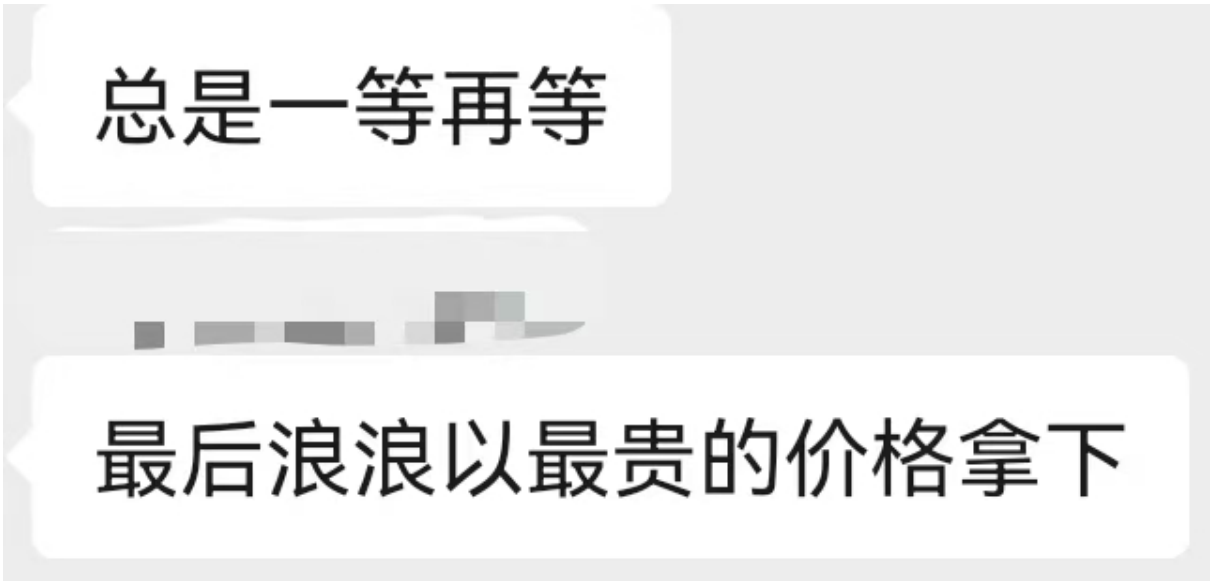
周记总结

生活日记：

在某天早上10点多摸鱼的时候决定买个键盘 纠结白色还是黑色 准备吃中饭来进行这个伟大的抉择

然后三个小时涨价30 淘宝我跟你拼了

蹲到正常价位再买



学习日记

本周进度

git学了一点 第一天报错网络连接不上 问了别人很多都有这个问题 最后换了个端口号 但是我预感后边提交还会有很多大问题 警惕心已经拉满了

js总感觉边学边忘

react看完了黑马的 本来想看的是尚硅谷 但是那版貌似有点老

周一先上手稍微搭建一下react吧

好菜

好好学习天天向上

保佑报错全部解决

小小知识点

箭头函数
()中定义参数，如果只有一个参数，可以不写括号;
{}中写函数体，如果函数体中只有返回值，可以不写return。

箭头函数和普通函数的区别:

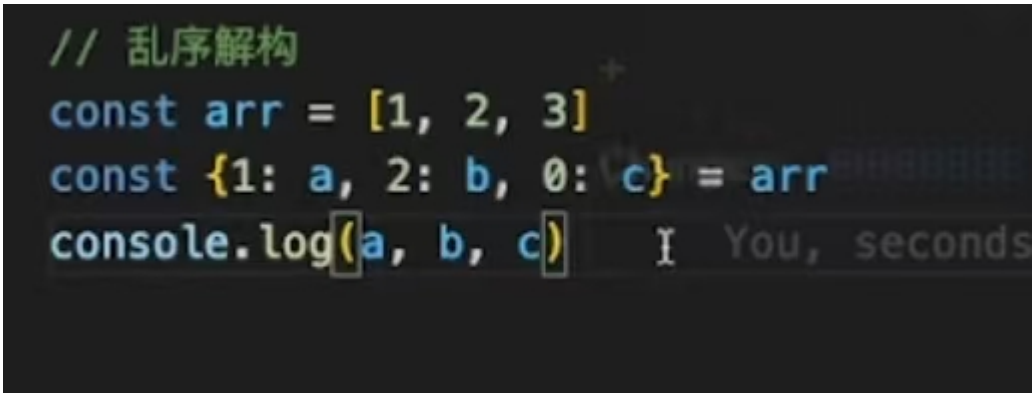
1. this指向不同
2. 普通函数，谁调用这个函数，this指向谁
3. 箭头函数，在哪里定义函数，this指向谁

解构赋值：

数组根据位置赋值

```
let a = 10;
let b = 20;
[a,b] = [b,a];
```

数组乱序解构：



输出：

2 3 1

对象通过属性名赋值

function createStudent( ){
let name ="小明";
let age = 2;
let friend = "小红";
return {
name:name,
age:age,
friend:friend}

}
obj =createStudent( );

解构：

let {name}= createStudent();
console.log(name);

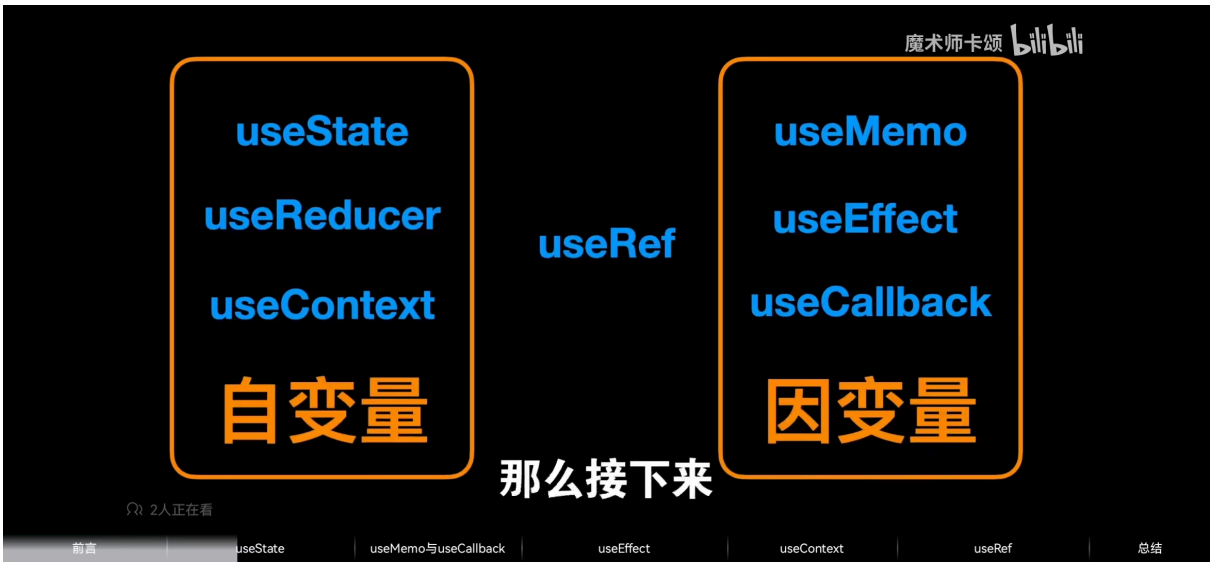
二维深层解构

```
// 深层对象
const obj = {
  nickname: '林三心',
  age: 20,
  gender: '男',
  doing: {
    morning: '睡觉',
    evening: '睡觉'
  }
}
const { nickname: myname, doing: { morning } } = obj

console.log(myname, morning)
```

原名：改名 输出：林三心睡觉

React Hooks



useState 自变量

x是自变量 setX是改变x的方法

useState会返回一个数组，包含两个元素：当前状态值（count）和更新状态的函数（setCount），在代码中通常使用解构赋值来获取这两个值，如const [count, setCount] = useState(0)。

useMemo useCallback 定义无副作用的因变量

useEffect定义有副作用的因变量

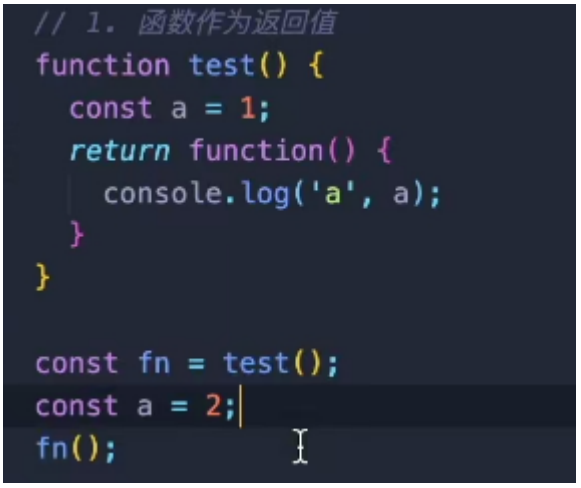
x变化后改变title

useReducer 方便操作更多自变量

useContent 跨组件层级地操作自变量

useRef 缓存数据 作为标记变量

闭包:一个函数和它的周围状态的引用捆绑在一起的组合



```
// 1. 函数作为返回值
function test() {
  const a = 1;
  return function() {
    console.log('a', a);
  }
}

const fn = test();
const a = 2;
fn();
```

a=1

```
// 2. 函数作为参数
function test(fn) {
  const a = 1;
  fn();
}

const a = 2;
function fn() {
  console.log('a', a);
}

test(fn);
```

a=2

this 的值是在 函数执行时 决定的，不是在函数定义时 决定

```
asyncTest() {
  setTimeout(function() {
    console.log('setTimeout 回调中的 this', this);
  }, 0)
}
```

```
setTimeout 回调中的 this                    index.js:14
  Window {window: Window, self: Window, docu
▶ ment: document, name: "", location: Locati
  on, …}
```

批处理

1. 事件循环 (Event Loop)

JavaScript 单线程的核心调度机制，任务分为：

- **同步任务**：立即执行（如普通函数调用）
- **异步任务**：推入任务队列（`setTimeout`→ 宏任务，`Promise`→ 微任务）

2. React 批处理更新

- **批处理触发时机**：在**同一事件循环周期内**的多个 `setState()`调用会被合并为一次更新

```jsx
const [count, setCount] = useState(0);

// 示例：点击事件中的连续更新
handleClick = () => {
  setCount(1);     // 🟢 不立即渲染
  setCount(2);     // 🟢 被合并
  // 最终只触发一次渲染，count 直接变为 2
}
```

3. 批处理如何利用事件循环

- **同步代码阶段**：所有 `setState`调用仅将更新请求存入队列，不触发渲染
- **微任务阶段**：React 在微任务队列中统一处理队列中的更新（类似 `Promise.then`的时机）

4. 突破批处理的场景

当更新发生在**异步回调**中时，会脱离批处理：

```jsx
handleClick = () => {
  setCount(1);                  // 🟢 批处理中

  setTimeout(() => {
    setCount(2);                // 🔴 脱离批处理！
    setCount(3);                // 🔴 单独触发渲染
  }, 0);
}
```

→ 原因：`setTimeout`回调属于**新的宏任务**，不在原事件循环周期内

原型对象

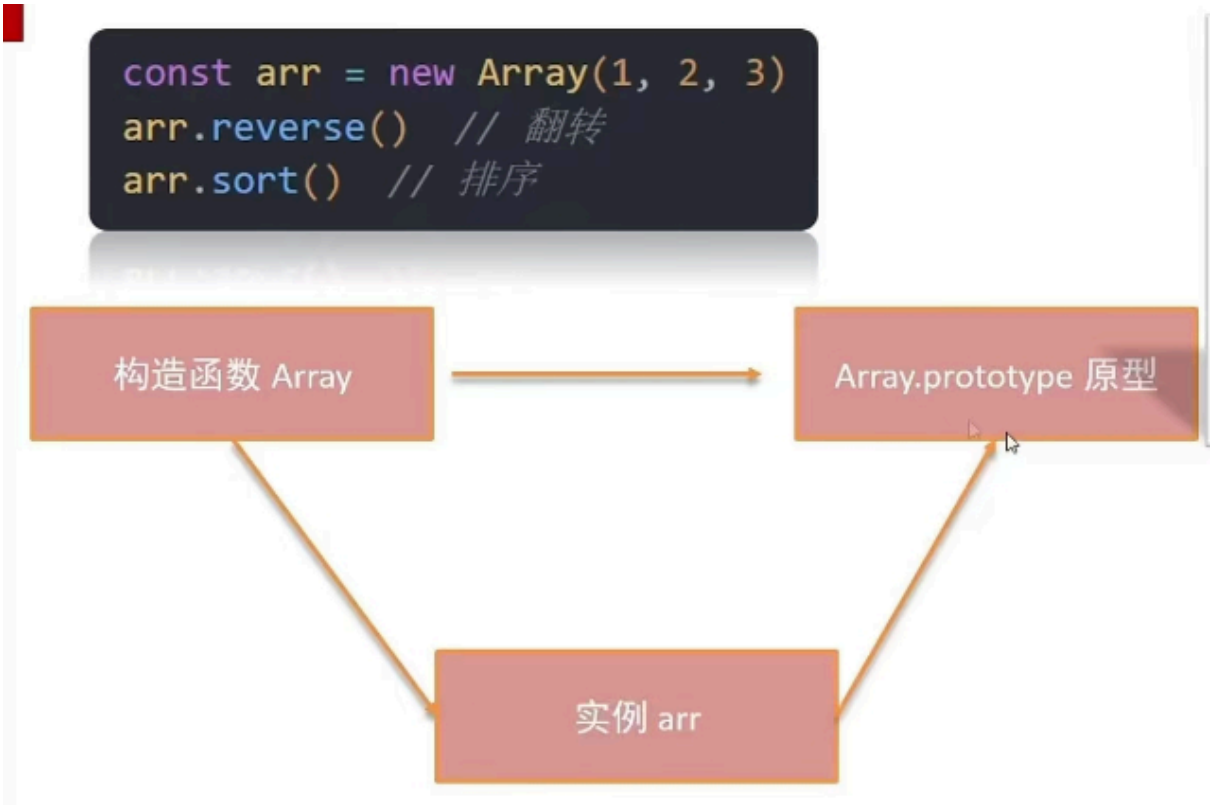
原型:每个函数都有prototype属性称之为原型

因为这个属性的值是个对象，也称为**原型对象**

作用:

1.存放一些属性和方法

2.在javaScript中实现继承



proto: 每个对象都有_proto__属性作用: 这个属性指向它的原型对象

原型链

原型链: 对象都有__proto__属性,这个属性指向它的原型对象,原型对象也是对象,也有__proto__属性,指向原型对象的原型对象,这样一层一层形成的链式结构称为原型链，最顶层找不到则返回 null