

前端押题单页版

+ Add a property



本课程属于饥人谷《前端体系课》的精品课程（[点此购买](#)），咨询全课程可在 B 站或知乎发私信。



本讲义还没有更新完毕，每周五会更新一次。

课程介绍

押题的精髓：题目量少，思路清晰，命中率高。

本押题的目标薪资为 20k 以下。20k 以上的岗位**也需要会这些题目**，但是更看重你的经验、学历、谈吐、项目。

比如 20k 以上的岗位可能会问这样的问题：

1. 你是如何管理下属的？项目工期紧，你如何说服属下加班？
2. 电商项目如何做前后端技术选型？你在项目中遇到最难的问题是什么，如何解决的？
3. 如何实现前端性能和异常监控系统？
4. Vue/React 源码你了解吗？React 的 Fiber 架构说一说，Vue 为什么需要 DOM diff？

这些题目都没有明确的答案，不过请放心，如果工资没有开到 20k 以上，你就不用回答这些题目。

本押题分为以下模块：HTML、CSS、JavaScript、TypeScript、Vue、React、算法、Node.js、浏览器、跨平台、工程化、HTTP、安全、其他。

看起来模块很多，但每个模块的题目数量会尽量少，求精不求全。

本押题的目标是让观众在前端面试中能遇到 50% 以上的原题。

HTML 押题



HTML 5 有哪些新标签？

纯记忆型题目，记住不要提自己不熟悉的标签，因为很可能会变成下一道题。

文章相关：header main footer nav section article figure mark

多媒体相关：video audio svg canvas

表单相关：type=email type=tel

MDN 把所有标签都列在[这里](#)了，且有教程

切记：不要给自己挖坑，不要说自己说不清楚的名词

? Canvas 和 SVG 的区别是什么？

答题思路为：先说一，再说二，再说相同点，最后说不同点。

1. Canvas 主要是用笔刷来绘制 2D 图形的。
2. SVG 主要是用标签来绘制不规则矢量图的。
3. 相同点：都是主要用来画 2D 图形的。
4. 不同点：Canvas 画的是**位图**，SVG 画的是**矢量图**。
5. 不同点：SVG 节点过多时**渲染慢**，Canvas 性能更好一点，但写起来更复杂。
6. 不同点：SVG 支持分层和事件，Canvas 不支持，但是可以用库实现。

基本上考看别人的总结和平时自己写博客总结，否则答不好。

得分点：位图 v.s. 矢量图、渲染性能、是否支持分层和事件.....

? 如何理解 HTML 中的语义化标签

答法如下：

1. 是什么：语义化标签是一种写 HTML 标签的**方法论/方式**。
2. 怎么做：实现方法是遇到标题就用 h1 到 h6，遇到段落用 p，遇到文章用 article，主要内容用 main，边栏用 aside，导航用 nav.....（就是找到中文对应的英文）
3. 解决了什么问题：明确了 HTML 的书写规范
4. 优点是：一、适合搜索引擎检索；二、适合人类阅读，利于团队维护。
5. 缺点是：没有。
6. 怎么解决缺点：无需解决。

总结：「是什么、怎么做、解决了什么问题、优点是、缺点是、怎么解决缺点」

CSS 押题

? BFC 是什么

答题思路还是「是什么、怎么做、解决了什么问题、优点是、缺点是、怎么解决缺点」

是什么：

避免回答，直接把 BFC 翻译成中文「块级格式化上下文」即可，千万别解释。

怎么做：

背诵 BFC 触发条件，虽然 [MDN 的这篇文章](#) 列举了所有触发条件，但本押题告诉你只用背这几个就行了

- 浮动元素（元素的 float 不是 none）
- 绝对定位元素（元素的 position 为 absolute 或 fixed）
- 行内块 inline block 元素
- overflow 值不为 visible 的块元素
- 弹性元素（display 为 flex 或 inline-flex 元素的直接子元素）

解决了什么问题：

1. 清除浮动（为什么不用 .clearfix 呢？）
2. 防止 margin 合并
3. 某些古老的布局方式会用到（已过时）

优点：无。

缺点：有副作用。

怎么解决缺点：使用最新的 `display: flow-root` 来触发 BFC 就没有副作用了，但是很多人不知道。

? 如何实现垂直居中？

建议自己写博客总结一下，面试时直接把博客链接甩出去

七种方式实现垂直居中 · 语雀

如果 .parent 的 height 不写，你只需要...

 <https://www.yuque.com/docs/share/708bd899-0c46-47ea-...>



得分点：flex 方案、grid 方案、transform 方案.....

? CSS 选择器优先级如何确定

建议写博客总结，面试甩链接，这样就不用自己背了。

这里有 CSS 2.1 规格文档的权威算法：（但并不适用于 CSS 3）

属性赋值，层叠（Cascading）和继承

一旦用户代理已经解析了文档并构造好了文档树，它就必须给树中的每个元素上适用于目标媒体类型的每个属性赋值。属性的最终值是4步计算的结果：先通过指定来确定值（"指定值（specified value）"），接着处

 <http://www.ayqy.net/doc/css2-1/cascade.html#specificity>

如果记不住，可以记下这三句话：

1. 选择器越具体，其优先级越高
2. 相同优先级，出现在后面的，覆盖前面的
3. 属性后面加 !important 的优先级最高，但是要少用

? 如何清除浮动?

实践题，建议写博客，甩链接。

方法一，给父元素加上 .clearfix

```
.clearfix:after{ content: ''; display: block; /*或者 table*/ clear: both; }
.clearfix{ zoom: 1; /* IE 兼容*/ }
```

方法二，给父元素加上 overflow:hidden。

? 两种盒模型 (box-sizing) 的区别?

答题思路为：先说一，再说二，再说相同点，最后说不同点。

第一种盒模型是 content-box，即 width 指定的是 content 区域宽度，而不是实际宽度，公式为

实际宽度 = width + padding + border

第二种盒模型是 border-box，即 width 指定的是左右边框外侧的距离，公式为

实际宽度 = width

相同点是都是用来指定宽度的，不同点是 border-box 更好用。

一点有意思的小历史，请看视频。

JavaScript 押题基础篇

? JS 的数据类型有哪些?

纯记忆题，答案有 8 个词，建议背诵 10 次。

字符串、数字、布尔、undefined、null、大整数、符号、对象

string、number、boolean、undefined、null、bigint、symbol、object

提了就零分的答案有：数组、函数、日期。这些是类 class，不是类型 type

? 原型链是什么?

大概念题，答题思路为大概概念化成小概念（分割），抽象化成具体（举例）。

我的答题思路如下：

哦，原型链涉及到的概念挺多的，**我举例说明一下吧。**

假设我们有一个普通对象 `x={}`，这个 `x` 会有一个隐藏属性，叫做 `__proto__`，这个属性会指向 `Object.prototype`，即

```
x.__proto__ === Object.prototype // 原型
```

此时，我们说 **x 的原型** 是 `Object.prototype`，或者说 `Object.prototype` 是 `x` 的原型。

而这个 `__proto__` 属性的唯一作用就是用来指向 `x` 的原型的。

如果没有 `__proto__` 属性，`x` 就不知道自己的原型是谁了。

为什么我用问号来表示？因为这样你不容易晕。

接下来我来说说原型链，我还是举例说明吧。

假设我们有一个数组对象 `a=[]`，这个 `a` 也会有一个隐藏属性，叫做 `__proto__`，这个属性会指向 `Array.prototype`，即

```
a.__proto__ === Array.prototype
```

此时，我们说 `a` 的原型是 `Array.prototype`，跟上面的 `x` 一样。但又有一点不一样，那就是 `Array.prototype` 也有一个隐藏属性 `__proto__`，指向 `Object.prototype`，即

```
// 用 x 表示 Array.prototype x.__proto__ === Object.prototype
```

这样一来，`a` 就有两层原型：

1. `a` 的原型是 `Array.prototype`
2. `a` 的原型的原型是 `Object.prototype`

于是就通过隐藏属性 `__proto__` 形成了一个链条：

```
a ==> Array.prototype ==> Object.prototype
```

这就是原型链。

以上我对「原型链是什么」的回答。

怎么做：

看起来只要改写 `x` 的隐藏属性 `__proto__` 就可以改变 `x` 的原型（链）

```
x.__proto__ = 原型
```

但是这不是标准推荐的写法，为了设置 `x.__proto__`，推荐的写法是

```
const x = Object.create(原型) // 或 const x = new 构造函数() // 会导致  
x.__proto__ === 构造函数.prototype
```

没错，JS 就是这么别扭。

解决了什么问题：

在没有 Class 的情况下实现「继承」。以 `a ==> Array.prototype ==> Object.prototype` 为例，我们说：

1. a 是 Array 的实例，a 拥有 Array.prototype 里的属性
2. Array 继承了 Object（注意专业术语的使用）
3. a 是 Object 的间接实例，a 拥有 Object.prototype 里的属性

这样一来，a 就既拥有 Array.prototype 里的属性，又拥有 Object.prototype 里的属性。

优点：

简单、优雅。

缺点：

跟 class 相比，不支持私有属性。

怎么解决缺点：

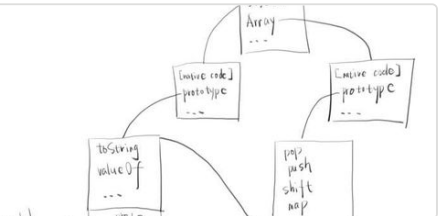
使用 class 呗。但 class 是 ES6 引入的，不被旧 IE 浏览器支持。

建议熟读这篇文章：

JS 中 `__proto__` 和 `prototype` 存在的意义是什么？

你的 JS 代码还没运行的时候，JS 环境里已经有一个 window 对象了 window 对象有一个 Object 属性，window.Object 是一个函数

[知 https://www.zhihu.com/question/56770432/answer/31534...](https://www.zhihu.com/question/56770432/answer/31534...)



? 这段代码中的 this 是多少？

把判断依据背下来才能全对

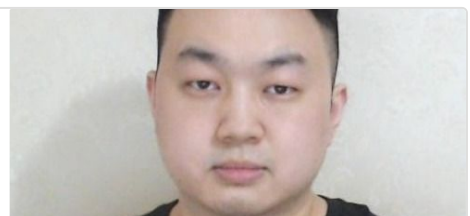
```
var length = 4; function callback() { console.log(this.length); // => 打印  
出什么? } const obj = { length: 5, method(callback) { callback(); } };  
obj.method(callback, 1, 2);
```

建议熟读这篇文章：

this 的值到底是什么？一次说清楚

你可能遇到过这样的 JS 面试题：var obj = { foo: function(){ console.log(this) } } var bar = obj.foo obj.foo() // 打印出的 this 是

[知 https://zhuanlan.zhihu.com/p/23804247](https://zhuanlan.zhihu.com/p/23804247)



完。

? JS 的 new 做了什么?

记忆题，建议博客，甩链接

1. 创建临时对象/新对象
2. 绑定原型
3. 指定 this = 临时对象
4. 执行构造函数
5. 返回临时对象

建议熟读这篇文章：

JS 的 new 到底是干什么的?

大部分讲 new 的文章会从面向对象的思路讲起，但是我始终认为，在解释一个事物的时候，不应该引入另一个更复杂的事物。

 <https://zhuanlan.zhihu.com/p/23987456>

大侠
生命值
行走(动作)
奔跑(动作)
死亡(动作)
攻击(动作)
防御(动作)

完。

? JS 的立即执行函数是什么?

概念题，「是什么、怎么做、解决了什么问题、优点是、缺点是、怎么解决缺点」

是什么：

声明一个匿名函数，然后立即执行它。这种做法就是立即执行函数。

怎么做：

```
(function){alert('我是匿名函数')}() // 用括号把整个表达式包起来
(function){alert('我是匿名函数')}() // 用括号把函数包起来 !function()
{alert('我是匿名函数')}() // 求反，我们不在意值是多少，只想通过语法检查。
+function(){alert('我是匿名函数')}() -function(){alert('我是匿名函数')}()
~function(){alert('我是匿名函数')}() void function(){alert('我是匿名函数')}
() new function(){alert('我是匿名函数')}() var x = function(){return '我是
匿名函数'}()
```

上面每一行代码都是一个立即执行函数。（举例法）

解决了什么问题：

在 ES6 之前，只能通过它来「创建局部作用域」。

优点：

兼容性好。

缺点：

丑。为什么这么丑？看视频分析。

怎么解决缺点：

使用 ES6 的 block + let 语法，即

```
{ let a = '我是局部变量啦' console.log(a) // 能读取 a } console.log(a) // 找不到 a
```

? JS 的闭包是什么？怎么用？

概念题，「是什么、怎么做、解决了什么问题、优点是、缺点是、怎么解决缺点」

是什么

闭包是 JS 的一种**语法特性**。

| 闭包 = 函数 + 自由变量

对于一个函数来说，变量分为：全局变量、本地变量、自由变量

怎么做

```
let count function add () { // 访问了外部变量的函数 count += 1 }
```

把上面代码放在「非全局环境」里，就是闭包。

| 注意，闭包不是 count，闭包也不是 add，闭包是 count + add 组成的整体。

怎么制造一个「非全局环境」呢？答案是立即执行函数：

```
const x = function () { var count function add () { // 访问了外部变量的函数 count += 1 } }()
```

但是这个代码什么用也没有，所以我们需要 `return add`，即：

```
const add2 = function () { var count return function add () { // 访问了外部变量的函数 count += 1 } }()
```

此时 add2 其实就是 add，我们可以调用 add2

```
add2() // 相当于 add() // 相当于 count += 1
```

至此，我们就实现了一个完整的「闭包的应用」。

注意：闭包 ≠ 闭包的应用，但面试官问你「闭包」的时候，你一定要答「闭包的应用」，这是规矩。

解决了什么问题：

1. 避免污染全局环境。（因为用的是局部变量）
2. 提供对局部变量的间接访问。（因为只能 `count += 1` 不能 `count -= 1`）
3. 维持变量，使其不被垃圾回收。

优点：

简单，好用。

缺点：

闭包**使用不当**可能造成内存泄露。

注意，重点是「使用不当」，不是闭包。

「闭包造成内存泄露」这句话以讹传讹很多年了，曾经旧版本 IE 的 bug 导致的问题，居然被传成这样了。

举例说明：

```
function test() { var x = {name: 'x'}; var y = {name: 'y', content: "-----这里很长，有一万三千五百个字符那么长-----" } return function fn() { return x; }; } const myFn = test() // myFn 就是 fn 了 const myX = myFn() // myX 就是 x 了 // 请问，y 会消失吗？
```

对于一个正常的浏览器来说，y 会在**一段时间**后自动消失（被垃圾回收器给回收掉）。

但旧版本的 IE 并不是正常的浏览器，所以是 IE 的问题。

当然，你可以说

君子不立于危墙之下，我们应该尽量少用闭包，因为有些浏览器对闭包的支持不够好

但你不可说「闭包造成内存泄露」。对吗？

怎么解决缺点：

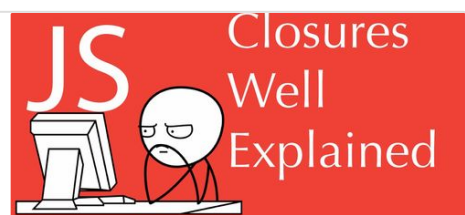
慎用，少用，不用。（我偏要用）

建议熟读这篇文章：

「每日一题」JS 中的闭包是什么？

大名鼎鼎的闭包！这一题终于来了，面试必问。请用自己的话简述 由于评论里不停有人给出错误的答案，这里先声明：如果你对

 <https://zhuanlan.zhihu.com/p/22486908>



完。

? JS 如何实现类?

方法一：使用原型

```
function Dog(name){ this.name = name this.legsNumber = 4 }
Dog.prototype.kind = '狗' Dog.prototype.say = function(){ console.log(`汪汪汪~ 我是${this.name}, 我有${this.legsNumber}条腿。`) } Dog.prototype.run
= function(){ console.log(`${this.legsNumber}条腿跑起来。`) } const d1 =
new Dog('啸天') // Dog 函数就是一个类 d1.say()
```

请试着实现一个 Chicken 类，没 name 会 say 会 fly。

方法二：使用 class

```
class Dog { kind = '狗' // 等价于在 constructor 里写 this.kind = '狗'
constructor(name) { this.name = name this.legsNumber = 4 // 思考: kind 放
在哪, 放在哪都无法实现上面一样的效果 } say(){ console.log(`汪汪汪~ 我是
${this.name}, 我有${this.legsNumber}条腿。`) } run(){
console.log(`${this.legsNumber}条腿跑起来。`) } } const d1 = new Dog('啸
天') d1.say()
```

请试着实现一个 Chicken 类，没 name 会 say 会 fly。

? JS 如何实现继承?

方法一：使用原型链

```
function Animal(legsNumber){ this.legsNumber = legsNumber }
Animal.prototype.kind = '动物' function Dog(name){ this.name = name
Animal.call(this, 4) // 关键代码1 } Dog.prototype.__proto__ =
Animal.prototype // 关键代码2, 但这句代码被禁用了, 怎么办 Dog.prototype.kind
= '狗' Dog.prototype.say = function(){ console.log(`汪汪汪~ 我是
${this.name}, 我有${this.legsNumber}条腿。`) } const d1 = new Dog('啸天')
// Dog 函数就是一个类 console.dir(d1)
```

如果面试官问被 ban 的代码如何替换，就说下面三句：

```
var f = function(){ } f.prototype = Animal.prototype Dog.prototype = new f()
```

方法二：使用 class

```
class Animal{ constructor(legsNumber){ this.legsNumber = legsNumber }  
run(){}} class Dog extends Animal{ constructor(name) { super(4)  
this.name = name } say(){ console.log(`汪汪汪~ 我是${this.name}, 我有  
${this.legsNumber}条腿。`) } }
```

完。

JavaScript 押题手写篇

? 手写节流 throttle、防抖 debounce

记忆题，写博客，甩链接。

节流：

```
// 节流就是「技能冷却中」 const throttle = (fn, time) => { let 冷却中 =  
false return (...args) => { if(冷却中) return fn.call(undefined, ...args)  
冷却中 = true setTimeout(()=>{ 冷却中 = false }, time) } } // 还有一个版本是  
在冷却结束时调用 fn // 简洁版，删掉冷却中变量，直接使用 timer 代替 const  
throttle = (f, time) => { let timer = null return (...args) => {  
if(timer) {return} f.call(undefined, ...args) timer = setTimeout(()=>{  
timer = null }, time) } }
```

使用方法：

```
const f2 = throttle(()=>console.log(1), 3000) f2() f2()
```

防抖：

```
// 防抖就是「回城被打断」 const debounce = (fn, time) => { let 回城计时器 =  
null return (...args)=>{ if(回城计时器 !== null) { clearTimeout(回城计时器)  
// 打断回城 } // 重新回城 回城计时器 = setTimeout(()=>{ fn.call(undefined,  
...args) // 回城后调用 fn 回城计时器 = null }, time) } }
```

使用方法:

```
const f2 = debounce(()=>console.log(1), 3000) f2() f2()
```

完。

? 手写发布订阅

记忆题, 写博客, 甩链接

```
const eventHub = { map: { // click: [f1, f2] }, on: (name, fn)=>{
  eventHub.map[name] = eventHub.map[name] || [] eventHub.map[name].push(fn)
}, emit: (name, data)=>{ const q = eventHub.map[name] if(!q) return
  q.map(f => f.call(null, data)) return undefined }, off: (name, fn)=>{
  const q = eventHub.map[name] if(!q){ return } const index = q.indexOf(fn)
  if(index < 0) { return } q.splice(index, 1) } } eventHub.on('click',
  console.log) eventHub.on('click', console.error) setTimeout(()=>{
  eventHub.emit('click', 'frank') },3000)
```

也可以用 class 实现。

```
class EventHub { map = {} on(name, fn) { this.map[name] = this.map[name]
  || [] this.map[name].push(fn) } emit(name, data) { const fnList =
  this.map[name] || [] fnList.forEach(fn => fn.call(undefined, data)) }
  off(name, fn) { const fnList = this.map[name] || [] const index =
  fnList.indexOf(fn) if(index < 0) return fnList.splice(index, 1) } } // 使
  用 const e = new EventHub() e.on('click', (name)=>{ console.log('hi '+
  name) }) e.on('click', (name)=>{ console.log('hello '+ name) })
  setTimeout(()=>{ e.emit('click', 'frank') },3000)
```

完。

? 手写 AJAX

记忆题, 写博客吧

```
const ajax = (method, url, data, success, fail) => { var request = new
  XMLHttpRequest() request.open(method, url); request.onreadystatechange =
  function () { if(request.readyState === 4) { if(request.status >= 200 &&
  request.status < 300 || request.status === 304) { success(request) }else{
  fail(request) } } }; request.send(); }
```

完。

? 手写简化版 Promise

代码不用记细节，主要说思路。

记忆题，写博客吧

```
class Promise2 { #status = 'pending' constructor(fn){ this.q = [] const
  resolve = (data)=>{ this.#status = 'fulfilled' const f1f2 =
  this.q.shift() if(!f1f2 || !f1f2[0]) return const x =
  f1f2[0].call(undefined, data) if(x instanceof Promise2) { x.then((data)=>
  { resolve(data) }, (reason)=>{ reject(reason) }) }else { resolve(x) } }
  const reject = (reason)=>{ this.#status = 'rejected' const f1f2 =
  this.q.shift() if(!f1f2 || !f1f2[1]) return const x =
  f1f2[1].call(undefined, reason) if(x instanceof Promise2){
  x.then((data)=>{ resolve(data) }, (reason)=>{ reject(reason) }) }else{
  resolve(x) } } fn.call(undefined, resolve, reject) } then(f1, f2){
  this.q.push([f1, f2]) } } const p = new Promise2(function(resolve,
  reject){ setTimeout(function(){ reject('出错') },3000) }) p.then( (data)=>
  {console.log(data)}, (r)=>{console.error(r)} )
```

完

? 手写 Promise.all

记忆题，写博客吧。

要点：

1. 知道要在原型上写
2. 知道 all 的参数和返回值
3. 知道用数组来记录结果
4. 知道只要有一个 reject 就整体 reject

```
Promise.prototype.myAll = function(promiseList){ return new
  Promise((resolve, reject)=>{ const resultList = [] let count = 0
  promiseList.map((promise, index)=>{ promise.resolve(result=>{
  resultList[index] = result count += 1 if(count >= promiseList.length){
  resolve(resultList) } }, (reason)=>{ reject(reason) }) }) }) }
```

进一步提问：是否知道 Promise.allSettled()

? 手写深拷贝

方法一，用 JSON：

```
const b = JSON.parse(JSON.stringify(a))
```

答题要点是指出这个方法有如下缺点：

1. 不支持 Date、正则、undefined、函数等数据
2. 不支持引用（即环状结构）
3. 必须说自己还会方法二

方法二，用递归：

要点：

1. 递归
2. 判断类型
3. 检查环
4. 不拷贝原型上的属性

```
const deepClone = (a, cache) => { if(!cache){ cache = new Map() // 缓存不能全局，最好临时创建并递归传递 } if(a instanceof Object) { // 不考虑跨 iframe if(cache.get(a)) { return cache.get(a) } let result if(a instanceof Function) { if(a.prototype) { // 有 prototype 就是普通函数 result = function(){ return a.apply(this, arguments) } } else { result = (...args) => { return a.call(undefined, ...args) } } } else if(a instanceof Array) { result = [] } else if(a instanceof Date) { result = new Date(a - 0) } else if(a instanceof RegExp) { result = new RegExp(a.source, a.flags) } else { result = {} } cache.set(a, result) for(let key in a) { if(a.hasOwnProperty(key)){ result[key] = deepClone(a[key], cache) } } return result } else { return a } } const a = { number:1, bool:false, str: 'hi', empty1: undefined, empty2: null, array: [ {name: 'frank', age: 18}, {name: 'jacky', age: 19} ], date: new Date(2000,0,1,20,30,0), regex: /\.(j|t)sx/i, obj: { name:'frank', age: 18}, f1: (a, b) => a + b, f2: function(a, b) { return a + b } } a.self = a const b = deepClone(a) b.self === b // true b.self = 'hi' a.self !== 'hi' //true
```

完。

? 手写数组去重

1. 使用计数排序的思路，缺点是只支持字符串
2. 使用 Set（面试已经禁止这种了，因为太简单）
3. 使用 Map，缺点是兼容性差了一点

```
var uniq = function(a){ var map = new Map() for(let i=0;i<a.length;i++){  
let number = a[i] // 1 ~ 3 if(number === undefined){continue}  
if(map.has(number)){ continue } map.set(number, true) } return  
[...map.keys()] }
```

完。

DOM 押题

? 请简述 DOM 事件模型

先经历从上到下的捕获阶段，再经历从下到上的冒泡阶段。

`addEventListener('click',fn,true/false)` 第三个参数可以选择阶段。

可以使用 `event.stopPropagation()` 来阻止捕获或冒泡。

? 手写事件委托

错误版（但是可能通过面试）

```
ul.addEventListener('click', function(e){ if(e.target.tagName.toLowerCase  
( ) === 'li'){ fn()}// 执行某个函数 } })
```

bug 在于，如果用户点击的是 li 里面的 span，就没法触发 fn，这显然不对。

好处

1. 节省监听器
2. 实现动态监听

坏处

调试比较复杂，不容易确定监听者。

解决坏处

解决不了

高级版（不用背）

思路是点击 span 后，递归遍历 span 的祖先元素看其中有没有 ul 里面的 li。

```
function delegate(element, eventType, selector, fn) { element.addEventListener(eventType, e => { let el = e.target while (!el.matches(selector)) { if (element === el) { el = null break } el = el.parentNode } el && fn.call(el, e, el) }) return element } delete(ul, 'click', 'li', f1)
```

完。

? 手写可拖曳 div

参考代码: <https://jsbin.com/munuzureya/edit?html,js,output>

要点:

1. 注意监听范围, 不能只监听 div
2. 不要使用 drag 事件, 很难用。
3. 使用 transform 会比 top / left 性能更好, 因为可以避免 reflow 和 repaint

HTTP 押题

? GET 和 POST 的区别有哪些?

区别一: 幂等性

1. 由于 GET 是读, POST 是写, 所以 GET 是幂等的, POST 不是幂等的。
2. 由于 GET 是读, POST 是写, 所以用浏览器打开网页会发送 GET 请求, 想要 POST 打开网页要用 form 标签。
3. 由于 GET 是读, POST 是写, 所以 GET 打开的页面刷新是无害的, POST 打开的页面刷新需要确认。
4. 由于 GET 是读, POST 是写, 所以 GET 结果会被缓存, POST 结果不会被缓存。
5. 由于 GET 是读, POST 是写, 所以 GET 打开的页面可被书签收藏, POST 打开的不行。

区别二: 请求参数

1. 通常, GET 请求参数放在 url 里, POST 请求数据放在 body (消息体) 里。(这里注意老师的讲解)
2. GET 比 POST 更不安全, 因为参数直接暴露在 URL 上, 所以不能用来传递敏感信息。(xjb扯)
3. GET 请求参数放在 url 里是有长度限制的, 而 POST 放在 body 里没有长度限制。(xjb扯)

区别三：TCP packet

- 1. GET 产生一个 TCP 数据包；POST 产生两个或以上 TCP 数据包。

⚠ 根据技术规格文档，GET 和 POST 最大的区别是语义；但面试官一般问的是实践过程中二者的区别，因此你需要了解服务器和浏览器对 GET 和 POST 的常见实现方法。

完。

❓ HTTP 缓存有哪些方案？

两种方案区别如下：

	缓存（强缓存）	内容协商（弱缓存）
HTTP 1.1	Cache-Control: max-age=3600 Etag: ABC	If-None-Match: ABC 响应状态码：304 或 200
HTTP 1.0	Expires: Wed, 21 Oct 2015 02:30:00 GMT Last-Modified: Wed, 21 Oct 2015 01:00:00 GMT	If-Modified-Since: Wed, 21 Oct 2015 01:00:00 GMT 响应状态码：304 或 200

面试官可能还会提到 `Pragma`，但 MDN 已经明确不推荐使用它。

更详细的内容可以看我的课程《全面攻克 Web 性能优化》中的《缓存与内容协商》视频。

❓ HTTP 和 HTTPS 的区别有哪些？

HTTPS = HTTP + SSL/TLS（安全层）

区别列表

- 1. HTTP 是明文传输的，不安全；HTTPS 是加密传输的，非常安全。
- 2. HTTP 使用 80 端口，HTTPS 使用 443 端口。
- 3. HTTP 较快，HTTPS 较慢。
- 4. HTTPS 的证书一般需要购买（但也有免费的），HTTP 不需要证书。

HTTPS 的细节可以看网上的博客，比较复杂，难以记忆，建议写博客总结一下。

[图解SSL/TLS协议 - 阮一峰的网络日志 \(ruanyifeng.com\)](#)

[HTTPS原理以及握手阶段](#)

? HTTP/1.1 和 HTTP/2 的区别有哪些?

区别列表

1. HTTP/2 使用了**二进制传输**，而且将 head 和 body 分成**帧**来传输；HTTP/1.1 是字符串传输。
2. HTTP/2 支持**多路复用**，HTTP/1.1 不支持。多路复用简单来说就是一个 TCP 连接从单车道（不是单行道）变成了几百个双向通行的车道。
3. HTTP/2 可以**压缩 head**，但是 HTTP/1.1 不行。
4. HTTP/2 支持**服务器推送**，但 HTTP/1.1 不支持。（实际上没多少人用）

更详细的内容可以看我的课程《[全面攻克 Web 性能优化](#)》中的《[什么是多路复用](#)》视频。

? TCP 三次握手和四次挥手是什么?

建立 TCP 连接时 server 与 client 会经历三次握手

1. 浏览器向服务器发送 TCP 数据：SYN(seq=x)
2. 服务器向浏览器发送 TCP 数据：ACK(seq=x+1) SYN(y)
3. 浏览器向服务器发送 TCP 数据：ACK(seq=y+1)

关闭 TCP 连接时 server 与 client 会经历四次挥手

1. 浏览器向服务器发送 TCP 数据：FIN(seq=x)
2. 服务器向浏览器发送 TCP 数据：ACK(seq=x+1)
3. 服务器向浏览器发送 TCP 数据：FIN(seq=y)
4. 浏览器向服务器发送 TCP 数据：ACK(seq=y+1)

为什么 2、3 步骤不合并起来呢？看起来是脱裤子放屁。

答案：2、3 中间服务器很可能还有数据要发送，不能提前发送 FIN。

TCP 的细节我作为一个十年的前端，也不太想去参透。

? 说说同源策略和跨域

同源策略是什么?

如果两个 URL 的协议、端口和域名都完全一致的话，则这两个 URL 是同源的。

http://www.baidu.com/s http://www.baidu.com:80/ssdasdsadad

同源策略怎么做？

只要在浏览器里打开页面，就默认遵守同源策略。

优点

保证用户的隐私安全和数据安全。

缺点

很多时候，前端需要访问另一个域名的后端接口，会被浏览器阻止其获取响应。

比如甲站点通过 AJAX 访问乙站点的 /money 查询余额接口，请求会发出，但是响应会被浏览器屏蔽。

怎么解决缺点

使用跨域手段。

1. JSONP（前端体系课有完整且详细的介绍）

- 甲站点利用 script 标签可以跨域的特性，向乙站点发送 get 请求。
- 乙站点**后端改造** JS 文件的内容，将数据传进回调函数。
- 甲站点通过回调函数拿到乙站点的数据。

2. CORS（前端体系课有完整且详细的介绍）

- 对于简单请求，乙站点在响应头里添加 `Access-Control-Allow-Origin: http://甲站点` 即可。
- 对于复杂请求，如 PATCH，乙站点需要：
 - 响应 OPTIONS 请求，在响应中添加如下的响应头

```
Access-Control-Allow-Origin: https://甲站点 Access-Control-Allow-Methods: POST, GET, OPTIONS, PATCH Access-Control-Allow-Headers: Content-Type
```

- 响应 POST 请求，在响应中添加 `Access-Control-Allow-Origin` 头。
- 如果需要附带身份信息，JS 中需要在 AJAX 里设置 `xhr.withCredentials = true`。

3. Nginx 代理 / Node.js 代理

- a. 前端 ⇒ 后端 ⇒ 另一个域名的后端

详情参考 [MDN CORS 文档](#)。

? Session、Cookie、LocalStorage、SessionStorage 的区别

- Cookie V.S. LocalStorage
 1. 主要区别是 Cookie 会被发送到服务器，而 LocalStorage 不会
 2. Cookie 一般最大 4k，LocalStorage 可以用 5Mb 甚至 10Mb（各浏览器不同）
 - LocalStorage V.S. SessionStorage
 1. LocalStorage 一般不会自动过期（除非用户手动清除）
 2. SessionStorage 在会话结束时过期（如关闭浏览器之后，具体由浏览器自行决定）
 - Cookie V.S. Session
 1. Cookie 存在浏览器的文件里，Session 存在服务器的文件里
 2. Session 是基于 Cookie 实现的，具体做法就是把 SessionID 存在 Cookie 里
- 其他区别请在网上找高票答案看看，自己写文章总结一下。

TypeScript 押题

? TS 和 JS 的区别是什么？有什么优势？

1. 语法层面：TypeScript = JavaScript + Type（TS 是 JS 的超集）
2. 执行环境层面：浏览器、Node.js 可以直接执行 JS，但不能执行 TS（Deno 可以执行 TS）
3. 编译层面：TS 有编译阶段，JS 没有编译阶段（只有转译阶段和 lint 阶段）
4. 编写层面：TS 更难写一点，但是**类型更安全**
5. 文档层面：TS 的代码写出来就是文档，IDE 可以完美**提示**。JS 的提示主要靠 TS 其他.....自己搜一下博客

? any、unknown、never 的区别是什么？

any V.S. unknown

二者都是顶级类型（top type），任何类型的值都可以赋值给顶级类型变量：

```
let foo: any = 123; // 不报错 let bar: unknown = 123; // 不报错
```

但是 unknown 比 any 的类型检查更严格，any 什么检查都不做，unknown 要求先收窄类型：

```
const value: unknown = "Hello World"; const someString: string = value;  
// 报错: Type 'unknown' is not assignable to type 'string'.(2322)
```

```
const value: unknown = "Hello World"; const someString: string = value as  
string; // 不报错
```

如果改成 any，基本在哪都不报错。所以能用 unknown 就优先用 unknown，类型更安全一点。

never

never 是底类型，表示不应该出现的类型，这里有一个[尤雨溪给出的例子](#)：

```
interface A { type: 'a' } interface B { type: 'b' } type All = A | B  
function handleValue(val: All) { switch (val.type) { case 'a': // 这里 val  
被收窄为 A break case 'b': // val 在这里是 B break default: // val 在这里是  
never const exhaustiveCheck: never = val break } }
```

现在你应该理解什么是「不应该出现的类型」了吧。

? type 和 interface 的区别是什么？

官方给出的[文档说明](#)：

1. 组合方式：interface 使用 extends 来实现继承，type 使用 & 来实现联合类型。
2. 扩展方式：interface 可以重复声明用来扩展，type 一个类型只能声明一次
3. 范围不同：type 适用于基本类型，interface 一般不行。
4. 命名方式：interface 会创建新的类型名，type 只是创建类型别名，并没有新创建类型。

其他.....建议搜一下博客。

? TS 工具类型 Partial、Required、Readonly、Exclude、Extract、Omit、ReturnType 的作用和实现？

1. 将英文翻译为中文。
 - a. Partial 部分类型
 - b. Required 必填类型
 - c. Readonly 只读类型
 - d. Exclude 排除类型
 - e. Extract 提取类型
 - f. Pick/Omit 排除 key 类型
 - g. ReturnType 返回值类型
2. 举例说明每个工具类型的用法。

Vue 2 押题

 Vue 2 的生命周期钩子有哪些？数据请求放在哪个钩子？

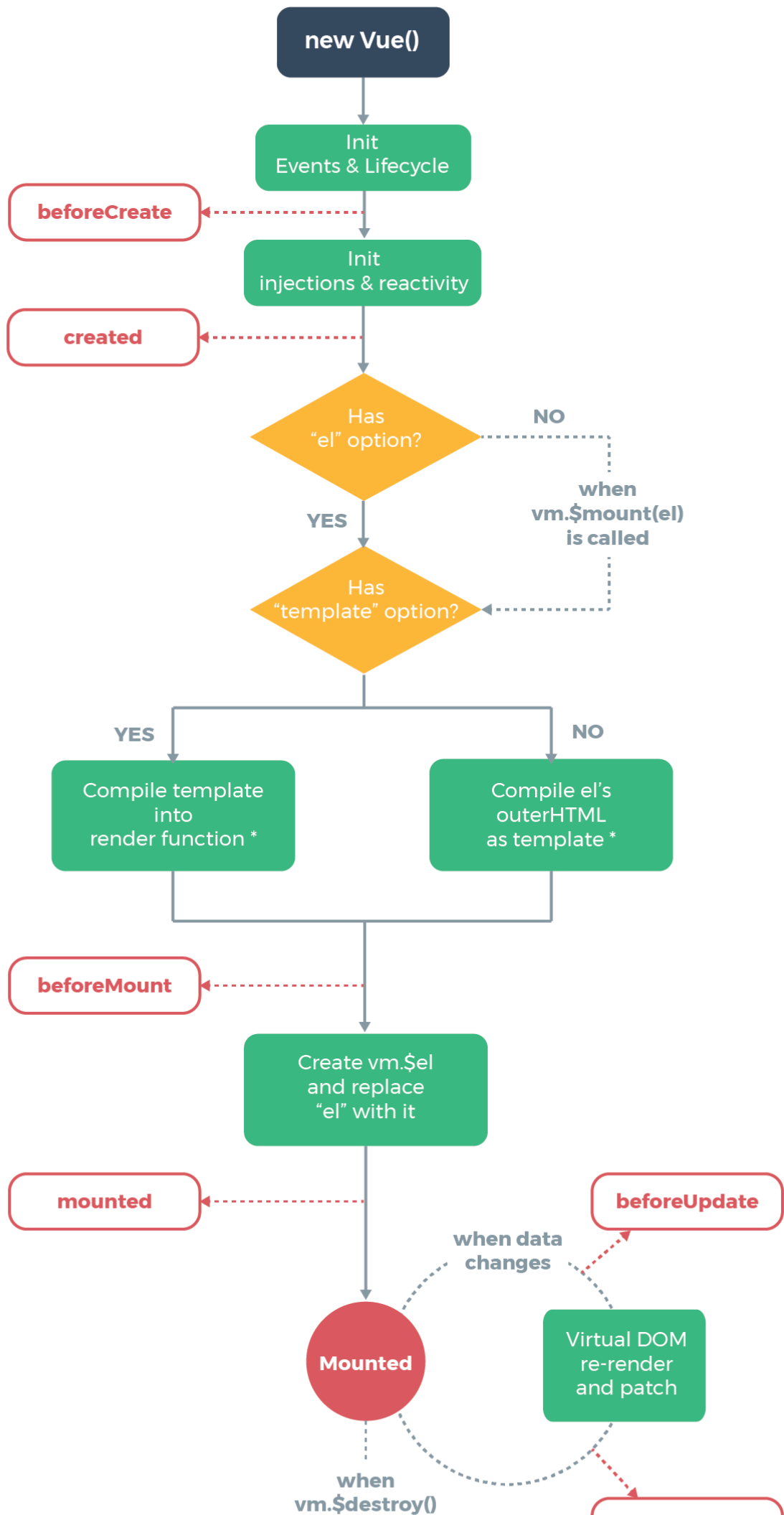
Vue 2 文档写得很清楚，红色空心框中的文字皆为生命周期钩子：

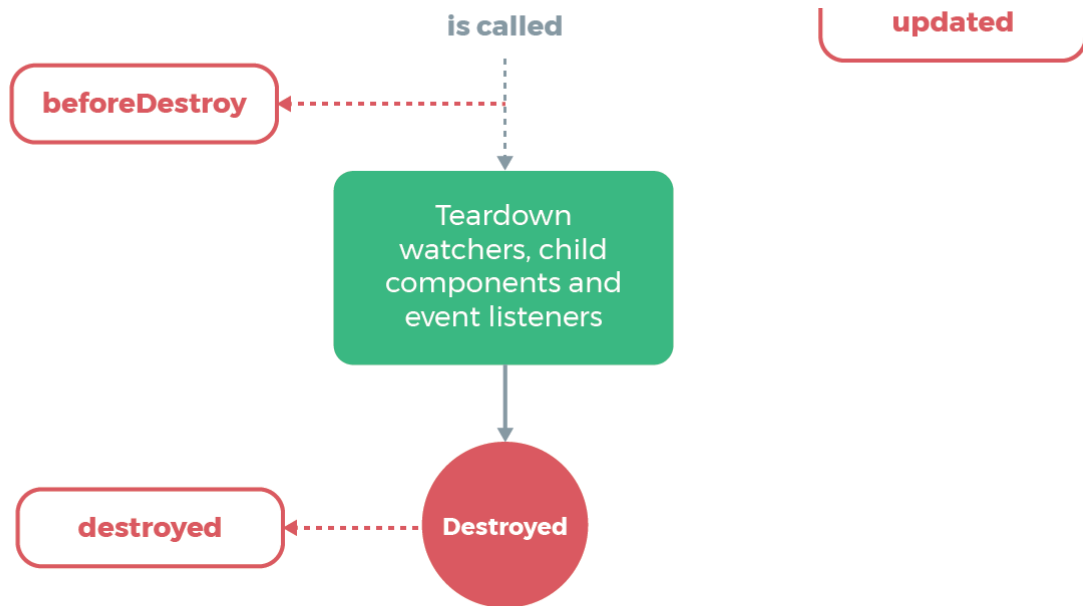
1. create x 2 (before + ed) - SSR
2. mount x 2
3. update x 2
4. destroy x 2

还有三个写在钩子列表里：

1. activated
2. deactivated
3. errorCaptured

请求放在 mounted 里面，因为放在其他地方都不合适（xjb扯）。





* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

完。

? Vue 2 组件间通信方式有哪些?

1. 父子组件：使用「props 和事件」进行通信
2. 爷孙组件：
 - a. 使用两次父子组件间通信来实现
 - b. 使用「provide + inject」来通信
3. 任意组件：使用 `eventBus = new Vue()` 来通信
 - a. 主要API 是 `eventBus.$on` 和 `eventBus.$emit`
 - b. 缺点是事件多了就很乱，难以维护
4. 任意组件：使用 Vuex 通信 (Vue 3 可用 Pinia 代替 Vuex)

? Vuex 用过吗？怎么理解？

1. 背下文档第一句：Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式 + 库

2. 说出核心概念的名字和作用：store/State/Getter/Mutation/Action/Module
 - a. store 是个大容器，包含以下所有内容
 - b. State 用来读取状态，带有一个 mapState 辅助函数
 - c. Getter 用来读取派生状态，附有一个 mapGetters 辅助函数
 - d. Mutation 用于同步提交状态变更，附有一个 mapMutations 辅助函数
 - e. Action 用于异步变更状态，但它提交的是 mutation，而不是直接变更状态。
 - f. Module 用来给 store 划分模块，方便维护代码

常见追问：Mutation 和 Action 为什么要分开？

答案：为了让代码更易于维护。（可是 Pinia 就把 Mutation 和 Action 合并了呀）
完。

? VueRouter 用过吗？怎么理解？


1. 背下文档第一句：Vue Router 是 Vue.js 的官方路由。它与 Vue.js 核心深度集成，让用 Vue.js 构建单页应用变得轻而易举。
2. 说出核心概念的名字和作用：`router-link` `router-view` 嵌套路由、Hash 模式和 History 模式、导航守卫、懒加载
3. 常见追问：
 - a. Hash 模式和 History 模式的区别？
 - i. 一个用的 Hash，一个用的 History API
 - ii. 一个不需要后端 nginx 配合，一个需要
 - b. 导航守卫如何实现登录控制？

```
router.beforeEach((to, from, next) => { if (to.path === '/login')  
  return next() if (to是受控页面 && 没有登录) return next('/login')  
  next() })
```

推荐阅读：

路由守卫

在系统路由跳转前做权限校验，是经常遇到的需求。本文将使用 Vue-Router 中的路由守卫功能实现权限控制和加载进度。Vue-

 https://blog.csdn.net/sinat_36521655/article/details/10612...

原创

完。

? Vue 2 是如何实现双向绑定的？

网上的博客讲得很绕，你可以尝试理解看看。

vue的双向绑定原理及实现 - canfoo#! - 博客园

使用vue也好有一段时间了，虽然对其双向绑定原理也有了解个大概，但也没好好探究下其原理实现，所以这次特意花了几晚时间

 <https://www.cnblogs.com/canfoo/p/6891868.html>

```
<script src="js/watcher.js"></script>
<script src="js/compile.js"></script>
<script src="js/index.js"></script>
<script type="text/javascript">

new SelfVue({
  el: '#app',
  data: {
    title: 'hello world',
    name: 'canfoo'
  },
  methods: {
    clickMe: function () {
      this.title = 'hello world';
    }
  }
})
```

1. 说明一般使用 `v-model` / `.sync` 实现，`v-model` 是 `v-bind:value` 和 `v-on:input` 的语法糖
 - a. `v-bind:value` 实现了 $\text{data} \Rightarrow \text{UI}$ 的单向绑定
 - b. `v-on:input` 实现了 $\text{UI} \Rightarrow \text{data}$ 的单向绑定
 - c. 加起来就是双向绑定了
2. 这两个单向绑定是如何实现的呢？
 - a. 前者通过 `Object.defineProperty` API 给 `data` 创建 `getter` 和 `setter`，用于监听 `data` 的改变，`data` 一变就会安排改变 `UI`
 - b. 后者通过 `template compiler` 给 `DOM` 添加事件监听，`DOM input` 的值变了就会去修改 `data`。

完。

Vue 3 押题

? Vue 3 为什么使用 Proxy?

1. 弥补 `Object.defineProperty` 的两个不足
 - a. 动态创建的 `data` 属性需要用 `Vue.set` 来赋值，Vue 3 用了 `Proxy` 就不需要了
 - b. 基于性能考虑，Vue 2 篡改了数组的 7 个 API，Vue 3 用了 `Proxy` 就不需要了
2. `defineProperty` 需要提前递归地遍历 `data` 做到响应式，而 `Proxy` 可以在真正用到深层数据的时候再做响应式（惰性）

? Vue 3 为什么使用 Composition API?

答案参考尤雨溪的博客：[Vue Function-based API RFC - 知乎 \(zhihu.com\)](#)

1. `Composition API` 比 `mixins`、高阶组件、`extends`、`Renderless Components` 等更好，原因有三：
 - a. 模版中的数据来源不清晰。
 - b. 命名空间冲突。
 - c. 性能。
2. 更适合 `TypeScript`

? Vue 3 对比 Vue 2 做了哪些改动?

官方文档写了（中文在这），这里列出几个容易被考的：

1. createApp() 代替了 new Vue()
2. v-model 代替了以前的 v-model 和 .sync
3. 根元素可以有不止一个元素了
4. 新增 Teleport 传送门
5. destroyed 被改名为 unmounted 了（before 当然也改了）
6. ref 属性支持函数了

其他建议自己看看写写。

React 押题

? 虚拟 DOM 的原理是什么？

1. 是什么

虚拟 DOM 就是虚拟节点（这句汉化很重要）。React 用 JS 对象来**模拟** DOM 节点，然后将其渲染成真实的 DOM 节点。

2. 怎么做

第一步是模拟

用 JSX 语法写出来的 div 其实就是一个虚拟节点：

```
<div id="x"> <span class="red">hi</span> </div>
```

这代码会得到这样一个对象：

```
{ tag: 'div', props: { id: 'x' }, children: [ { tag: 'span', props: {
  className: 'red' }, children: [ 'hi' ] } ] }
```

能做到这一点是因为 JSX 语法会被转译为 createElement 函数调用（也叫 h 函数），如下：

```
React.createElement("div", { id: "x"}, React.createElement("span", {
  class: "red" }, "hi" ) )
```

第二步是将虚拟节点渲染为真实节点

```
function render(vdom) { // 如果是字符串或者数字，创建一个文本节点 if
  (typeof vdom === 'string' || typeof vdom === 'number') { return
  document.createTextNode(vdom) } const { tag, props, children } = vdom
  // 创建真实DOM const element = document.createElement(tag) // 设置属性
  setProps(element, props) // 遍历子节点，并获取创建真实DOM，插入到当前节点
  children .map(render) .forEach(element.appendChild.bind(element)) //
  虚拟 DOM 中缓存真实 DOM 节点 vdom.dom = element // 返回 DOM 节点 return
  element } function setProps // 略 function setProp // 略 // 作者：
  Shenfq // 链接: https://juejin.cn/post/6844903870229905422
```

注意，如果节点发生变化，并不会直接把新虚拟节点渲染到真实节点，而是先经过 diff 算法得到一个 patch 再更新到真实节点上。

3. 解决了什么问题

- a. DOM 操作性能问题。通过虚拟 DOM 和 diff 算法减少不必要的 DOM 操作，保证性能不太差
- b. DOM 操作不方便问题。以前各种 DOM API 要记，现在只有 setState

4. 优点

- a. 为 React 带来了跨平台能力，因为虚拟节点除了渲染为真实节点，还可以渲染为其他东西。
- b. 让 DOM 操作的整体性能更好，能（通过 diff）减少不必要的 DOM 操作。

5. 缺点

- a. 性能要求极高的地方，还是得用真实 DOM 操作（目前没遇到这种需求）
- b. React 为虚拟 DOM 创造了**合成事件**，跟原生 DOM 事件不太一样，工作中要额外注意
 - i. 所有 React 事件都绑定到根元素，自动实现事件委托
 - ii. 如果混用合成事件和原生 DOM 事件，有可能会出 bug

6. 如何解决缺点

不用 React，用 Vue 3（笑）

 React 或 Vue 的 DOM diff 算法是怎样的？

1. 是什么

DOM diff 就是对比两棵虚拟 DOM 树的算法（废话很重要）。当组件变化时，会 render 出一个新的虚拟 DOM，diff 算法对比新旧虚拟 DOM 之后，得到一个 patch，然后 React 用 patch 来更新真实 DOM。

2. 怎么做

a. 首先对比两棵树的根节点

- i. 如果根节点的类型改变了，比如 div 变成了 p，那么直接认为整棵树都变了，不再对比子节点。此时直接删除对应的真实 DOM 树，创建新的真实 DOM 树。
- ii. 如果根节点的类型没变，就看看属性变了没有
 1. 如果没变，就保留对应的真实节点
 2. 如果变了，就只更新该节点的属性，不重新创建节点。
 - a. 更新 style 时，如果多个 css 属性只有一个改变了，那么 React 只更新改变的。

b. 然后同时遍历两棵树的子节点，每个节点的对比过程同上。

i. 情况一

```
<ul> <li>A</li> <li>B</li> </ul> <ul> <li>A</li> <li>B</li>
<li>C</li> </ul>
```

React 依次对比 A-A、B-B、空-C，发现 C 是新增的，最终会创建真实 C 节点插入页面。

ii. 情况二

```
<ul> <li>B</li> <li>C</li> </ul> <ul> <li>A</li> <li>B</li>
<li>C</li> </ul>
```

React 对比 B-A，会删除 B 文本新建 A 文本；对比 C-B，会删除 C 文本，新建 B 文本；（注意，并不是边对比边删除新建，而是把操作汇总到 patch 里再进行 DOM 操作。）对比空-C，会新建 C 文本。

你会发现其实只需要创建 A 文本，保留 B 和 C 即可，为什么 React 做不到呢？

因为 React 需要你加 key 才能做到：

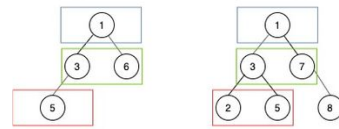
```
<ul> <li key="b">B</li> <li key="c">C</li> </ul> <ul> <li
key="a">A</li> <li key="b">B</li> <li key="c">C</li> </ul>
```

React 先对比 key 发现 key 只新增了一个，于是保留 b 和 c，新建 a。

以上是官方文档的内容，但是面试官想听的可能是源码分析之「双端交叉对比」：

Diff算法的核心就是 针对具有相同父节点的同层新旧子节点进行比较，而不是使用逐层搜索递归遍历的方式。时

 <https://canyuegongzi.github.io/web/vue/3.html#u...>



图文并茂地来详细讲讲Vue Diff算法 - 掘金

最近刚好看完Vue源码中的Diff算法，刚好在参加更文挑战，就做了一些动图还有流程图，图文并茂地来详细讲一讲，Vue的Diff算法叭。

 <https://juejin.cn/post/6971622260490797069>

我就当复读机帮你捋一遍。

? React 有哪些生命周期钩子函数？

React 的文档稍微有点乱，需要配合两个地方一起看才能记忆清楚：

React.Component - React

This page contains a detailed API reference for the React component class definition. It assumes you're familiar with

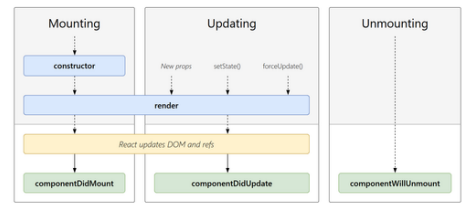
 <https://reactjs.org/docs/react-component.html#the-comp...>



React Lifecycle Methods diagram

Fully interactive and accessible React Lifecycle Methods diagram.

 <https://projects.wojtekmaj.pl/react-lifecycle-methods-diagr...>



总得来说：

1. 挂载时调用 constructor，更新时不调用
2. 更新时调用 shouldComponentUpdate 和 getSnapshotBeforeUpdate，挂载时不调用
3. should... 在 render 前调用，getSnapshot... 在 render 后调用
4. 请求放在 componentDidMount 里，最好写博客，容易忘。

? React 如何实现组件间通信

1. 父子组件通信：props + 函数
2. 爷孙组件通信：两层父子通信或者使用 Context.Provider 和 Context.Consumer
3. 任意组件通信：其实就变成了状态管理了
 - a. Redux
 - b. Mobx
 - c. Recoil

? 你如何理解 Redux?

1. 文档第一句话背下来: Redux 是一个状态管理库/状态容器。
2. 把 Redux 的核心概念说一下:
 - a. State
 - b. Action = type + payload 荷载
 - c. Reducer
 - d. Dispatch 派发
 - e. Middleware
3. 把 ReactRedux 的核心概念说一下:
 - a. connect()(Component)
 - b. mapStateToProps
 - c. mapDispatchToProps
4. 说两个常见的中间件 redux-thunk redux-promise

想深入了解可以看我的免费视频课:

来, 跟我一起手写 Redux! (建议 2 倍速播放) _哔哩...

课后练习: <https://docs.qq.com/doc/DU2h3UWNhbE9nUUJk>项目源码: <https://docs.qq.com/doc/DU0huaWt4cmxSZ0JD>加群请

 [https://www.bilibili.com/video/BV1dm4y1R7RK?from=search...](https://www.bilibili.com/video/BV1dm4y1R7RK?from=search)



? 什么是高阶组件 HOC?

参数是组件, 返回值也是组件的函数。什么都能做, 所以抽象问题就具体回答。

举例说明即可:

1. React.forwardRef
2. ReactRedux 的 connect
3. ReactRouter 的 withRouter

参考阅读: [「react进阶」一文吃透React高阶组件\(HOC\) - 掘金 \(juejin.cn\)](#)

? React Hooks 如何模拟组件生命周期?

1. 模拟 componentDidMount
2. 模拟 componentDidUpdate
3. 模拟 componentWillUnmount

代码示例如下：

```
import { useEffect,useState,useRef } from "react"; import "./styles.css";
export default function App() { const [visible, setNextVisible] =
useState(true) const onClick = ()=>{ setNextVisible(!visible) } return (
<div className="App"> <h1>Hello CodeSandbox</h1> {visible ? <Frank/> :
null} <div> <button onClick={onClick}>toggle</button> </div> </div> ); }
function Frank(props){ const [n, setNextN] = useState(0) const first =
useRef(true) useEffect(()=>{ if(first.current === true ){ return }
console.log('did update') }) useEffect(()=>{ console.log('did mount')
first.current = false return ()=>{ console.log('did unmount') } }, [])
const onClick = ()=>{ setNextN(n+1) } return ( <div>Frank <button
onClick={onClick}>+1</button> </div> ) }
```

完。

未完待续.....