23 届校招前端面经分享

作者：dossweet, 公众号： Coding101, github： dossweet

署名时间：2022 年 12 月 29 日 14:00

1. 2022 年 2 月 21 日 阿里一面

1.1 面试问题汇总

1. 简单介绍一下对面试部门的了解
2. 进程、线程和协程之间的区别与联系
3. 既然已经有了线程，为啥还需要协程？--协程是更轻量级的线程，相比于操作线程来说，操作协程所需的性能消耗更小，从而用户体验更好
4. 进程之间如何进行通信？
5. 什么是 TCP，以及 TCP 与 UDP 之间的概念和对比
6. HTTPS 与 HTTP 的对比
7. 为什么服务器传递过来的证书可以证明是经过权威部门认证过的？--因为浏览器中已经有权威部门颁发的公钥了，这个是不经过服务器的
8. 二叉树的结构特征，应用场景和遍历方式
9. JS 是单线程的，那页面如何保证流畅性，不让用户觉得卡顿？--异步执行，利用了事件循环机制
10. 什么是宏任务和微任务
11. 为什么要有宏任务和微任务？-优先级
12. 节流和防抖的区别
13. 手撕题--千位分隔符，注意有小数部分

```
function splitNumber(str){
    if (isNaN(str)) return 'please input a number!';
    let num = str.split('.');
    let ret;
    if (num.length > 1){
        ret = addSplit(num[0])+'.' + addSplit(num[1]);
    }else{
        ret = addSplit(num[0]);
    }
    return ret;
}

function addSplit(str,flag){
    let arr = flag?str.split('').reverse():str.split('');
    arr.unshift(0);
    let ret = [];
    for (let i = 1; i < arr.length; i++){
        ret.push(arr[i]);
        if (i % 3 === 0){
            ret.push(',');
        }
    }
    return flag?ret.reverse().join(''):ret.join('');
}

console.log(splitNumber('gggg.hhh'));
console.log(splitNumber('13879026728.2234421212'));
console.log(splitNumber('13879026728'));
```

1.2 复盘

八股文相关回答的都很好，回答到了点子上。但是手撕题写的不好。。

1. 没有编程风格
2. 没有采用 ES6 新语法，比如 reverse 等
3. 下标没有从 0 开始
4. 有更好的解题方法：正则表达式
5. 做题之前先和面试官互动，讲一下解题思路

2. 2022 年 2 月 23 日 阿里二面

2.1 面试问题汇总

1. **从什么时候开始接触前端的？是如何接触前端的？**
2. **React 框架和 Vue 框架的不同点和相同点，从设计理念上说**
3. **JQuery 和 React 框架和 Vue 框架的对比**
4. **介绍一下项目和项目难点**
5. **刚才讲到了微信图片防盗链，那如果是你，你会怎么处理来阻止防盗**

2.2 面试复盘

1. 学知识要深入
2. 做完一个任务后，要学着总结和反思

3. 2022 年 2 月 28 日 微软一面

3.1 面试问题汇总

1. **自我介绍**
2. **介绍项目**
3. **手撕构造一颗二叉树**
4. **手撕一个链表**
5. **手撕在 N 范围内寻找指定个数的质数**

3.2 面试复盘

1. 要重视计算机基础，数据结构这些基本的要会

4. 2022 年 3 月 2 日 微软二面

4.1 面试问题汇总

1. 英文自我介绍

2. 前端和后端有什么区别

3. 讲一下服务器和客户端分别运行在什么端口

4. 讲一下虚拟地址和物理地址

5. 讲一下编译和解释的区别

6. 讲一下如果有一个很大的文件，里面全是单词，我要怎么统计出每个单词出现的频率？

7. 那如果这个文件很大的话，大到操作系统都打不开呢

8. 用不同的服务器如何进行操作？

9. 寻找一个数组中升序的三个数

4.2 面试复盘

1. 算法很重要，第 9 个问题用回溯法直接可以解的出来的。。对于一个题，我们优先考虑的就是用暴力来解题，你刷了那么多类型的题，应该在做题时想想属于哪个类型的题，然后套用模板来解题呀~
2. 虚拟地址和物理地址

程序所使用的地址，称为逻辑地址。

通过段式内存管理映射的地址，称为虚拟地址

逻辑地址和虚拟地址在操作系统的物理内存中不真实存在，他们最终都将由操作系统和 CPU 硬件翻译成真正的物理地址，然后才能从真实的物理位置获取该地址的值。

物理内存都是通过 CPU 来进行访问的。CPU 访问物理地址的过程可以描述为物理寻址的过程。

因为物理地址存在于操作系统中，因此直接访问是十分消耗计算机性能的。因此就提出了虚拟地址的概念，

操作系统会给每个进程分配一个虚拟地址空间，这个虚拟地址空间不同于其他进程。每个进程包含的栈、堆、代码段这些都会从这个地址空间中被分配一个地址，这个地址就称为虚拟地址。

那么为什么虚拟地址可以使得不同的进程不会同时访问同一个物理地址，而没有虚拟地址的话，不同的进程就可能访问到同一块物理地址呢？

因为虚拟地址其实由物理地址分出来的。

我们把物理地址（也就是内存）分成很多的块，这些块就叫做页，这个过程就叫内存分页。分出来的这些页呢合起来就是页表，虚拟地址就存在于页表中的每一个页之中。然后我们再给不同的进程分别分配不同的虚拟地址，从而就可以避免不同的进程访问到同一块物理地址。

因此物理地址才是真正的内存。

虚拟地址不在内存中，因此虚拟地址的空间大小也可以比物理地址的空间大小大（最大为4G）。这也就是程序的局部性原理

虚拟地址包含两个部分：用户态和内核态，内核态主要干的事情就是系统调用。因为进程不能直接操作物理地址，都是通过虚拟地址来映射过去的，因此物理地址和虚拟地址都会划分出一块区域给内核态，因此，即使是不同的虚拟地址，他们的内核态都是一样的，且都指向同一个物理地址的内核态。一般虚拟地址会留 1/3 的空间给内核态，其余为用户态。

在我们平常的开发中，用户态就可以满足需求了，但是我们前面也提到了，程序想要访问内存的话，就需要从虚拟地址映射为物理地址，但是很多时候，我们运行的程序使用虚拟内存就可以满足要求了。我们平常用的堆和栈，其实都是存在于虚拟内存之中的。

CPU 可以通过生成一个虚拟地址来访问主存，这个虚拟地址被送到内存之前会先被转换成合适的物理地址，这个虚拟地址到物理地址的转换过程就称为地址转换。实现是通过 MMU（内存管理单元）来实现的

5. 2022 年 3 月 4 日 微软三面

5.1 面试问题汇总

1. 面试官看了我的博客，对我提出了表扬

2. 一道场景设计题：一个拥有 2 万个成语的数组，来玩成语接龙游戏，使得接龙足够长

我和面试官全程就讨论了这个题，大概讨论了有 10 分钟，就开始让我自己写了，其实我写的也有不对的地方，只不过大体的框架已经写出来了，面试官相对比较满意

3. 反问环节，问了下苏州微软这边的业务

大概介绍了一下 leader 所在的部门业务

5.2 面试复盘

有自己的博客和开源项目真的很有优势

6. 2022 年 3 月 5 日 美团笔试

五道编程题

1. 从一个数组中找出最长子集，这个子集中的元素不能是相邻的元素，即第一个元素是 1 的话，第二个元素就不能是 2

   解题思路：使用回溯法暴力解。

   刚开始过了 17%，后面发现回溯之前忘记排序了，因此先排了下序，再进行回溯。

   最终只过了 27%。

2. 最大子段和，数组内的元素允许翻转一次，求最大子段和。

   其实如果全是正数的话，数组之和就是最大字段和。

   最主要的问题是，有负数时，需要进行翻转来保证目前区间里的字段和最大

   解题思路：用双指针。慢指针用于找第一个正数，快指针用于找尾部的元素。与此同时，当快指针找最后的元素过程中，遇到的第一个负数忽略不计，后续如果在遇到负数的话，就判断负数加后面的元素是否非负，如果非负的话，就保留，否则该子段终止，慢指针向后移动至第一个正数。快指针指向慢指针的下一个元素，重新开始计算子段和。

   这种解法有点问题感觉。感觉正确的做法是用动态规划。

3. 切豆腐，保证每次切一刀，剩下的豆腐体积最大。

   不会

4. 忘记啥题了，反正很复杂

5. 前端方向的题目，根据线索来猜字符串

   第一行输入两个元素，第一个元素代表要猜的字符串的长度 s，第二个元素代表要猜的轮数 m

   第二行开始包含三个元素，第一个元素是猜测的字符串，第二个元素是猜对的个数，第三个元素是猜错位置但存在于字符串中的元素。

   一共有 m 行，要求我们通过这 m 次猜测，推断出字符串的内容。如果推断不出来，就输出?

7. 2022 年 3 月 8 日 字节一面

7.1 面试问题汇总

1. 什么时候开始接触前端的

2. 为什么想要做前端

3. 平常是如何学习前端的

4. HTML 语义化标签

语义化标签，旨在让标签有自己的含义，优势是：

（1） 使得在没有 CSS 的情况下，页面也能呈现出很好的内容结构、代码结构

（2） 有利于 SEO：和搜索引擎建立良好沟通，有助于爬虫抓取更多的有效信息

（3） 方便其他设备（比如盲人阅读器来解析）来解析 HTML 内容，从而渲染出页面

（4） 方便团队的开发和维护，使得代码更具可读性

常见的语义化标签有：

<a>

<header>

<footer>

<h1><h2><h3><h4><h5>

<article><section>

5. HTML 的 meta 属性

meta 用于定义页面的说明，关键字，最后修改日期，和其他的元数据。这些元数据将服务于浏览器（如何布局或重载页面），搜索引擎和其他网络服务。

meta 标签共有两个属性，分别是 http-equiv 属性和 name 属性。

name 属性更为常用：**name 属性主要用于描述网页**，比如网页的关键词，叙述等。与之对应的属性值为 content，**content 中的内容是对 name 填入类型的具体描述，便于搜索引擎抓取**。meta 标签中 name 属性语法格式是：

```
<meta charset='utf-8'>
<meta name='参数' content='具体的描述'>
<meta name='viewport' content='width=device-width,initial-scale=1'>
<meta name='keywords' content='个性主页，学习，记录，成长'>
<meta name='description' content='热爱前端与编程。目前大二，这是我的前端博客'>
<meta name='robots' content='none'>
<meta name='author' content='xxx'>
<meta name='copyright' content='xxxx'>// 代表该网站为 xxx 个人版权所有
```

keywords 是网站的关键词

description 是网站的描述

上述两个属性都有助于搜索引擎读取网页。

robots 用来告诉爬虫哪些页面需要索引，哪些页面不需要索引。content 的参数有 all,none,index,follow 等，默认是 all

1. CSS 盒模型

出了道题：

w 100 b 10 p 20 m 30

问怪异盒子和标准盒子下该盒子的宽度是多少

2. CSS 的水平垂直居中的几种方式

这篇总结的特别好：https://www.cnblogs.com/formercoding/p/12826126.html

1. 利用 flex 的 align-items:center 垂直居中，justify-content:center 水平居中
2. 利用相对定位和绝对定位的 margin:auto

   父组件用相对定位，子组件用绝对定位，然后子组件内设置 top,right,bottom,left 都是 0，margin:auto 即可使子组件水平垂直居中
3. 利用相对定位和绝对定位，再加上外边距和平移的配合

父组件用相对定位，子组件用绝对定位，利用 **margin 偏移外容器的 50%**，再利用 **translate** 平移**回补自身宽高**的 50%即可

```
.box {
  width: 100vw;
  height: 500px;
  background: skyblue;
  position: relative;
}

.child {
  width: 200px;
  height: 200px;
  background-color: deepskyblue;

  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```

4. 利用 textAlign 和 verticalAlign---使行内元素水平垂直居中

利用 **textAlign:center** 实现行内元素的水平居中，再利用 **vertical-align:middle** 实现行内元素的垂直居中，前提是要先加上伪元素并给设置高度为 100%，用过 elementUI 的可以去看看其消息弹窗居中实现方式就是如此。

5. 上面都是在未知外容器和自身宽高下实现水平垂直居中的，如果已知其宽高，可以有更多种简单的方式实现居中，其原理无非是利用绝对定位的 top/left 偏移、margin 偏移、padding 填充，在此就不分析了。还有就是单纯文字的居中利用 lineHeight 和 textAlign 即可实现（这个方法是在父元素中进行设置）

3. CSS 的 position 有哪些属性

absolute--生成绝对定位的元素，相对于 static 定位以外的第一个父元素进行定位

fixed--生成绝对定位的元素，相对于浏览器窗口进行定位

relative--生成相对定位的元素，相对于其正常位置进行定位

static--默认值，没有定位，元素出现在正常的流中

inherit--规定应该从父元素继承 position 属性

sticky--粘性定位

4. JS 中 new 一个对象的过程

（1） 创建一个空的简单对象

（2） 为步骤 1 新创建的对象添加属性_proto_,该属性连接至构造函数的原型对象

（3） 将步骤 1 新创建的对象作为 this 的上下文

（4） 执行该函数，如果该函数没有返回对象，则返回 this

5. new 出来的这个对象的指向

当 new 出一个对象时，this 指向哪里

6. new 出一个对象时，构造函数的返回值类型

就是前面步骤里说的，如果这个对象指向的构造函数有返回值的话，那么它的返回类型就是根据这个返回值来的，如果没有返回值的话，就返回 this，即这个 new 出来的对象

7. JS 中的闭包概念以及应用场景

闭包的结构特性就是函数里面返回函数，

除此之外，闭包可以使得返回的这个函数有权限访问构造出这个闭包的函数内的私有变量。

闭包的应用场景有：节流、防抖、函数柯里化

8. js 获取 html 元素的方法有哪些？

document.getElementById();

document.getElementsByClassName();

document.getElementsByTagName();

document.querySelector();

document.querySelectorAll();

9. JWT 相关概念

10.token 和 cookie 也可以进行加密，为啥一定要用 JWT 呢

11.git 的基本使用

12.websocket 与 http 的区别

13.手写一个防抖

14.针对手写的防抖，为什么要用 that 来替换 this

当有多级函数时，这个 this 的指向是会一直变动的，这个问题有可能会导致我们获取元素失败。因此就需要使用 that 来保存当前的 this，防止这个问题的出现。

7.2 面试复盘

反问环节注意事项：

1. 请问贵部门，前端的技术栈都使用哪些？
2. 部门方向的未来战略是什么？
3. 目前部门方向遇到的最大困难或者难点在于哪些
4. 字节使用 react 框架，那么请问，字节在 react 框架基础上，做了一些什么样的变动，或者特点是什么

8. 2022 年 3 月 10 日 字节二面

8.1 面试问题汇总

1. 介绍一下百度训练营的项目

2. 在百度训练营的项目中承担的什么角色

3. 服务端的内容如何呈现在页面上

4. 介绍一下阿里训练营的项目

5. 承担的是什么角色

6. 使用 css 画一个 loading 的图标

```
.loading{
    width:35px;
    height:35px;
    border:5px solid rgba(189, 189, 189, 0.25);
    border-left-color: rgba(3, 155, 229, 1);
    border-top-color: rgba(3, 155, 229, 1);
    border-radius: 50%;
    animation: rotate 600ms infinite linear;
}

@keyframes rotate{
    to {
        transform:  rotate(1turn);
    }
}
```

animation 的第一个参数表示动画的名称是什么

animation 的第二个参数表示旋转的周期是多少

animation 的第三个参数表示 animation-iteration-count(动画播放次数)，它的属性值有两种：

（1） 直接写数字，自定义想要播放动画的次数

（2） infinite: 设置动画无线循环播放

animation 的第四个参数表示 animation-iteration-function(动画以何种运行轨迹完成一个周期)

（1） linear: 表示动画从头到尾的速度都是相同

（2） ease-in: 表示动画以低速开始

（3） ease-out：表示动画以低速结束

（4） ease-in-out: 表示动画以低速开始和结束

transform 表示动画的执行方式，后面跟的参数为 rotate(135deg)表示旋转 135 度，1turn 表示旋转一周

7. 如何让 loading 的大小随着不同的页面而有不同的呈现

使用 rem 和 em

8. 如何让一个 button 内的 span 随着父元素的颜色的变化而变化

使用 inherit 来继承父元素的 color 属性

inherit 可以继承的属性有：

color，font-size，font-weight，text-align, line-height,...

9. 介绍一下 websocket

10.在 websocket 监听页面时，如果防止脚本冒充用户？

回答：设置滚动条长时间不滚动的话，就计为无效

11.那如果是一个有经验的前端，让滚动条滚动了呢？

无解...

12.一道笔试题：

让字符串中的元素向左移动 n 位

```
function str(str, n){
    if(n > str.length){
        n = n % str.length;
    }
    let firstStr = str.substring(0, n);
    let secondStr = str.substring(n);
    str = secondStr + firstStr;
}
```

8.2 面试复盘

注意考虑边界条件

9. 2022 年 3 月 11 日 美团一面

9.1 面试问题汇总

1. src 和 href 的区别，在浏览器加载的时候有什么区别

首先 src 和 href 都是用于外部资源的引入。

href 指的是超文本引用。

href 这个属性指定 web 资源或者 css 资源文件的位置，从而建立当前元素（如锚点 a）或当前文档（如链接）与目标锚点或目标资源之间的链接。因此当浏览器在 html 文件时，如果遇到了 href 这个属性，html 页面的解析和渲染不会暂停，css 文件的加载是同时进行的。

而 **src 是 source（源）的简写，这个属性是将外部资源嵌入到当前文档中元素所在的位置**。因此当浏览器解析到了这句代码时，HTML 页面的加载和解析都会暂停，直到浏览器加载、编译、执行完毕这个 js 文件或者加载完 image 的 src 指向的那个图片资源后，才会继续解析 HTML 文件的内容。

src 一般引入 js 文件和图片文件。

2. defer 与 async

script 标签用于加载脚本与执行脚本，是前端开发中非常重要的标签。

直接使用 script 脚本的话，html 会按照顺序来加载并执行脚本，在脚本加载及执行过程中，会阻塞后续的 DOM 渲染。

加载和执行要求的。

3、对于 async 而言，脚本的加载和执行是紧紧挨着的，不管声明顺序如何，只要加载完就会立即执行。因此 async 对于应用脚本的用处不大，因为它完全不考虑依赖（哪怕是最低级的顺序执行），不过它对于那些可以不依赖任何脚本或者不被任何脚本依赖的脚本来说是非常合适的。比如 ajax 请求，百度统计、Google Analytics。

4、对于 defer 而言，它会等待所有元素解析完成之后，在 DOMContentLoaded 事件触发之前完成，并不是在当前 script 文件加载和解析完成后就立即执行。因此如果我们的 js 脚本代码依赖于页面中的 DOM 元素，或者被其他 js 文件所依赖的话，还是用 defer 更稳妥一些。比如：评论框、代码语法高亮、polyfill.js

3. 原生 js 获取页面对象的方法

getElementById

getElementByName

getElementByClassName

getElementByTagName

querySelector

querySelectorAll

document.documentElement 用于获取 html 这个标签

document.body 用于获取 body 这个标签

4. 前端存储

cookie，webstorage：localStorage 和 sessionStorage，IndexDB

cookie 常用于和服务器端进行通信，作为 HTTP 规范的一部分而存在。

而 **webStorage** 就是用来做前端本地存储的，

localStorage 和 sessionStorage 的大小都是 5M。且不支持跨域。如果需要多次写入 localstorage，尽量一次性写入。localstorage 是同步执行，可能会阻塞 UI

而且 sessionStorage 的跨域限制比 localStorage 更严格，即使是两个相同的 tab 页，也算是跨域。

websql 像关系型数据库，使用 sql 语句进行操作，**indexdb** 像 nosql，直接使用 js 方法操作数据。

访问：indexdb 和尾部 storage 一致，均是在创建数据库的域名下才能访问，且不能指定访问域名。

存储时间：indexdb 的存储时间为永久，除非用户清除数据，可以用作长效的存储。

大小限制：基本上没有存储限制，但是 indexDB 的数据库超过 50M 的时候浏览器会弹出确认。

性能测试：indexdb 查询少量数据花费差不多 20ms 左右，大量数据的情况下，相对耗时会变长一些，但是也就 30MS 左右，可以说是非常给力了。

IndexedDB 具有以下特点。

key/value 的存储方式：IndexedDB 和 localStorage 的存储方式很类似，都是通过一个 key 对应一个 value，而且 key 是唯一的方式进行存储的，但是 indexedDB 和 localStorage 有很不一样的一点，就是可以直接存储对象数组等，不需要想 localStorage 那样必须转为字符串。

异步调用：**IndexedDB 是使用异步调用的，当我们存储一个较大的数据时，不会因为写入数据慢而导致页面阻塞。**

支持事务：IndexedDB 支持事务，如果有用过 mysql 和 mongoDB 的人就很清楚了，能**确保我们多个操作只要其中一步出现问题，可以整体回滚。**

同源限制：IndexedDB 和 localStorage 一样，都是有同源策略的问题，不能跨协议、端口、域名使用。

储存空间：**IndexedDB 我认为最最最大的优势在于存储空间相比 localStorage 要大得多，一般来说不少于 250MB。**

支持二进制：IndexedDB 不但可以存储对象，字符串等，还可以存储二进制数据。

什么场景下使用:

在 PC 的 Chorme 中是可以存到 localStorage 的，但是在 IOS 中，却报出空间不足，无法放入 localStorage 中，所以这个时候就是使用 indexedDB 了！因为 indexedDB 的空间大得我可以完全不去考虑数据大小，而且还能直接以对象的形式存入，无需转为 JSON 字符串。大大

减少了转换的运算。但是因为使用 indexedDB 和使用 localStorage 是完全不一样的，基本上都是异步操作而且还要考虑一些低版本的手机可能不支持的情况，所以要封装中间件，同样的调用，根据设备对 indexedDB 的兼容情况，自动决定使用 indexedDB 还是 localStorage。最终完成需求，并且优化前后达到超过 70%的优化率，页面的渲染基本秒开。

大部分主流程的浏览器其实都已经兼容了 indexedDB

几种前端数据存储方式

5. localStorage 和 sessionStorage 的有效时间

localStorage 存储的数据没有时间限制，永久存储，永不失效，除非手动删除。

localStorage 的检测方法：

```
if(window.localStorage){
 alert('This browser supports localStorage');
}else{
 alert('This browser does NOT support localStorage');
}
```

常用 API：

setItem

getItem

clear

key

removeItem

存储的内容：

只要可以序列化为字符串的内容都可以存储。包括：数组、图片、json、样式、脚本等

sessionStorage 和 localStorage 的用法一致，不过 sessionStorage 会在页面关闭后被清空。

6. canvas 和 svg 的区别

svg 绘制出来的每一个图形的元素都是独立的 DOM 节点，能够方便的绑定事件或用来修改。canvas 输出的是一整幅画布。

svg 输出的图形是矢量图形，后期可以修改参数来自由放大缩小，不会是真和锯齿。而

canvas 输出标量画布，就像一张图片一样，放大会失真或者锯齿

7. js 的基本类型

es6 之前，js 的基本类型有 string，number，boolean, null，undefined,引用类型是 object

es6 之后，js 的基本类型又新增了一个 symbol

8. symbol 的作用

symbol 作为 js 的基本数据类型之一，它与其他的基本类型的不同之处在于，其他的基本类型在值相等的情况下，在强等于的比较中仍然是 true。

但是 **symbols 是一种无法被改变的基本类型数据，它的值一旦被创建后，就不能被修改了**。

因此两个 symbols 类型的对象在强等于的情况下，返回值也是 false。

这有点类似于对象创建的实例互相不相等的情况。

```
const s1 = Symbol();
const s2 = Symbol();
console.log(s1 === s2); // false
```

```
const s1 = Symbol('debug');
const s2 = Symbol('debug');
let s3 = 'debug';
console.log(s1 === s3);
console.log(s1 === s2);
console.log(s1);
```

```
false
```

```
false
```

```
Symbol(debug)
```

（2）symbol 的另一个重要的用途是用作对象的 key

如果我们使用 for .. in 和 foreach 等迭代的方式，是拿不到键为 symbol 的属性的，这是为了兼容老版的浏览器（即向后兼容性），老版本兼容 symbols，因此 Object.keys()不能返回 symbols。

不过可以通过 Reflect.ownKeys()方法拿到包含 Symbols 在内的所有 key.

Object.getOwnPropertySymbols()方法也可以拿到 Symbols 属性。这个方法如果打印为空，则代表浏览器暂时还不支持这个方法。

```
const obj = {};
obj[Symbol()] = '橙子';
obj[Symbol()] = '柚子';
console.log(obj);
```

Reject.ownKeys(obj)只能获取到属性值为 Symbol 的键，但是却拿不到值，那如何才能拿到值呢：就是下列代码中的 console.log(sym['blue']);

然后把拿到的值 toString()就可以了。

```
let s1 = Symbol('hello');
let obj = {
    [s1]: 'foo bar'
};
console.log(obj);
Object.defineProperty(obj, s1, {
    value: 'world'
})
console.log(obj);
obj[s1] = 'hello'
console.log(obj);
```

可以直接通过 Obj[s1]修改，也可以通过 Object.defineProperty 修改

Symbol 的作用

9. object.defineProperty()方法可以修改 symbol 标签吗？

~~不能~~。可以

读下这篇文章：https://blog.csdn.net/yun_shuo/article/details/116242091

10. 如何判断数据类型

（1） typeof

（2） instanceof，用于判断引用数据类型的对象具体属于哪个类型

（3） toString()方法

toString()是 Object 的原型方法，调用该方法，默认返回当前对象的[[class]],这是一个内部属性，其格式为[object xxx],其中 XXX 就是对象的类型。

对于 Object 对象，直接调用 toString()就能返回[object Object]。而对于其他对象，则需要通过 call/apply 来调用才能返回正确的类型信息。

```
> var f = new F()
< undefined
> f.constructor == F
< true
```

可以看出，**F 利用原型对象上的 constructor 引用了自身**，当 F 作为构造函数来创建对象时，原型上的 constructor 就被遗传到了新创建的对象上，**从原型链角度讲，构造函数 F 就是新对象的类型。这样做的意义是，让新对象在诞生以后，就具有可追溯的数据类型**。

同样，js 中的内置对象在内部构建时也是这样做的：

```
> ''.constructor == String
< true
> new Number(1).constructor == Number
< true
> true.constructor == Boolean
< true
> new Function().constructor == Function
< true
> new Date().constructor == Date
< true
> new Error().constructor == Error
< true
> [].constructor == Array
< true
> document.constructor == HTMLDocument
< true
> window.constructor == Window
< true
>
```

可以看出，无论是 new Number(1)，还是 new Function 或者 new Date()等，都是 new 出了一个 new 后面跟着的那个类型的具体对象，因此我们再使用

new Function().constructor == Function 就是 true。

刚才测试了一下，即使是强等于情况下，返回结果也是 true。

```
new Function().constructor ==Function;
true
new Function().constructor === Function;
true
```

11. null 和 undefined

undefined 是已存在但未赋值的属性，变量和属性

null 是不存在这个属性或者变量

12. Object.assign 平常怎么使用

Object.assign() 方法用于将所有可枚举属性的值从一个或多个源对象分配到目标对象。它将返回目标对象.

```
const target = { a: 1, b: 2 };
const source = { b: 4, c: 5 };

const returnedTarget = Object.assign(target, source);

console.log(target);
// expected output: Object { a: 1, b: 4, c: 5 }

console.log(returnedTarget);
// expected output: Object { a: 1, b: 4, c: 5 }
```

语法：

```
Object.assign(target, ...sources)// 这里的...sources 表示可以有多个源对象
```

参数：

target 是目标对象

sources 是源对象

返回值：

返回值也是目标对象

很重要的一点是：

如果目标对象和源对象之间有属性重复的话，则目标对象中的相同源属性会被覆盖。

后面的源对象的属性将类似地覆盖前面的源对象的属性。

Object.assign 方法只会拷贝源对象自身的并且可枚举的属性到目标对象。该方法使用源对象的[[Get]]和目标对象的[[Set]]，所以它会调用相关 getter 和 setter。因此，它分配属性，而不仅仅是复制或定义新的属性。如果合并源包含 getter，这可能使其不适合将新属性合并到原型中。为了将属性定义（包括其可枚举性）复制到原型，应使用 Object.getOwnPropertyDescriptor()和 **Object.defineProperty()** 。

String 类型和 Symbol 类型的属性都会被拷贝。

在出现错误的情况下，例如，如果属性不可写，会引发 TypeError，如果在引发错误之前添加

```
const obj = { a: 1 };
const copy = Object.assign({}, obj);
console.log(copy); // { a: 1 }
```

深拷贝问题：

针对深拷贝，需要使用其他办法，因为 Object.assign()拷贝的是（可枚举）属性值。**假如源值是一个对象的引用，它仅仅会复制其引用值。**

```
const log = console.log;

function test() {
  'use strict';
  let obj1 = { a: 0 , b: { c: 0}};
  let obj2 = Object.assign({}, obj1);
  log(JSON.stringify(obj2));
  // { a: 0, b: { c: 0}}

  obj1.a = 1;
  log(JSON.stringify(obj1));
  // { a: 1, b: { c: 0}}
  log(JSON.stringify(obj2));
  // { a: 0, b: { c: 0}}

  obj2.a = 2;
  log(JSON.stringify(obj1));
  // { a: 1, b: { c: 0}}
  log(JSON.stringify(obj2));
  // { a: 2, b: { c: 0}}

  obj2.b.c = 3;
  log(JSON.stringify(obj1));
  // { a: 1, b: { c: 3}}
  log(JSON.stringify(obj2));
  // { a: 2, b: { c: 3}}

  // Deep Clone
  obj1 = { a: 0 , b: { c: 0}};
  let obj3 = JSON.parse(JSON.stringify(obj1));
  obj1.a = 4;
  obj1.b.c = 4;
  log(JSON.stringify(obj3));
  // { a: 0, b: { c: 0}}
}

test();
```

合并对象：

```
const o1 = { a: 1 };
const o2 = { b: 2 };
const o3 = { c: 3 };

const obj = Object.assign(o1, o2, o3);
console.log(obj); // { a: 1, b: 2, c: 3 }
console.log(o1);  // { a: 1, b: 2, c: 3 }, 注意目标对象自身也会改变。
```

合并具有相同属性的对象

```
const o1 = { a: 1, b: 1, c: 1 };
const o2 = { b: 2, c: 2 };
const o3 = { c: 3 };

const obj = Object.assign({}, o1, o2, o3);
console.log(obj); // { a: 1, b: 2, c: 3 }
```

拷贝 symbol 属性的值

```
const o1 = { a: 1 };
const o2 = { [Symbol('foo')]: 2 };

const obj = Object.assign({}, o1, o2);
console.log(obj); // { a : 1, [Symbol("foo")]: 2 } (cf. bug
1207182 on Firefox)
Object.getOwnPropertySymbols(obj); // [Symbol(foo)]
```

继承属性和不可枚举的属性是不能被拷贝的：

```
const obj = Object.create({foo: 1}, { // foo 是个继承属性。
    bar: {
        value: 2  // bar 是个不可枚举属性。
    },
    baz: {
        value: 3,
        enumerable: true  // baz 是个自身可枚举属性。
    }
});

const copy = Object.assign({}, obj);
console.log(copy); // { baz: 3 }
```

原始类型的值会被包装为对象：

```
const v1 = "abc";
const v2 = true;
const v3 = 10;
const v4 = Symbol("foo")

const obj = Object.assign({}, v1, null, v2, undefined, v3, v4);
// 原始类型会被包装，null 和 undefined 会被忽略。
// 注意，只有字符串的包装对象才可能有自身可枚举属性。
console.log(obj); // { "0": "a", "1": "b", "2": "c" }
```

详细请看 MDN

13. Object.assign 是深拷贝还是浅拷贝

浅拷贝

14. 箭头函数和普通函数有什么区别？

箭头函数的优点是简单明了，适用于替代比较简单的函数。

箭头函数不能用来创建构造函数的实例。因此不能用 new 来创建构造函数原型。

箭头函数没有自身 arguments 对象。

箭头函数的 this 值取决于普通函数的 this 值。

16. ES6 中的 proxy 有了解吗

```
let handler = {
    get: function(target, name){
        if(target.hasOwnProperty(name)){
            return target[name];
        }else{
            console.warn('没有找到你要的属性');
            return;
        }
    },
    set: function(target, name, value){
        target[name] = '#' + value;
    },
    deleteProperty: function(target, name){
        delete target[name];
    },
    defineproperty: function(
       // 该方法用于拦截 Object.defineProperty()操作
       // 最后的返回结果是 Boolean
    }
}

let p = new Proxy({}, handler);// p 是 Proxy 的第一个参数{}的代理，handler，是对代理的拦截处理,set 是赋值前的拦截处理，get 时读取值之后的拦截处理。

p.a = 1;
p.abc = 'abc';
console.log(p.a);
```

总结下来，我认为 proxy 它的根本作用就是先拷贝源对象，然后之后在这个拷贝后的 proxy 对象上，通过 handler 拦截器做一些设置值之前和读取值之前的拦截操作。太神奇了!

proxy 作为 ES6 的新特性，它的缺点是不支持向后兼容。我们可以使用代理来做很多事情，包括跟踪属性访问、隐藏属性、阻止修改或者删除属性、函数参数验证、构造函数参数验证、数据绑定以及可观察对象等等，非常神奇

17. map 和 weakmap

js 中 map 的出现，就是为了解决 js 中的对象的键只能为字符串的这个缺点。

但是 map 的缺点就是耗费内存，强引用。

因此在 es6 版本中，提出了 weakMap 的概念。

那么什么是强引用，什么是弱引用呢？

就是创建引用之后，无法被垃圾资源回收机制进行回收的，就是强引用。强到即使你设置了 null，也无法回收。

而垃圾资源回收的意义就在于尽可能的避免内存泄露，所谓内存泄露，就是被没用的应用长时间占着内存，造成内存使用率虚高。

map 与 weakmap 的区别：

ES6 考虑到了这一点，推出了：WeakMap 。它对于值的引用都是不计入垃圾回收机制的，所以名字里面才会有一个"Weak"，表示这是弱引用（对对象的弱引用是指当该对象应该被 GC 回收时不会阻止 GC 的回收行为）。

（1）强/弱引用的区别：Map 强，WeakMap 弱

（2）WeakMap 只接受对象为 key 值

在实现完美的深拷贝中，我们使用 WeakMap 代替 Map 的使用来解决循环引用的问题，进行优化。

Map 相对于 WeakMap ：

（1）Map 的键可以是任意类型，**WeakMap 只接受对象作为键**（null 除外），不接受其他类型的值作为键

（2）Map 的键实际上是跟内存地址绑定的，只要内存地址不一样，就视为两个键；

WeakMap 的键是弱引用，键所指向的对象可以被垃圾回收，此时键是无效的

（3）**Map 可以被遍历， WeakMap 不能被遍历**

只要外部的引用消失，WeakMap 内部的引用，就会自动被垃圾回收清除。由此可见，有了它的帮助，解决内存泄漏就会简单很多。

因此我们可以发现，weakMap 的使用，是有很多的限制的，那什么场景下适合使用 weakMap 呢？就是用来保存 DOM 节点

这篇文档讲的特别好：浅析 Map 和 WeakMap 有什么不同

18. arguments 是什么类型

19. DOM 和 BOM 之间有什么关系

https://www.cnblogs.com/liweibin00/p/11720563.html

20. 封装一个组件如何考虑浏览器的兼容性

通过学习，我发现封装一个组件基本上最后都是以一个 js 的文件暴露出去。也就是说我们的页面元素最终都是通过 js 插入在页面上的。除此之外，我们还可以单独写一个 css 文件来规定样式。

下面这个组件封装的过程值得我们学习：

前端如何封装一个组件

21. 封装一个组件的设计思路-以 axios 为例

22. 讲讲 promise，如何把 ajax 改造成 promise

23. react 事件对象的模型的原理是什么

e.preventDefault()可以阻止事件的默认行为

e.stopProppagation()可以阻止事件传递，比如一个 div 里有个 button 标签，我们给 div 和 button 都绑定了 onclick 事件，那么里面的 button 执行了 onclick 事件后，就会向上冒泡执行 div 的 onclick 事件，这个是非常不提倡的，因此我们常采用 addEventListener 来为元素绑定点击事件。

首先明确几个概念：

①我们在 jsx 中绑定的事件 (demo 中的 handerClick，handerChange), 根本就没有注册到真实的 dom 上。是绑定在 document 上统一管理的。

②真实的 dom 上的 click 事件被单独处理, 已经被 react 底层替换成空函数。

③我们在 react 绑定的事件, 比如 onChange，在 document 上，可能有多个事件与之对应。

④ react 并不是一开始，把所有的事件都绑定在 document 上，而是采取了一种按需绑定，

比如发现了 onClick 事件, 再去绑定 document click 事件。

那么什么是 React 的事件合成呢？

在 react 中，我们绑定的事件 onClick 等，并不是原生事件，而是由原生事件合成的 React 事件，比如 click 事件合成为 onClick 事件。比如 blur，change，input，keydown，keyup 等，合成为 onChange。

那么 react 采取这种事件合成的模式呢？

一方面，将事件绑定在 document 统一管理，防止很多事件直接绑定在原生的 dom 元素上。造成一些不可控的情况

另一方面， React 想实现一个全浏览器的框架， 为了实现这种目标就需要提供全浏览器一致性的事件系统，以此抹平不同浏览器的差异。

react 事件合成总结：

这个阶段主要形成了几个重要对象，**构建初始化 React 合成事件和原生事件的对应关系，合成事件和对应的事件处理插件关系。**接下来就是事件绑定阶段。

react 事件绑定流程总结：

1 diffProperties 处理 React 合成事件

2 legacyListenToEvent 注册事件监听器

3 绑定 dispatchEvent，进行事件监听

接下来讲讲，在 react 中执行了一个 onclick 事件后，react 进行了哪些处理：

（1） 首先执行的是 dispatchEvent 函数

（2） legacy 事件处理系统与批量更新

（3） 执行事件插件函数

（4） extractEvents 形成事件对象 event 和 事件处理函数队列

总结起来，react 执行事件触发的流程是：

①首先通过统一的事件处理函数 dispatchEvent, 进行批量更新 batchUpdate。

②然后执行事件对应的处理插件中的 extractEvents，合成事件源对象, 每次 React 会从事件源开始，从上遍历类型为 hostComponent 即 dom 类型的 fiber, 判断 props 中是否有当前事件比如 onClick, 最终形成一个事件执行队列，React 就是用这个队列，来模拟事件捕获 -> 事件源 -> 事件冒泡这一过程。

③最后通过 runEventsInBatch 执行事件队列，如果发现阻止冒泡，那么 break 跳出循环，最后重置事件源，放回到事件池中，完成整个流程。

【react 进阶】一文吃透 react 事件触发

24. vue 在移动端用的多吗

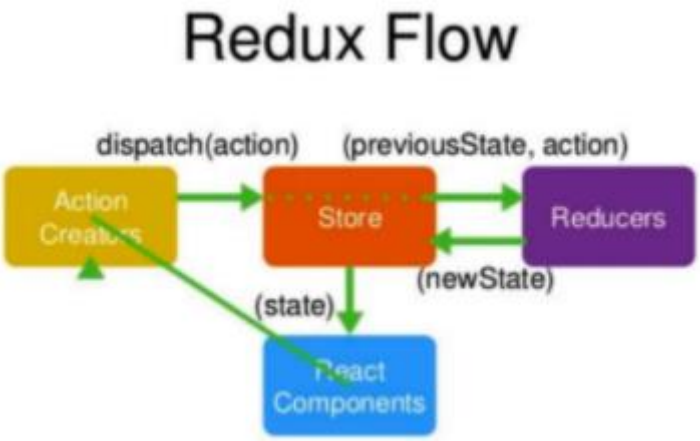
25. vue 中的 compute 和 watch 你用过吗

26. vue 和 react 的异同点

27. react 在嵌套层级很深的组件上修改一个页面级元素，如何影响同级的其他页面元素

28. 页面组件如何使用 react-redux 进行数据通信？

29. react-redux 的核心图



把 redux 工作流程必做借书过程的话，react 组件是“借书的人”，store 是“图书管理

员”，action 是“借书的人和图书管理员说的话”，reducer 是“图书管理员的查书手册”。当借书者（react 组件）想要借一本书时，会和图书管理员（store）说一句话（action），这句话有固定格式，必须是对象或者函数，包含 type 和 value,图书管理员（store）听到话后回去查手册（reducer），手册（reducer）会根据之前的数据（state)和这句话（action)，告诉图书管理员（store）现在的数据是什么，管理员（store）拿到现在的数据替换老数据

30.vue 的双向绑定原理？

10. 2022 年 3 月 14 日 字节三面

10.1 面试问题汇总

1. **之前的面试体验如何**
2. **介绍一下自己最有挑战，最有收获的项目**
3. **项目的技术栈选型有什么考虑**
4. **vue 和 react 有什么区别**
5. **数据流框架在使用上有什么考虑**
6. **react 框架中经常使用的 hooks 有哪些？**
7. **useMemo 和 useCallback 在使用场景上有什么区别？**

useMemo 和 useCallback 都是性能优化的手段，类似于类组件中的 shouldComponentUpdate，是在组件渲染期间执行的钩子函数，用于判定该组件的 props 和 state 是否有变化，从而避免每次父组件 render 时都去重新渲染子组件。

useMemo 和 useCallback 都干的是缓存的事，

useCallback 是 useMemo 的语法糖

useMemo 返回的是一个值，用于避免在每次渲染时都进行高开销的计算。

useCallback 返回的是一个函数，当把它返回的这个函数作为子组件使用时，可以避免每次父组件更新是都重新渲染这个子组件。

那既然都有 useEffect 了，为什么还需要 useMemo 和 useCallback 呢？

其实 useEffect 它的执行时期是 DOM 渲染完成之后。

不过 useMemo 和 useCallback 也有缺点，那就是需要一直维护第二个参数数组，也就是依赖数组。并且保留旧值的副本，以便于当子组件的值没有更新时直接返回这个旧值的副本。

https://zhuanlan.zhihu.com/p/445048107

https://jishuin.proginn.com/p/763bfbd4ed55

8. useEffect 如何做一个 already 的效果？

9. 节流和防抖有什么区别？

10.我看你的功能有跟微信做相关的操作，具体是实现了什么功能？

11.什么样的请求算作是跨域？

12.如何解决跨域？

13.nginx 里需要配置请求头里的哪些参数？

```
   location /{
    add_header  'Access-Control-Allow-Origin'  '*' ;
    add_header  'Access-Control-Allow-Credentials'  'true' ;
    add_header  'Access-Control-Allow-Methods'  'GET, POST, OPTIONS' ;
    add_header  'Access-Control-Allow-Headers'  'DNT,web-token,app-
token,Authorization,Accept,Origin,Keep-Alive,User-Agent,X-Mx-ReqToken,X-Data-Type,X-
Auth-Token,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Range' ;
    add_header  'Access-Control-Expose-Headers'  'Content-Length,Content-Range' ;
    # 此处8091是我们本地运行项目的端口号，设置成跟你本地起的服务端口号一样就行
```

```
server {
  listen 80;
  server_name cache.lion.club;
  location / {
    proxy_cache cache_zone; # 设置缓存内存，上面配置中已经定义好的
    proxy_cache_valid 200 5m; # 缓存状态为200的请求，缓存时长为5分钟
    proxy_cache_key $request_uri; # 缓存文件的key为请求的URI
    add_header Nginx-Cache-Status $upstream_cache_status # 把缓存状态设置为头部信息，响应给客户端
    proxy_pass http://cache_server; # 代理转发
  }
}
```

14.讲讲 cors 跨域请求的简单请求和非简单请求？

15.出于什么初衷去写博客和开源项目的？

16.字节很喜欢有产品思维&有技术能力的同学

10.2 面试复盘

应该去了解一下 react 中的常用 hook：

当然不止文档里的这几种：React 常见 Hooks

回答项目中功能实现的点时的思路是：

1. 为什么要做这个
2. 怎么做的
3. 达到了什么效果

11. 2022 年 3 月 17 日 字节 HR 面

1. **前面的面试体验怎么样**
2. **为什么想做前端**
3. **怎么学习前端的**
4. **讲一下你是怎么参与到这个项目中的**
5. **谈谈你在这个项目中的收获**
6. **了解面试的团队吗**
7. **还面了哪些公司**
8. **能够实习多久**

12. 2022 年 3 月 17 日 美团二面

1. 介绍一下自己

2. 介绍一下阿里训练营的课程讲了些什么，学到了什么

3. 介绍一下百度训练营的课程讲了些什么，做了什么项目

4. 为什么你们的项目在百度训练营里能拿第一

5. 这半年你在干什么

6. 介绍下 react 框架

7. 介绍下 react 框架的缺点

8. react 框架除了上手慢，对新手不友好，其余还有啥缺点

9. 介绍下项目

10.调用微信 JDK 的过程

前端这块需要引入微信的 JDK 作为使用微信提供的 API 的基础条件。除此之外，使用微信的 API 需要进行鉴权，即提供 APPID、APPSecret 以及最重要的微信要求的签名 signature。有些这些我们才可以在微信的容器里调用分享，拍照，获取地理位置等权限。考虑到安全性问题，生成签名这个需要服务端来完成，因此调用微信 JDK 的过程就需要前后端来配合。

因为在项目没上线之前，前后端联调微信 JDK 的过程比较复杂，如果前后端两个人来做的话会有很多外部因素。然后我自己又懂一些后台，因此我就自己使用 node.js 针对微信调用这个功能写了后台的代码来和前端配合。

整个过程就是，前端把想要分享的页面 url 发送给后台，后台拿到这个 url 后，首先向微信服务器通过 appid 和 APPsecret 来获取 access_token，作为后续申请签名的凭证，然后拿着这个 access_token 去获取 js_token，有了 js_token 后，就可以按照微信要求的算法使用 js_token、随机数、时间戳和 url 生成前端需要调用 JDK 的签名，然后返回这个生成的签名给前端即可。

当前端收到这个签名后，再拿着 AppID 就可以正常调用微信的 JDK 了。

11.你们的产品数据怎么样

12.建议上线后关注下数据量

13.数据统计模块是干嘛的

14.TCP 和 UDP 的区别

15.UDP 的使用场景

16.我看你的研究方向是 xxx，讲一下 xxx 是干什么的？

17.为啥不做科研方向呢？

2022 年 3 月 24 日 读了别人的春招准备之路的进一步感悟

1. 感觉自己的 JS 基础还很不牢靠，包括浏览器、计网、操作系统还有 JS 基础
2. react 和 vue 欠缺的知识还很多，包括源码解读，需要再好好了解
3. 前端前沿知识要适当的了解，包括 ES6 最新的语法有哪些，什么是微前端，什么是 severless,设计一个组件时要怎么考虑兼容性问题

个人理解，从前端角度来说，serverless 使得前端工程师可以成为全栈工程师。像小程序中的云函数，云开发，我认为就是 serverless 的一个体现。它使得前端工程师可以自己编写后端的接口，并且不用考虑后端采用什么样的框架，直接用 js 来写业务接口即可，对前端工程师非常友好~当然这只是从前端开发者的角度来局部理解的。

2022 年 7 月 6 日 中电十所一面

1. 你的浏览器插件最后的保存格式是什么样的？
2. 浏览器里输入 URL 后发生了什么？
3. http 请求属于什么协议
4. 服务端如何渲染页面-jsp 页面
5. get 请求和 post 请求的区别
6. 前端如何获取页面上的元素
7. 写过后端吗，用过什么后端语言
8. 了解过数据库嘛

2022 年 7 月 18 日 字节终面

1. 使用 electron 做开发和使用 web 技术做开发的区别点在哪里？
2. 在使用 electron 做开发时，有涉及到跟系统底层相关的相互吗？比如展示菜单，最小化到系统托盘等？
3. electron 偏底层的知识有了解到吗？比如 IPC 还有通信这些
4. electron 是如何调用系统底层的 api 的，即浏览器调用 node 方法之间是有一层通信在的？这个过程是什么？
5. 在项目中你们有自己重新实现菜单栏工具栏的 UI 吗？这个要如何实现？
6. electron 还有清理磁盘垃圾、操作打印机等功能
7. useMemo 和 useCallback 的使用场景
8. react 中的 provider 实现原理
9. react 和 vue 如何实现语言国际化

10.js 中 Number 的上限是多少？2^53 - 1

11.Number.MaxValue 和 Number.MaxSafeInteger 有什么区别？

12. 算法题：超大数相加

我自己的版本：

```
function add(a, b){
    let arrA = a.split('').reverse();
    let arrB = b.split('').reverse();
    let ret = '';
    let minLength = Math.min(arrA.length, arrB.length);
    let needAdd = false;
    for(let i = 0; i < minLength; i++){
        let sum = parseInt(arrA[i]) + parseInt(arrB[i]);
        if(needAdd === true){
            sum += 1;
        }

        if(sum >= 10 ){
            sum = sum%10;
            needAdd = true;
        }else{
            needAdd = false;
        }
        ret += sum;
    }

    ret = ret.split('').reverse().join('');
    if(arrA.length > arrB.length){
        let remain = arrA.slice(arrA.length - minLength);
        let remainStr = parseInt(remain.reverse().join(''));
        if(needAdd = true){
            remainStr += 1;
        }
        ret = remainStr + ret;
    }else if(arrB.length > arrA.length){
        let remain = arrB.slice(arrB.length - minLength);
        let remainStr = parseInt(remain.reverse().join(''));
        if(needAdd = true){
            remainStr += 1;
        }
        ret = remainStr + ret;
    }else{
        if(needAdd === true){
            ret = 1 + ret;
        }
    }
    return ret;
}
```

最优雅的版本：

```
function add01(a, b){
    let arrA = a.split('').reverse();
    let arrB = b.split('').reverse();
    let ret = [];
    let minLength = Math.min(arrA.length, arrB.length);
    let needAdd = false;
    for(let i = 0; i < minLength; i++){
        let sum = parseInt(arrA[i]) + parseInt(arrB[i]);
        if(needAdd === true){
            sum += 1;
        }

        if(sum >= 10 ){
            sum = sum%10;
            needAdd = true;
        }else{
            needAdd = false;
        }
        ret.push(sum);
    }

    ret = ret.reverse().join('');

    let remain;

// 最优雅的版本
function addBig(a, b){
    let temp = 0,
        res = '';
    // 转成数组
    a = a.split('');
    b = b.split('');
    while( a.length || b.length || temp){
        //首先使用~操作符进行取反是会将字符串转换为数字的，在进行一个取反操作，就转换为了字符串本身代表的数字，
        // 但不仅如此。考虑下我们数组循环的最后为空时，[].pop()的结果为undefined， 而 Number(undefined)的结果为NaN，~NaN 的结果为-1， ~~NaN 的结果 为0，
        // 当你使用 NaN 进行加法运算时，一切结果都没有了意义。所以这个地方必须为~~a.pop()， 而不能使用 Number 进行强制转换

        temp += ~~a.pop() + ~~b.pop();
```

拓展一下，两数相乘怎么做？

比如：23 * 49

思路：拿第二个数的每一位和第一个数相乘，把结果存入数组中，然后数组的元素依次做加法，最后得到的就是两数相乘的结果。

2022 年 8 月 4 日 用友一面

1. H5 与原生 App 之间如何进行通信

首先，我们的 H5 页面在 App 中是通过 webview 来呈现界面的。

但是 H5 页面中的 JS 代码是不具备调用原生 APP 才拥有的功能的。

如果我们想要调用，就必须采用 JSBridge 来和 APP 进行通信，让 APP 去发起我们要完成的操作，然后然后结果给 H5。

总结下来，就是需要 APP 端和 H5 的共同配合：

首先是 APP 端：

- 我们需要使用 addJavascriptInterface 来声明 JS 端与 APP 端通信时的暗号。便于后续 H5 向 APP 端发送信号时有方法可以接收。

```
wv.addJavascriptInterface(myJavaScriptInterface,"AndroidFunction"
);
```

- 使用 loadUrl 方法去加载 H5 页面，APP 端向 H5 端传递数据就是通过在 URL 里进行拼接

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        final JavaScriptInterface myJavaScriptInterface = new
JavaScriptInterface(this);

        //方式二：Webview 控件跳转网页
        WebView wv = findViewById(R.id.wv);

wv.addJavascriptInterface(myJavaScriptInterface,"AndroidFunction"
);

wv.loadUrl("http://uniapp.htt5.com/test.html?name=htt");// 将参数
拼接在 url 里即可
    }

    // 用于接收 H5 给安卓传递的参数
    public class JavaScriptInterface {
        Context mContext;

        JavaScriptInterface(Context c) {
            mContext = c;
        }

        @JavascriptInterface
        public void onShare(String webMessage) {
            System.out.println(webMessage);
        }
    }
```

然后是 H5 端：

```
<body>
    <a href="tel:18428338069">400-0000-688</a>
    <a href="sms:18688888888">发短信</a>
    <p id="name"></p>
    <button onclick="toAPP()">app交互</button>
    <script>
        var name = window.location.href;
        var p = document.getElementById('name');
        p.innerHTML = name;

        function toAPP() {
            alert("hello");
            AndroidFunction.onShare('sweet');
        }
    </script>
</body>
```

2. 移动端适配的方法

现在主流的解决方案就是把 px 转换为 vw 的单位。

比如说，我们的设计稿是 750px，然后我们的字体大小是 15px，那么对应到 vw 中，因为 vw 只有 100 等份嘛，然后 100vw 就等于设计稿的 750px。

那么 15px 对应到 vw 里，应该就是 15px / 750px * 100vw

所以 px2vw 的转换公式为：

```
@function rpx($size){
  @return #{$size / 设计稿的宽度 * 100}vw
}
```

那个 postcss-px-px-to-viewport 插件的核心原理是这个，

只不过在我们的开发过程中，直接使用 px 的单位就可以了，这个插件会在 webpack 打包时自动帮我们执行这个转换操作。

```
module.exports = {
  plugins: {
    ...
    'postcss-px-to-viewport': {
      unitToConvert: 'px',
      viewportWidth: 320,
      unitPrecision: 5,
      propList: ['*'],
      viewportUnit: 'vw',
      fontViewportUnit: 'vw',
      selectorBlackList: [],
      minPixelValue: 1,
      mediaQuery: false,
      replace: true,
      exclude: [],
      landscape: false,
      landscapeUnit: 'vw',
      landscapeWidth: 568

    }
  }
}
```

设计稿的宽度

官网： https://evrone.com/postcss-px-viewport

2022 年 8 月 12 日 用友二面

1. 自我介绍
2. 先问了自己做的浏览器插件具体功能，然后说用户是否可以动态选择打印区域？回答说没有
3. 然后就出了一个场景题：如何动态生成可打印的区域供用户选择

   最终的结论是：先把选择权交给用户，让用户来选择想打印的区域，然后收集用户打印区域的信息，供以后其他用户打印该页面时使用。（现实中我可不会这样干..真麻烦）

9. 介绍一个自己最热门的博客-js 的四种异步调用的方式
10. promise 上有什么方法可以等所有 promise 执行完之后再执行后续操作
11. promise 的实现原理是什么
12. 假设没有 promise，怎么通过 es5 来实现 promise
13. 网络请求库 axios 是如何解决兼容性问题的？比如说有些低版本浏览器不支持 ES6 的 promise

    回答：使用回调函数呗
14. 介绍一下百度训练营的这个项目，自己承担的什么角色，遇到了什么问题，如何解决的
15. 之前有没有接触过用友这边的业务？
16. 介绍了一下部门的情况
17. 是否可以提前来实习

2022 年 8 月 8 日 百度一面

2022 年 8 月 11 日 百度二面

1. 自我介绍
2. 介绍训练营相关的内容
3. 介绍在微软的项目
4. 不同终端中邮件渲染方式有什么不同
5. 浏览器缓存的策略有哪些
6. 强缓存与协商缓存有什么区别？
7. HTTP-ONLY 的含义是什么？
8. 对应到真实的业务请求，什么时候会用到 HTTP-ONLY？
9. JS 异步调用的方式有哪些？
10. Vue 中的设计模式有哪些？

11. Proxy 的优势，介绍一下 ES6 中的 Proxy

12. 实现一个睡眠函数

13. 实现 new 一个对象的过程

2022 年 8 月 12 日 百度三面

1. 自我介绍
2. 介绍一下自己做的插件
3. 介绍一下自己实习期间的收获
4. 除了百度还投了哪些公司
5. 在这些公司中，排序是什么，为什么这样排
6. 介绍一下读研期间最有成就感的事情
7. 介绍一下在实现这个插件过程中遇到的困难
8. 介绍一下你的职业规划
9. 介绍一下 vue 框架和 react 框架的区别及优缺点
10. 介绍一下 vue2.0 框架和 vue3.0 框架之间的区别
11. 介绍一下项目优化的手段（即压缩项目开发周期的手段）
12. 介绍一下项目性能优化的手段
13. 介绍一下你如何看待前端的发展以及前端工程师的发展
14. 面试官介绍了一下部门的业务
15. 反问了面试官对我个人的评价

2022 年 8 月 16 日 快手一面

1. 读程序说结果

```
li > a[href*="en-US"] > .inline-warning

li > a[href*="en-US"] > span
```

2. 读程序说结果

```
for(var i = 0; i < 5; i++) {
    setTimeout(function(i) {
    return function() {
       console.log(i);
     }
}(i), 1000)
}
console.log(i);
  function(i) {
    return function() {
       console.log(i);
     }
}(i)
for(var i = 0; i < 5; i++) {
    setTimeout(function(i) {
       console.log(i);
    }, 1000)
}
console.log(i);
```

3. 读程序说结果

```
var name = "222"

var a={

    name:"111",

    say:function(){

        console.info(this.name);

    }

}

var fun = a.say;

fun();

a.say();

var b = {

    name:"333",

    say:function(fun){

       fun();

}

}

b.say(a.say);

b.say = a.say;

b.say();

var fun1 = a.say.bind(b);

fun1.call(a);
```

4. 读程序说结果

给定一个数字数组，找到数组每个值后面第一个大于的它自身的元素，如果没找到，设为-1。最后返回一个包含所有找到的值的新数组

示例

输入：`[1, 5, 8, 7, 2, 9, 2]`

输出：`[5, 8, 9, 9, 9, -1, -1]`

要求：`O(n)`

5. HTML 中的盒模型

6. CSS 选择器优先级

浏览器通过优先级来判断哪一些属性值与一个元素最为相关，从而在该元素上应用这些属性值。

优先级划分为六个，分别为：

通配符选择器、标签选择器、类选择器、ID 选择器、内联样式、!import

通配符选择器--优先级为 0 级--计算值：0

标签选择器--优先级为 1 级--计算值：1

类选择器、属性选择器、伪类选择器--优先级为 2 级--计算值: 10

ID 选择器--优先级为 3 级--计算值: 100

内联样式--优先级为 4 级--计算值：1000

!import--优先级为 5 级--计算值: 10000

优先级如何进行比较呢？

从优先级高的开始，依次进行比较，如果当前比较的优先级相同，则右移一位，直至结束。

如果比较完发现相同的话，后面的样式就会覆盖前面的。

选择器	计算值	计算细则
*{ }	0	1个0级通配选择器，优先级数值计算结果为0
p { }	1	1个1级通配选择器，计算结果为1
ul > li { }	2	2个1级标签选择器，1个0级选择符，计算结果为1+0+1
li > ol + ol { }	3	3个1级标签选择器，2个0级选择符，计算结果为1+0+1+0+1
.foo { }	10	1个2级类名选择器，计算结果为10
a:not([rel=nofollow]) { }	11	1个标签选择器，1个0级否定伪类，1个2级属性选择器，计算结果为1+0+10
a:hover { }	11	1个1级标签选择器，1个2级伪类，计算结果为1+10
ol li.foo { }	12	2个1级标签选择器，1个2级类名选择器，1个0级空格选择符，计算结果为1+0+1+10
li.foo.bar { }	21	1个1级标签选择器，2个2级类名选择器，计算结果为1+10+10
#foo { }	100	1个3级id选择器，计算结果为100
#foo .bar p { }	111	1个3级id选择器，1个2级类名选择器，1个1级标签选择器，2个0级空格选择器，计算结果为100+10+1+0+0

参考自这篇博客：https://juejin.cn/post/6994580720807051301

7. 不定高元素的水平垂直居中

讲了 flex 布局，绝对定位+margin：auto，绝对定位+transform: translate(-50%,-50%);

8. 当 margin 为百分比的时候，是相对于父元素还是子元素的 50%来计算的？

实践了一下，是父元素。

而且块级元素上下方向上的 margin 会重叠。值为两者中绝对值较大的那一个。

2 个 block 块负 margin 的情况

- 负的 margin-top 会往上移动，并且下面的元素会整体跟着移动
- 正的 margin-top 就是把 content 往下移动
- 负的 margin-bottom 会出现下面元素往上移动，覆盖的情况
- 正的 margin-bottom 是拉大和下面元素的距离

9. 纵向上 margin 为百分比时，是基于父元素的宽的百分比还是基于父元素的高的百分比？

实践了一下，是基于父元素的宽来进行百分比的。padding 也是这样。

10. 除了 translate 为百分比时，是基于子元素的，其余的 margin、padding 这些都是基于父元素来进行处理的。

11. CSS3 动画属性

最常用的是 **animation** + **transform** 的方式

animation 规定动画的名称、运行的周期、运行的次数、运行的速度

transform 来规定动画怎样运动，是平移、旋转、缩放、斜向运动等等

除了上述两种之外，还有一个 **transition**,这个其实就是过渡。

```
    .one {
            width: 100px;
            height: 100px;
            margin: 200px auto;
            background-color: #cd4a48;
            -webkit-transition: width, height 2s ease;
            -moz-transition: width, height 2s ease;
            -ms-transition: width, height 2s ease;
            -o-transition: width, height 2s ease;
            transition: width, height 2s ease;
        }

        .one:hover {
            width: 300px;
            height: 300px;
        }
```

https://juejin.cn/post/6844903541887205389

2022 年 8 月 22 日 美团一面

1. vue 框架和 react 框架的区别
2. 项目中的难点以及你是怎么解决的
3. css 中的 position 取值有哪些
4. absolute 和 relative 的区别
5. absolute 是相对于什么来定位的
6. relative 是相对于什么来定位的
7. vue 中父子组件通信的方式有哪些
8. v-show 和 v-if 的区别
9. js 的基本数据类型有哪些
10. 最近有学什么新技术吗？
11. 手撕题：url 参数的截取
12. 反问部门技术栈

8. useMemo 和 ReactMemo 有什么区别
9. 反问： 你们考察校招生的标准是什么？

   答： 前端基础，基础是否扎实，是否有自驱力，是否有技术热情

   实习经历，项目经历

2022 年 8 月 31 日 美团三面

1. 从什么时候开始学习前端的？
2. 为什么会对前端感兴趣？
3. 你觉得前端的哪个模块比较难？
4. 担任组长和担任组员有什么区别？
5. 担任组长和担任组员的时候有没有受到过什么委屈？
6. 讲一下最近学的新技术?
7. 聊一下浏览器插件？
8. 介绍一下项目以及项目过程中的难点？
9. PC 端怎么没做所有的模块？
10. 反问：你们考察校招生的标准是什么？

答： 前端基础，项目经历，协作能力、实习经历、沟通能力、表达能力等

11. 反问：您觉得你们部门现在前端方向的技术难点是什么

答： 低代码平台、跨端

2022 年 9 月 5 日 阿里 一面

1. vue 和 react 的优缺点

首先说一下 vue 和 react 都有的共同的优缺点：

1. 优点

- vue 和 react 框架都有一个生命周期和钩子函数的概念，这个是它们的核心。

那么这个生命周期和钩子函数的好处有以下两个方面：

1. **细分了浏览器里输入 url 到页面渲染完成的过程**。

这样做的好处就是，我们可以在这个期间做很多性能优化的事情，比如在页面创建之前做什么事情，在元素挂载到页面上时做什么事情，在页面销毁之前和之后做什么事情。这些性能优化的手段可以使得用户体验更好。

2. **规范了前端程序员的编程习惯，利于协作开发**。

在 vue 框架和 react 框架中，都有一个钩子函数的概念，我们前端程序员在编程时，需要按照需求把需要在生命周期的不同阶段处理的任务放在对应的钩子函数里，这样其实我们整体的代码结构就是清晰的。这样也利于协作开发和维护。

- **vue 和 react 都有虚拟 DOM 和 diff 算法的概念**

所谓虚拟 DOM，指的是在 JS 内存中存在的与真实的 DOM 树具有相同结构的 DOM 树，当我们的页面需要更新前，首先会在虚拟 DOM 树上，根据 diff 算法找出需要更新的节点，然后再去更新真实的 DOM 元素。

因为操作 JS 对象消耗的性能远小于操作原生 DOM，因此虚拟 DOM 和 diff 算法是 vue 框架和 react 框架都有的优点。

2.缺点

- 无论是 vue 框架，还是 react 框架，它们都是单页面框架（SPA），不是应用级框架。也就是说它们需要配合 router 工具，状态管理库，网络请求库，webpack 打包机、UI 组件库这些第三方库来共同构造一款应用。

接着分别说一下他们两个各自的优缺点：

首先说一下 vue 的优缺点：

1. **优点：**

- 有很多现成的模板供开发者使用。
- vue 提供的模块可以使得页面的更新更加精准。

  我们都知道 js 是单线程的，而 vue 框架还是 react 框架，本质上都是 js 的框架，它们通过 diff 算法找出需要需要更新的节点，将其渲染到页面上的过程是异步的。因此如果这个渲染的过程很长的话，可能出现的一个问题就是页面发生卡顿。

  针对这个问题，vue 提供的模板可以在进行 DOM 算法比对的过程中更加精准和快速的找到需要更新的节点，从而使得页面渲染的速度更快。

  虽然说 react 框架在 16 版本之后引入了 fiber 来把 diff 比对的过程变成时间片轮转的形式来尽可能的提高用户体验，但它还是没有办法做到精准更新。

  因此这是 vue 相比于 react 框架在页面更新上的优势。
- 提供了双向绑定，方便视图层和数据层的同步更新。具体来说就是有 v-model，这是 vue

的最大特色之一。

- 移动端 vue 可使用的 UI 组件库更多

2. **缺点**：

- 没有 react-native 这种可以用 vue 框架来编写移动端应用程序的工具
- 其实从 vue3.0 在 vue2.0 基础上的改进，我们就可以归纳出 vue 相比于 react 框架存在的很多缺点。

  在 Vue3.0 中，支持包含生命周期的代码抽离形成单独的库，方便不同的组件进行调用。它解决的问题就是 vue2.0 被大家广为嘲笑的一个点就是不适合大型项目。因为很多 js 代码不能逻辑复用。所以这是 vue 相比于 react 存在的一个缺点。
- 生态没有 react 繁荣。这个一方面是 vue 框架提供的模板给程序员造成了很多的限制，一方面确实是影响力没有 react 强。

接下来说说 React 的优缺点：

1. **优点：**

- 生态繁荣

  相比于 Vue 框架只是由尤雨溪为代表的独立开发者在维护和迭代来说，react 框架背靠 Fackbook 大公司，社区生态很繁荣，有很多大神提供了各式各样的扩展来丰富 react 的功能。比如说状态管理库，vue 广为流传的是 vuex 和 pinia，这两个其实都是 vue 的核心开发人员提出来的。而 react，在状态管理库上，就有多种选择方案，我知道的有 redux，react redux，mobx 等等，而 react 自己提出的 useContext 和 useReducer 也可以满足状态管理库的功能。这么看下来，react 的生态确实要比 vue 繁荣很多
- 更灵活。

  vue 提供的指令方便了程序员的开发，但是也对程序员的开发做了一定的限制。而 React 本身就是 JS，并没有提供像 v-model 这样的指令，因此就需要程序员自己来开发。因为没有标准答案，我个人认为一定程序上是促进了 react 社区的生态繁荣。
- 丰富的第三方库。（也就是生态繁荣）

  这得益于 react 的灵活以及 react 框架提出的 hooks。使得各路大神封装了各式各样的类库。这个点其实也就是生态繁荣。

2. **缺点:**

- 移动端的 UI 组件库没有 vue 的多
- 更考验程序员的前端基本功，因为没有 vue 的模块化编程

3. Vue 和 React 框架的异同点

相同点: 其实就是前面说的几点：

- 生命周期和钩子函数
- 虚拟 DOM 和 diff 算法
- 不能算是企业应用级前端框架，需要配合路由库、UI 组件库、网络请求库、打包机来使用

不同点就是各自的优点吧：

vue 提供模板，react 不提供

Vue 没有 react 灵活

vue 的社区生态没有 react 繁荣

Vue 的 diff 算法原理和 React 不同

Vue 有双向绑定，react 没有

2. react 的受控组件和非受控组件

所谓受控组件，从字面意思上来看，就是说这个组件是受控制的。

对应到 HTML 标签中，可能不受控制的组件通常有以下几种：

input 输入框、form 表单、textarea 和 select 表单元素。

这类标签通常保持自己的状态，并根据用户输入进行更新。

而在 React 中，可变状态一般保存在组件的 state（状态）属性中，并且只能通过 setState 更新。

因此，为了使这些根据用户输入进行更新的组件受控，react 可以将渲染表单的 React 组件的更新控制在用户输入之后。

因此，**其值由 React 控制的输入表单元素称为“受控组件”。不由 React 控制的叫“非受控组件”**。

```
<input
  type="text"
  id="username"
  onChange={(e) => this.handleUsernameChange(e)}
  value={username}
/>
```

通过value和onChange来把input输入框变成受React控制的组件

对于非受控组件，React 是通过 Ref 来从 DOM 节点中获取表单数据。

然后通过 React.ref.current 来为 DOM 元素设置值。

React 受控组件和非受控组件

3. react 组件通信的几种方式

- 使用 props 传值
- 使用 redux 等状态管理库
- 使用 useContext 和 useReducer/useRef 这些 hooks

4. react 为什么要使用 redux 这种状态管理库？

- 首先状态管理库的作用就是便于不同组件之间的通信，将需要共享的变量抽离出来统一管理
- 我们可以设想一下如果没有状态管理库的话，react 中的非父子组件之间要如何通信。很容易想到的一种解决方案就是使用 webstorage，就是 localstorage 和 sessionstorage 这种，但是 webstorage 这种方案有几个严重的缺点，
- 首先，它做不到响应式，当我们的页面多处使用 webstorage 里存储的值时，我们改动一个，必须要编写触发页面更新的函数才能使页面上使用了这个 webstorage 值的变量都更新---很麻烦
- 不同组件可以对 webstorage 中的同一个变量编写不同的操作方法，这使得 webstorage 中的值不容易维护
- webstorage 中只能存储单一的变量，我们在读取或者写入的时候，如果想要先进行一些复杂操作的话，在每个写入的地方或者读取的地方都需要操作一遍，很麻烦。
- **不支持时间旅行**
- 因此就需要状态管理库来更方便的管理全局变量。

5. 网站性能优化的手段有哪些

- 采用代码分包的形式来减少首页的包体积，从而加速首页渲染
- 采用服务端渲染（SSR）和客户端渲染（CSR）和静态站点生成（SSG）相结合的方式来合理生成页面，从而加速页面渲染
- 采用虚拟滚动的方式来优化列表加载
- 采用节流和防抖来减少网络请求频率
- 缓存需要进行复杂计算的变量结果，避免页面卡顿
- 使用 preload 和 preconnect 和 dns-prefetch 和 prefetch 来提前加载在页面跳转时需要请求

的资源以及提前进行 dns 解析

- 将大任务进行拆解，从而降低复杂度
- 使用 web worker 单开一个线程来计算复杂度较高的任务
- 使用 webAssembly 来处理需要复杂计算的任务，因为 webAssembly 的运行性能接近原生，因此在许多计算耗时的场景上会被使用来优化。

6. 使用原生 node 编写一个 post 方法有哪些步骤

7. post 请求如何接收客户端传递过来的所有参数

8. node 中 file 内置库的原理是什么？

9. file 读取文件时采用的 stream 了解吗？

10. 项目有什么项目难点吗？

- 一个是登录鉴权，需要借助 redis 缓存数据库来做鉴权处理
- 一个是需要和百度 AI 进行对接，调用他们的 OCR 识别接口，在识别之前，需要将图片转成 base64 格式
- 一个是数据库表的主从表多表联查。

2022 年 9 月 19 日 阿里 二面

1. 自我介绍
2. 介绍一下浏览器脚本实现上的难点
3. 看一下浏览器脚本的源码和反馈
4. 很多网站都提供沉浸式阅读，相比之下你的脚本有什么用
5. 知道为什么 button 插不到页面的 DOM 节点上吗
6. 讲一下两个训练营的项目，讲一下自己在阿里训练营中承担的角色，以及项目成员的分工
7. 蒙层的图片和底部的图片轮廓没有重合
8. 为什么需要把 png 转成 svg--便于获取拼图每条边的坐标
9. 讲一下自己最有收获的项目，在里面承担了什么样的角色
10. 讲一下项目中是怎么使用 jwt 的
11. 脚本是否有收益
12. 讲一下微软实习的项目
13. 讲一下自己的职业规划

14. 拿了哪些厂的 offer
15. 面试官看了我的博客，说我从 2018 年初就开始写博客了，而且涉及的方向还挺广
16. 简历上写的专利具体是做了什么
17. 反问面试官对我的综合评价
18. 反问面试官更看重候选人的学历背景还是技术背景
19. 上一轮笔试哪个题做的不好？为什么？
20. 出了一道 LRU 缓存的题目

```
// 使用 map 来做缓存池对于 set 和 get 方法来说，时间复杂度都是 o(1)
// 但是 map 进行 delete 操作的时候其实是一个遍历的过程，因此还有可优化的空间
// 因为链表的删除操作，耗费的时间复杂度是 o(1),因此，我们就可以使用 map 和双向链表相结合的方式来优化 LRUCache 算法
// 链表的好处就是可以把随机存放的数据按照不同的要求联系起来
// 为啥需要双向链表而不是单向链表呢？
// 主要原因是：当我们需要删除一个节点时，如果是单向链表的话，我们没办法直接获取前驱节点，必须要通过一次遍历才能拿到。
// 这样时间复杂度就是 o(n)
// 但如果我们采用的是双向链表的话，直接修改要删除节点的前一个节点的 next 为当前要删除节点的下一个节点即可。
const linkLineMode = function (key = "", val = "") {
    this.val = val;
    this.key = key;
    this.pre = null;
    this.next = null;
}

// 设置链表初始状态下节点及它们之间的指向，（生成头节点和尾结点）
const linkLine = function () {
    let head = new linkLineMode("head", "head");
    let tail = new linkLineMode("tail", "tail");
    head.next = tail;
    tail.pre = head;
    this.head = head;
    this.tail = tail;
}

// 链表头节点添加，每次有新元素的时候就添加在头节点处，因此链表元素越靠前，说明元素等级越高
linkLine.prototype.append = function (node) {
    node.next = this.head.next;
    node.pre = this.head;
    this.head.next.pre = node;
    this.head.next = node;
}

// 链表删除指定节点
linkLine.prototype.delete = function (node) {
    node.pre.next = node.next;
    node.next.pre = node.pre;
```

2022 年 9 月 24 日 华为 一面

1. 自我介绍
2. 介绍项目中的难点和亮点
3. 介绍 JS 的优势
4. 手撕题：移动零

```
const main = function (arr){
    let count = 0;
    for (let i = 0; i < arr.length; i++){
        if (arr[i] === 0){
            count++;
        }else{
            arr[i - count] = arr[i];
        }
        if (i + count >= arr.length){
            arr[i] = 0;
        }
    }
    return arr;
}
console.log(main([0, 1, 0, 3, 1, 2]));
```

2022 年 9 月 24 日 华为 二面

1. 自我介绍
2. 为什么不留在微软
3. 介绍一下什么是跨域
4. 介绍一下，对于节假日想要定时修改网页主题这种应该怎么做

- 第一种方式，通过协商缓存在向服务器确认的时候返回新版本的文件
- 第二种方式，通过修改文件的 hash 从而使得客户端加载的是最新版的服务端文件

5. 如何查看网站的性能？

浏览器控制台有个 performance 模块，这是 window 的一个接口，通过这个接口，可以查看网页加载的性能

6. 网站性能优化手段有哪些？

- 缓存来减少网络请求频率
- 异步组件+代码分包来减少首页白屏时间
- 懒加载->虚拟滚动
- 节流防抖
- 缓存高性能计算的结果

7. 常用的鉴权方案有哪些

cookie+session

token+redis

jwt

8. 为什么我们在鉴权的时候不直接使用 cookie，而是要把它放在请求头里来鉴权

因为 cookie 是可盗取的，为了防止 CSRF 攻击和 xss 攻击，不能直接使用 cookie

9. 了解 CSRF 攻击和 XSS 攻击吗

10. 了解 Flutter 吗

11. 读过什么源码

12. vue 框架现在处于一个什么样的瓶颈

13. 介绍一下项目中的难点

14. 知道谷歌开发者大会吗

15. 最近有学什么新技术

16. 博客为什么只停留到了 5 月 29 号

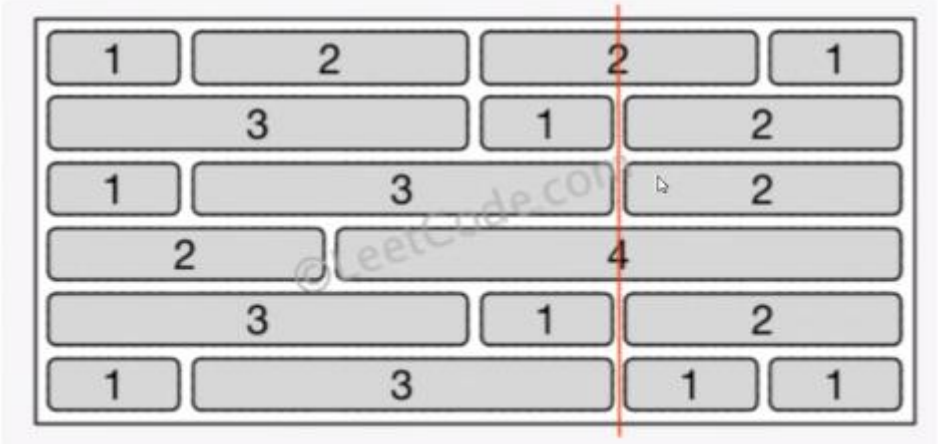
17. 手撕代码

WALL

示例 1：输入：wall = [[1,2,2,1],[3,1,2],[1,3,2],[2,4],[3,1,2],[1,3,1,1]]　　输出：2

示例 2：输入：wall = [[1],[1],[1]]　输出：3

接口：　Integer findMinSize(List<List<Integer>> wall)



```
const main = function (wall){
    // 统计每一行缝隙出现的位置
    // 按列来统计每一个缝隙出现的次数
    // 用总行数减去-max(缝隙出现的次数)
    let temp = [];
    for (let i = 0; i < wall.length; i++){
        let row = [];
        let sum = 0;
        for (let j = 0; j < wall[i].length; j++){
            sum = sum + wall[i][j]
            row.push(sum);
        }
        temp.push(row);
    }
    let obj = {};
    let sum = 0;
    for (let i = 0; i < temp.length; i++){
        for(let j = 0; j < temp[i].length - 1; j++){
            if (obj[temp[i][j]]){
                obj[temp[i][j]]++;
                sum = sum > obj[temp[i][j]] ? sum :
obj[temp[i][j]];
            }else{
                obj[temp[i][j]] = 1;
            }
        }
    }
    let ret = wall.length - sum;
    return ret;
}

console.log(main([[1, 2, 2, 1], [3, 1, 2], [1, 3, 2], [2, 4], [3,
1, 2], [1, 3, 1, 1]]))
```

2022 年 9 月 26 日 华为 主管面

1. 自我介绍
2. 看你写的博客大多数都是前端方向的，你的职业规划是什么？

3. 你的博客怎么主要是去年的？
4. 介绍一篇最热门的博客
5. 你简历上写的项目是个人项目还是实验室接的
6. 介绍一下项目中你认为最难的一个模板
7. 对面试的部门了解吗
8. 反问，华为对于新人的培养

2022 年 9 月 27 日 工商银行软件开发中心 技术一面

1. 自我介绍
2. 成绩怎么样
3. 介绍一下浏览器脚本的主要功能
4. 介绍一下浏览器脚本的亮点
5. 介绍一款你觉得不错的项目经历
6. 问一个 java 的问题，重载和重写的区别
7. 平常有什么爱好？技术方面和非技术方面？
8. 如何看待加班
9. 如何看待外派（工商银行软开中心第一年会外派到珠海等地）
10. 成都工商银行软件开发中心更想招后端方向的同学