**2023.02.01路刻**

1、回流重绘；

2、CSSOM；

3、地址栏URL敲回车；

4、react服务端渲染；

5、计时器；

6、取消事件监听；

7、引入css性能差异；

8、css性能提高；

9、useEffect返回值；

**2023.2.6快手**

1、服务端渲染;

2、http2.0;

3、实现 get(obj={a:{b:{c:2}}},path='a.b[c]',defVa=3);

4、es6功能;

5、useEffect、useMemo;

6、set、map;

7、new Promise(()=>{throw error}).then(()=>{},()=>{}).catch(()=>{}).then(()=>{});

8、inner-join;

9、webpack;

10、nodejs框架；

11、http缓存；

12、防抖节流；

13、垂直居中；

14、组件库设计思路；

15、单元测试；

**2023.2.7字节外包**

1、webpack优化；

2、es6功能；

3、set、map；

4、服务器端渲染；

5、http缓存；

6、事件循环：

async function async1() {

  console.log('async1 start')//2

  await async2()

  console.log('async1 end')//6

}

async function async2() {

  console.log('async2')//3

}

console.log('script start')//1

setTimeout(function () {

  console.log('setTimeout')//9

}, 0)

async1();

new Promise(function (resolve) {

  console.log('promise1')//4

  resolve();

}).then(function () {

  console.log('promise2')//7

}).then(function () {

  console.log('promise3')//8

})

console.log('script end')//5

7、数组扁平化函数实现：

const arr = [1, 2, [3, 4], [5, [6, 7]]]

const arrFlat = [1, 2, 3, 4, 5, 6, 7]

console.log(arr.flat(1));

console.log(arr.flat(2));

8、this绑定：

var a = 1

function get() {

  const a = 2

  return () => { console.log(a); }

}

const obj = {

  a: 3

}

const getA = get();

getA()

obj.getA = get();

obj.getA()

const getBind = get.bind(obj)()

getBind()

9、useState执行：

function Component() {

  const [a, setA] = useState(1);

  function click1() {

    setA(a + 1);

    console.log('a1---', a);

    setA(a + 1);

    console.log('a2---', a);

    setA(a + 1);

    console.log('a3---', a);

  }

  return (

    <>

      <button onClick={click1}>A</button>

    </>

  );

}

10、数组set去重；

11、跨域；

**2023.02.14 益科**

// 1：数组去重

const arr = [1, '1', 0, '0', 1, 0, '1', '0'];

function filterArr(arr) {

// TODO:

return Array.from(new Set(arr))

}

const newArr = filterArr(arr);

console.log('newArr: ',newArr); // newArr内容：[1, '1', 0, '0']

// 2：实现方法 getIds ，

// 从数组 customers 中查找 所有年龄（age）小于 30 的人并将其 customerId 重新组成一个新的数组返回，达到如下代码执行效果：

const customers = [

{ name: 'Jack', age: 23, customerId: '001' },

{ name: 'Jam', age: 15, customerId: '002' },

{ name: 'Lilei', age: 36, customerId: '003' },

{ name: 'Hanmei', age: 47, customerId: '004' },

{ name: 'Dav', age: 29, customerId: '005' },

];

function getIds(data) {

// TODO

const arrRes = []

data.forEach(element => {

if (element.age < 30) {

arrRes.push(element.customerId)

}

});

return arrRes

}

const ids = getIds(customers);

console.log('ids: ', ids); // ids内容：['001','002','005']

// 3：实现方法 isSimilar , 用于对比两个字符串是否相似，相似返回 true ，不相似返回 false，达到如下代码的执行结果：

// 字符串包含的字符种类以及每种字符的个数均相等即为相似。

const isSimilar = function (str) {

const strArr = str.split('')

const arrSource = this.split('')

if (strArr.length !== arrSource.length) {

return false

}

let count = 0

arrSource.forEach(char => {

const index = strArr.indexOf(char)

if (index !== -1) {

strArr[index] = null

count++

} else {

return

}

});

if (count === arrSource.length) {

return true

}

return false

}

String.prototype.isSimilar = isSimilar;

const str1 = 'ababc';

const str2 = 'babac';

const str3 = 'abccc';

const str4 = 'abcccw';

// const str5 = 'cab c';

const str5 = 'babcc';

const str6 = 'abccc';

console.log(str1.isSimilar(str2)); // 输出 true

console.log(str2.isSimilar(str1)); // 输出 true

console.log(str1.isSimilar(str3)); // 输出 false

console.log(str1.isSimilar(str4)); // 输出 false

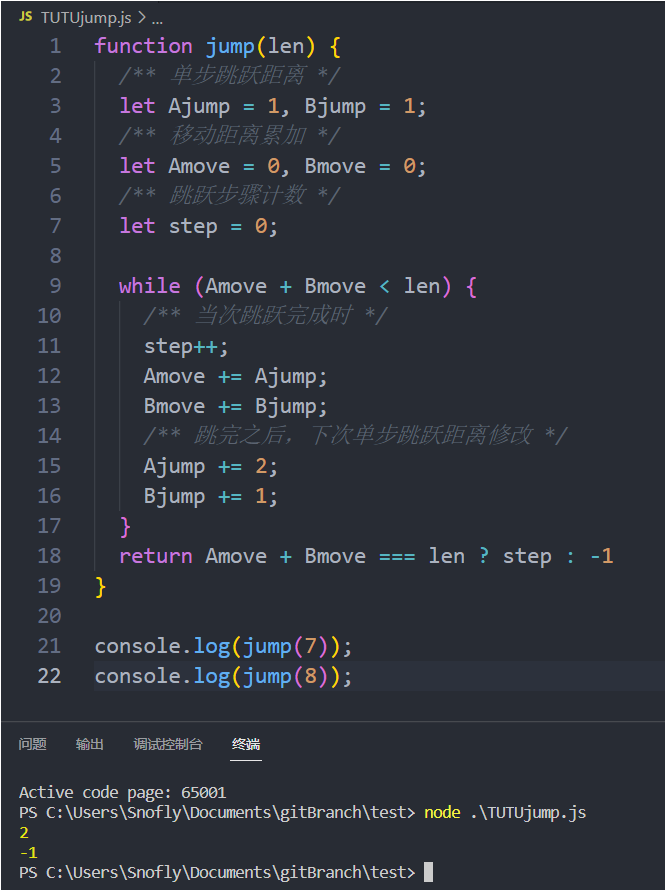
console.log(str5.isSimilar(str6)); // 输出 false

----------------------------------------

**2023.02.20 水木羽林**

笔试：

1. 相遇问题



1. 排序



面试：

1、URL回车

2023.02.20 奥本未来

笔试：

**一、阅读下面的源码，回答问题**

 1    var a = [];

 2    for (var i = 0; i < 10; i++) {

 3        a.push(

 4            function() {

 5                console.log(i);

 6            }

 7        );

 8    }

 9

10    for (var i = 0; i < 10; i++) {

11        a[i]();

12    }

1. **这段代码的输出是什么？解释一下为什么会有这样的输出。**

输出：0123456789。

解释：因为var将变量i提升到全局作用域，11行执行函数时，访问的i是被10行修改了的的全局变量。

我在下面改写一段代码来解释执行过程：

var a = [];

var i;

for (i = 0; i < 10; i++) {

  a.push(

    function () {

      console.log(i);

    }

  );

}

*/\* 数组元素是功能相同的函数, a[0] 和 a[9] 都一样\*/*

for (i = 0; i < 10; i++) {

  a[0]();

}

*/\*\* 变化i，可以看到实际上6行中访问的i就是全局变量i \*/*

for (i = 0; i < 10; i += 2) {

  a[0]();

}

1. **如果把10-12行的循环替换为：a.forEach(f => f()); 结果会有什么变化，为什么？**

输出：十个10。

解释：因为forEach遍历的时候，没有修改全局变量i，于是log时的i始终等于10。

1. **如果想要只修改第4-6行，实现输出结果和第2题的输出一致，可以怎么做？**

思路：将log放到异步队列中，当它开始执行时，变量i已经赋值为10了。

代码：

var a = [];

for (var i = 0; i < 10; i++) {

  a.push(

    function () {

      setTimeout(() => {

        console.log(i);

      }, 0);

    }

  );

}

for (var i = 0; i < 10; i++) {

  a[i]();

}

1. **JSX文件可以在JS文件中写类似HTML的标签。**
2. **浏览器引擎本身是不支持JSX的，我们编写的JSX文件为什么可以在浏览器中运行？**

答：因为JSX文件会先被转化为JS代码，再给浏览器执行。

1. **一个React程序的执行流程大概是什么样的？从JSX中编写的标签到文档中的一个DOM元素，中间会有哪些转换步骤？**

**流程：**

1. 触发更新
2. render阶段：计算更新会造成的副作用
3. commit阶段：在宿主环境执行副作用

通过修改组件 state 触发 fiber tree 更新，在 render 阶段，React 会计算 DOM 的更新，并将所有需要更新的 fiber 整理成一个 effect list，在 commit 阶段中， React 会遍历 effect list 执行所有的副作用，期间会执行更新相关的生命周期、挂载 DOM 等等。

**JSX到DMO的转换：**

1）Babel 会把 JSX 转译成一个名为 React.createElement() 函数调用。

2）React.createElement() 会预先执行一些检查，然后创建称为 React 元素的对象。

3）React 读取这些对象，然后使用它们来构建 DOM 以及保持随时更新：根据 react element 生成 fiber tree，再根据 fiber tree 生成 dom tree。

面试：

1、题目：原生js实现拖动。

2、useEffect 无第二参数，空数组，有值数组，值有ref的数组。

3、虚拟DOM。

4、函数组件和hooks。

5、useState 同步。

6、兄弟组件通信。

7、组件key的作用。

8、在if中使用useState。

9、伪元素的使用。