[前端面试题总结] ⭐ 从莉莉丝到滴滴--我的成长之路

「面经分享」

» 莉莉丝 凉经

复制代码

莉莉丝是一家游戏公司，所以考察相对来说更注重思维能力，面试官会比较积极的引导，答不出来面试官也会讲解一遍。

- 瀑布流列表对于请求过的数据是否做优化？了解虚拟滚动(虚拟列表)吗？

> 这题是基于我的项目介绍问的。虚拟列表是在**处理用户滚动时，只改变列表在可视区域的渲染部分**，它可以提高页面渲染性能，减少数据过多时的卡顿。

下面是一个简单的虚拟列表demo：

js 复制代码

```
function FixedSizeList(props){
    const {{containerHeight,itemCount,itemHeight}} = props; // 容器高度，列表数量，列表项高度
    const [scrollTop,setScrollTop] = useState(0)
    const items = [];
    // 渲染列表第一个index
    let startIdx = Math.floor(scrollTop / itemHeight);
    // 渲染列表最后一个index
    let endIdx = Math.floor((scrollTop+containerHeight)/itemHeight);
    // 在可视区外缓存列表项数量
    const paddingCount = 2;
    startIdx = Math.max(startIdx - paddingCount,0);
    endIdx = Math.min(endIdx + paddingCount,itemCount-1)

    for(let i = startIdx;i <= endIdx;i++) {
        items.push(<Component key={i} index={i}
                style={{height:itemHeight}}/>)
    }
    const top = startIdx * itemHeight;
    const contentHeight = itemHeight * itemCount;

    return (
        <div style={{height:containerHeight,overflow:'auto'}}
            onScroll={
                (e) => {
                    // 处理渲染有导致的白屏问题
```

```
                    // 改为同步更新
                    flushSync(() => {
                        setScrollTop(e.target.scrollTop)
                    })
                }
            }
        >
            <div style={{height:contentHeight}}>
                <div style={{height:top}}></div>
                {items}
            </div>
        </div>
    )
}
```

- 重排重绘

> **重排**：当 DOM 的变化影响了元素的几何信息(DOM 对象的位置和尺寸大小)，浏览器需要重新计算元素的几何属性，将其安放在界面中的正确位置，这个过程叫做重排。修改元素的尺寸、增加删除可见的元素、操作 Offset 相关属性等都会引起重排。
> **重绘**：当一个元素的外观发生改变，但没有改变布局,重新把元素外观绘制出来的过程，叫做重绘。如修改元素的颜色等。
> **如何减少重排、重绘：**
>
> 1. 分离读写操作：即在使用 offset 相关属性进行读操作之前，完成所有改变 DOM 的写操作。
> 2. 集中改变样式：将样式统一写在一个类名上，再将该类名加到属性上。
> 3. 离线修改 dom ：在要操作dom之前，通过display隐藏dom，当操作完成之后，才将元素的display属性为可见；或者通过使用DocumentFragment创建一个 dom 碎片,在它上面批量操作dom，操作完成之后，再添加到文档中，这样只会触发一次重排。
> 4. 设置 position 属性为 absolute 或 fixed
> 5. 启用 gpu 加速：transform: translate3d(10px, 10px, 0);

- 虚拟DOM
- 类组件/函数组件开发区别
- 实现自定义hook (对可以上下/左右滑动的滑块定义hooks)

平时自己写自定义hooks比较少，所以这题面试官引导了挺久，还是没写出来/(ㄒoㄒ)/

js 复制代码

```
const onSlide = (dir) => {
   console.log(dir)
 }
 const { domRef:domRef1 } = useSlide(onSlide,'v');
 const { domRef:domRef2 } = useSlide(onSlide,'h');
```

```
  return (
        <div className="App">
          <input type="range" ref={domRef1}/>
          <input type="range" ref={domRef2} style={{transform:'rotate(-90deg)',marginTop:'55px'
        </div>
      )
```

js 复制代码

```
// 自定义 hooks  useSlide
const useSlide = (cb,direction) => {
    const domRef = useRef(null);

    const handleUp = (e) => {
        if(direction==='v'){
            // cb(e.clientX)
            cb(e.target.value);
        }else if(direction==='h'){
            // cb(e.clientY)
            cb(e.target.value)
        }
    }

    useEffect(() => {
        domRef.current.addEventListener('mouseup',handleUp)
        return () => {
            domRef.current.removeEventListener('mouseup',handleUp)
        }
    },[])

    return {
        domRef
    }
}
```

- js数据类型，简单数据类型和复杂数据类型的区别
- 原型链 原型对象
- Promise与js事件机制
- 数组扁平化(堆栈写法，非递归)

js 复制代码

```
const flatter = (arr) => {
    let stack = [...arr];
    let res = [];
    while(stack.length){
        let num = stack.pop();
        if(Array.isArray(num)) {
```

```
                stack.push(...num);
            } else {
                res.push(num);
            }
        }
        return res.reverse();
    }
```

- 玩石子游戏(算法)
  一共n颗石子，两个人轮流拿，每次可以拿1或2或3颗，最后一次将石子取完的人输，判断自己最后是赢是输?

> 我一开始想能不能用动态规划去解，面试官说我想复杂了。当最后剩 4 颗石子的时候，无论我们怎么取都可以获胜；当剩 8 颗石子的时候，我们一定可以控制最后剩下 4 颗石子...所以我们只要**判断石子的数量是否是 4 的倍数** 即可判断自己最后是否可以获胜。

» 哈啰 已oc

面试官非常年轻，好像就已经是技术专家了，好羡慕，人也很好，整体面下来的感觉还是不错的。

复制代码

- cors跨域原理

> 通过设置响应头 `Access-Control-Allow-xxx` 字段来设置访问的白名单、可允许访问的方式等

```js
module.exports = async (ctx,next) => {
    // 设置响应头
    // 允许访问跨域服务的白名单 *允许所有
    ctx.set('Access-Control-Allow-Origin','*');
    ctx.set('Access-Control-Allow-Headers','Content-Type');
    ctx.set('Access-Control-Allow-Methods','GET,POST,OPTIONS');
    // OPTIONS 获取资源允许访问的方式
    // 其实在正式跨域之前浏览器会根据需要发起一次预检也就是option请求用来让服务端返回允许的方法
    await(next())
}
```

- 跨域拦截 是浏览器拦截还是服务器拦截

> 跨域是浏览器的同源策略造成的，同源是指"协议+域名+端口"三者相同，为了保证浏览器安全对响应的数据进行拦截，若发现是非同源的资源浏览器进程会把响应体丢弃。所以**跨域拦截是浏览**

> **器进行拦截。**
>
> 跨域是为了阻止用户读取到另一个域名下的内容，**跨域并不是请求发不出去，请求能发出去，服务端能收到请求并正常返回结果，只是结果被浏览器拦截了。**Ajax 可以获取响应，浏览器认为这不安全，所以拦截了响应。通过表单的方式可以发起跨域请求,因为表单并不会获取新的内容，所以可以发起跨域请求。

- 假设我现在在淘宝，要去百度，我还会携带上淘宝的cookie吗?

> 不会。**在 `cookie` 中， `domain` 属性指定了 cookie 的所属域名， `path` 属性用来指定路径**；这两个属性决定了服务器发送的请求是否带上这个 `cookie` 。淘宝的 cookie 和百度的域名、路径不一致，所以访问百度不会带上淘宝的 `cookie` 。

- 淘宝跳转到天猫页面为什么不需要重新登陆(taobao.com 和 tmall.com)?

> 这是因为淘宝和天猫页面设置了**单点登录SSO**，在多个系统的集群里，用户只需一次登陆，其他系统也会感知到用户登陆，不需要用户重新输入用户名密码登陆。
>
> 在淘宝和天猫之间部署一台专门用作登录的服务器，相当于实现一个**中转站**，这个中转站存放着需要共享登录状态服务应用的 `session` 信息，它就相当于一把钥匙，当用户进行应用跳转的时候都会来中转站看一看，如果 `session` 存在就直接使用这把钥匙把通往另一个应用的门打开，如果不在就得先确认用户身份(登录)制造 `session` 钥匙存在中转站再把门打开，用户的身份存储和身份的有效时间都是中转站说了算，统一管理的。

- http协议的特点

> 1. 灵活方便：支持传输多种形式的内容
> 2. 可靠传输：基于tcp/ip
> 3. 请求-应答：具有发送方和接收方
> 4. 无状态：每次请求都是独立无关的,不会记住上一次的状态

- BEM 命名规范

> B:block E:element M:modefiled 语义性好，模块化，组件化，可复用
> 这是面试官看到我的掘金第一篇文章来问的，有兴趣的朋友可以看看哦。BEM-实战淘宝订单模块

- flex:1的含义， flex:1和flex:2的区别

> **flex 是 flex-grow ， flex-shrink ， flex-basis的缩写**
> `flex:1` 即
> `flex-grow:1` 若存在剩余空间根据剩余空间进行放大；

> `flex-shrink:1` 若空间不够，将内容缩小；
> `flex-basis:0%` 内容本来大小

- var a = []，为什么可以直接使用a.push()，a.pop()

> a 通过对象字面量的方式生成一个数组对象实例，可以通过原型链查找方法，使用到 `Array` 原型上的 `push` 和 `pop` 方法。

- var a = new A()，a 和 A 的关系，A 和 Function 的关系

> a 是 A 的一个实例对象，a 的 `__proto__` 指向 A 的 `prototype` ，A 的 `__proto__` 指向 Function 的 `prototype` 。

- 浏览器事件机制 event loop
- Primise.then 中的微任务在下一轮还是这一轮执行
- commonjs之后为什么提出es molule，两个模块的特点

> `commonjs` 导出(module.exports)，导入(require)。
> 可以动态加载语句，**发生在代码运行时**，模块是**同步加载**的，只要加载完成才能进行后续操作；模块可以多次加载，但只会在第一次加载时运行一次，结果会被缓存，下次 加载直接读取缓存结果，除非清除缓存。**导出的值的拷贝的**，可以修改，模块内的修改不会影响拷贝的值，代码出错不好排查。
> `esmodule` 导出(export default)，导入(import from)
> 是**静态**的，只能声明在文件最顶部，**发生在代码编译时**；导出的值存在**映射关系引用**，是可读的，不可修改

- 函数式组件和类组件的区别
- usestate 重复修改多次，会渲染多次吗? 什么时机触发渲染?

> - 如果 `useState` 在**同步**代码中重复修改多次，只会渲染一次。因为出于性能考虑，React 会把多个 `setState()` 调用合并成一个调用，这也是 `state` 的**批量更新机制**。React 会将该 state “冲洗” 到浏览器事件结束的时候，再统一地进行更新，这时才会触发渲染。
> - 而如果 `setState` 在**异步**代码(如 setTimeout、async await)中重复修改多次，渲染会在每次修改后触发一次。

- redux
- 听说过基于redux的封装吗?

- 为什么本地开发更改一行代码，浏览器会局部更新那一部分？浏览器怎么知道代码变了？

> 这题触及到我的知识盲区了，于是面试官跟我讲解了一下，赞！
> 脚手架npm run dev后，会启动一个本地的服务 localhost，它基于 `socket` 连接本地编辑器和浏览器的通信，监听文件系统是否发生改变，当发生改变时 `webpack` 会告诉浏览器代码发生更改，浏览器就会做相应的渲染。这块应该是热更新的知识点。

- 装饰者模式
- ts 中 record是做什么的?

> `Record<K,T>` 构造具有给定类型 `T` 的一组属性 `K` 的类型。他会将一个类型的所有属性值都映射到另一个类型上并创造一个新的类型.

实例：

```js
interface EmployeeType {
    id: number
    fullname: string
    role: string
}

let employees: Record<number, EmployeeType> = {
    0: { id: 1, fullname: "John Doe", role: "Designer" },
    1: { id: 2, fullname: "Ibrahima Fall", role: "Developer" },
    2: { id: 3, fullname: "Sara Duckson", role: "Developer" },
}

// 0: { id: 1, fullname: "John Doe", role: "Designer" },
// 1: { id: 2, fullname: "Ibrahima Fall", role: "Developer" },
// 2: { id: 3, fullname: "Sara Duckson", role: "Developer" }
```


- 项目后台搭建为什么选择koa？为什么不是express？
- 说说个人的亮点

» 滴滴 已oc

感觉面试官问的问题可回答面比较广，所以可以挑自己擅长的领域说~ 而且面试官比较和蔼，让我不要紧张。面试官直接就跟

- css布局方式

> 说了以下四种布局方式，然后以水平垂直居中为例，具体的说了下每种布局方式是如何使用的。
>
> 1. flex
> 2. grid
> 3. float
> 4. position

- float离开文档流怎么做处理

> 我大概说了下**清浮动、高度塌陷**的不同处理方式，顺便说了下BFC

- position属性，分别相对谁定位，具体使用场景
- js 数据类型 Symbol使用场景
- 类型判断方法

> 我说了三种 **typeOf、instanceOf、Object.prototype.toString.call()** 以及他们各自的优缺点
> 面试官接着问了**instanceOf 原理**，说完继续问了下面两个问题
> Array instanceOf Array == ?(false)
> Object instanceOf Object == ?(true)

- 数组api，以及在项目使用场景
- forEach map区别 使用场景

> - `forEach` 和 `map` 都可以对数组的进行遍历。
> - `forEach` 返回值为 `undefined` ，所以不能进行链式调用，适用于不更改数组的情况；
> - `map` 会返回一个新数组，这个新数组由原数组中的每个元素都调用一次提供的 `callbackFn` 函数后的返回值组成。一般用于需要修改数组的情况。
> - 对于数组项是基本数据类型的数组， `forEach` 和 `map` 都不修改调用它的原数组本身，但是那个数组可能会被 `callbackFn` 函数改变; 对于数组项是引用数据类型的数组，用 `forEach` 和 `map` 都可以改变它的属性值。
>
> **又问：给对象数组中对象的每一项加一个属性，用 forEach 还是 map?**
> 这里用 `forEach` 会更好，直接在原数组上修改即可，如果用map，还要将返回的结果重新赋值给原来的数组变量，语义不好。

- 手写代码 将对象数组转为对象

```
const array = [
{
```

```
    code: 101,
    name: '北京',
    },
    {
    code: 102,
    name: '石家庄',
    },
    {
    code: 102,
    name: '江苏',
    children: [{
        code: 102,
        name: '南京',
    },{
        code: 102,
        name: '连云港',
    }]
    }
]
```

转换成：

js 复制代码

```
  {
    '北京': {
      code: 101,
      name: '北京'
    },
    '石家庄':{
       code: 102
       name: '石家庄'
    },
    '南京':{
       code: 102
       name: '南京'
    },
    '连云港':{
       code: 102
       name: '连云港'
    }
  }
```

实现代码：

js 复制代码

```
function toObj(arr) {
    let obj = {};
    for(let item of arr) {
        if(item['children']!==undefined) {
```

```
            obj = {...toObj(item['children']),...obj};
        }else {
            obj[item.name]=item;
        }

    }
    return obj;
}
```

- 根据代码看输出 this问题

```js
const obj1 = {
    hello: function () {
        console.log(this);   // obj1
        setTimeout(function () {
            console.log(this);   // window
        });
    }
}
obj1.hello();


const obj2 = {
    hello: function () {
        console.log(this);   // obj2
        setTimeout(() => {
            console.log(this);   // obj2
        });
    }
}
obj2.hello()

const obj3 = {
    hello: ()=> {
        console.log(this);   // wondow
        setTimeout(() => {
            console.log(this);    // window
        });
    }
}
obj3.hello()
```

- 闭包 以及项目中的使用
- 防抖节流
- vite webpack 了解多少

- js模块化 commonjs esmodule区别
- UMD AMD CMD

  > - **AMD (Asynchronous Module Definition 异步模块定义)**
  >
  > commonjs是同步加载的，不适用于浏览器环境，所有有了AMD CMD； AMD是异步加载的，模块的加载不影响后面语句的运行。所有依赖这个模块的语句都定义在一个回调函数中，等 加载完之后，这个回调才会运行。AMD 的模块引入由 define 方法来定义
  > - **CMD (Common Module Definition)**
  >
  > 对于依赖的模块，AMD 是提前执行，CMD 是延迟执行。CMD 推崇依赖就近，AMD 推崇依赖前置。
  > - **UMD (Universal Module Definition 通用规范模块)**
  >
  > 统一浏览器端(AMD)和非浏览器端(commonjs)的模块化方案。
  >
  > 1. 先判断是否支持AMD(define是否存在),存在则使用AMD方式加载模块；
  > 2. 再判断是否支持commonjs(exports是否存在)，存在则使用commonjs模块格式
  > 3. 前两个都不存在，则将模块公开到全局

- 浏览器缓存 怎么做demo的

  > 主要针对的是前端静态资源(js css image)，大大的减少了请求的次数，提高了网站的性能(两端)
  >
  > - **强缓存** 设置http响应头
  >
  > 1. http1.0版本： `Expires` 具体时间点，客户端时间不准可能会导致误差
  > 2. http1.1版本： `Cache-Control`：`max-age=xxx` 时间偏移量 倒计时
  >
  > - **协商缓存**
  >
  > ▪ Last-Modified 会经常更改的数据，不变则发送304。请求数量不变，请求体积减小
  >
  > 1. 设置响应头： `Last-Modified` 文件最近更改时间
  > 2. 判断请求头： `if-modified-since` == `Last-Modified` 发送304
  >
  > ▪ etag
  >
  > 根据文件生成哈希串
  >
  > 1. 设置响应体: `Etag`
  > 2. 判断请求头： `if-none-match` == `Etag` 发送304

- nginx
- 跨域 jsonp缺点
- 项目后端做了啥
- 异步并行/串行

  问 异步事件1、异步事件2是并行还是串行？怎么变为并行？

```js 复制代码
async() {
    await 异步事件1
    await 异步事件2
}
```

> 这里我说的是**让异步事件不写在 `await` 后面**，这是一种实现方式。当然还有更好的实现方式，所以面试官问了下一题。

```js 复制代码
async ()=>{
    let result1 = 异步事件1
    let result2 =  异步事件2
    let res1 = await result1
    let res2 = await result2
    console.log(res1,res2)
}
```

- promose.all 和 promise.allSettled

> 先说怎么用 `promise.all` 解决上一题吧

```js 复制代码
async ()=>{
    var result = await Promise.all([异步事件1,异步事件2])
    console.log(result[0],result[1])
}
```

> 这么写确实可以解决上题的问题，但是在 `promise.all` 中如果有一个异步事件出错，那么将会返回一个失败的 `promise` 。所以如果异步事件互不依赖的话使用 `promise.allSettled` 可能会更好，无论每个异步事件执行成功或失败都会返回一个带有执行结果的 `promise` 。 > 想了解更多更多 `promise.allSettled` 可以看这里~

```js 复制代码
async ()=>{
    var result = await promise.allSettled([异步事件1,异步事件2])
    console.log(result[0],result[1])
}
```