —— 通过提供标准化的组件和模板，以及 `JSON2Page` 的方式来进行提效
3. 部署项目 —— 通过 `gitlab` 仓库的 `CI/CD` 实现了项目的自动部署测试环境和一键手动上线

如何评定团队成员的绩效

我不直接负责每个成员的绩效，团队管理扁平化，绩效由整个大团队的leader来评定，但是会参考我的意见。在这样的团队人数情况下，我会直接根据我认为的每个成员的能力和表现来评定结果。

如何保证项目质量

保证项目质量主要考虑两个方面，一个是代码质量，一个是交付质量。

代码质量问题最多，也是最容易去做的，主要是通过 `Jest` 去做代码的单测以及通过 `Cypress` 去做UI的自动化测试。这一块就不详细展开了，具体的使用方式就是看官方文档，然后编写对应的测试用例。

交付质量这一块通常是与开发同学的个人意识和对需求的理解程度有关，之前经常会出现开发同学提测之后，QA发现主流程都走不通的情况。为了解决这个问题，我有两种措施，一个是需求提测前我会主动去过一遍需求的主流程看是否有问题，第二个则是要求开发同学自己写一份需求测试用例，在提测前需要自测通过该用例。

做过哪些技术推动业务的事情

> 这个就主要讲了一下做 `业务指标监控` 的事情，详细的内容在其他面经里都有，这里就不具体展开了。 【前端面经】2023面试复盘之字节跳动 【前端面经】2023面试复盘之美团

讲一下setState之后发生了哪些事情

> 先讲 `React` 的架构，包含了 `Renderer` 、 `Scheduler` 和 `Reconciler` 三部分，然后具体说了每一部分大概是做什么，之后讲 `setState` 其实就是触发组件的一次渲染过程，具体过程如下：
>
> 1. `setState` 会生成一份新的组件内状态数据并重新执行 `Reconciler` 中的 `render` 方法
> 2. `render` 方法会根据 `JSX` 和最新的数据去创建一个新的 `fiber` 节点树，每一个树节点的创建都是 `Reconciler` 中的一个工作单元
> 3. 所有的创建 `fiber` 节点工作单元生成后，这些工作单元的执行和调度会由 `Scheduler` 中的任务队列来执行
> 4. 任务队列每次取出一个创建 `fiber` 节点的任务执行，执行完成之后会调用浏览器的 `requestIdeCallback` 方法来判断当前刷新帧剩余时间是否够执行下一个任务
> 5. 如果时间够就执行下一个创建 `fiber` 节点任务，不够的话就先将创建任务暂停，等下一个刷新帧继续执行
> 6. 当所有的创建任务都执行完成之后，就生成了一棵新的 `fiber` 节点树，之后就是通过新旧两棵树去做 `diff` 算法获得要更新的树，后面的 `diff` 和渲染部分这里就不多介绍了

【代码题】实现一个实时搜索框组件

> 这道题每次都是我面外包的时候让写的，没想到有一天我也会自己做这个😂
>
> 这道题没有什么特殊的要求，就如题目所示，通过 `React` 实现一个实时搜索框组件即可，剩下的就是自由发挥了。

```
jsx
复制代码
const SearchBox = ({ onChange }) => {    const lockRef = useRef(0);    const [searchList, setSearchList] = useState([]);    const onInput = async e => {        lockRef.current += 1;        const temp = lockRef.current;        try {            const res = await fetch("/api/search", e.target.value);            // 处理竞态条件            if (lockRef.current !== temp) return;            setSearchList(res.json());        } catch (error) {            console.log(error);        }    };    return (        <div className="search-wrapper">            <input type="text" onInput={onInput} />            <ul className="complete-list">                {searchList.map(item => (                    <li key={item.value} onClick={onChange(item)}>                        {item.label}                    </li>                ))}            </ul>        </div>    );};
```

【代码题】查找有序数组中数字最后一次出现的位置

> 输入：nums = [5,7,7,8,8,10], target = 8
>
> 输出：4

```
1  js
2  //  最简答的方式就是直接遍历然后根据有序的条件找到当前值等于目标且下一个值不等于目标的结果
3  //  写出来之后面试官问了时间复杂度，这个就是单层循环的 O(N)，最坏情况就是刚好最后一个值是
   目标值
4  const findLast = (nums, target) => {
5      for (let i = 0; i < nums.length; i++) {
6        if (target === nums[i] && target !== nums[i + 1]) {
7          return i;
8        }
9      }
10     return -1;
11 };
12
13 //  问有没有更好的方式，就想到了二分查找，对于已经有序的数组，只需要通过双指针不断更新左右
   边界位置就行
14 //  二分法最主要的就是寻找二分结束的边界条件，这里选择所有的查找最后都只剩两个值
15 //  然后对这两个值再额外判断一下是否符合结果
16 //  面试官继续追问二分法的时间复杂度，这个我有点懵，不过考虑跟递归差不多，所以就回答了
   O(logN)，应该是没错
17 //  二分查找最坏的情况是刚好第一个值或者最后一个值，或者中间值是目标值
18 const findLast2 = (nums, target) => {
19     let left = 0;
20     let right = nums.length - 1;
21     while (right > left + 1) {
22         const mid = Math.floor((left + right) / 2);
23         if (nums[mid] > target) {
24             right = mid - 1;
25         } else {
26             left = mid;
27         }
28     }
29     if (nums[right] === target) {
30         return right;
31     }
32     if (nums[left] === target) {
33         return left;
34     }
35     return -1;
36 };
37
```