

## 1、React 中 keys 的作用是什么？

用于追踪那些列表中元素被修改、被添加或者被移除的辅助标识，用Key 值来判断该元素是新创建的还是被移动而来的元素，从而减少不必要的元素重渲染。

## 2、调用 setState 之后发生了什么？

React 会将传入的参数对象与组件当前的状态合并，然后触发所谓的调和过程 (Reconciliation)。经过调和过程，React 会以相对高效的方式根据新的状态构建 React 元素树并且着手重新渲染整个 UI 界面。在 React 得到元素树之后，React 会自动计算出新的树与老树的节点差异，然后根据差异对界面进行最小化重渲染。在差异计算算法中，React 能够相对精确地知道哪些位置发生了改变以及应该如何改变，这就保证了按需更新，而不是全部重新渲染。

## 3、react的diff算法（只有在React更新阶段才会有Diff算法的运用）

diff算法即差异查找算法，（即dom树的差异算法），React采用虚拟DOM 技术实现对真实DOM的映射，即React Diff算法的差异查找实质是对两个 JavaScript对象的差异查找；

## 7、react diff 原理

- 把树形结构按照层级分解，只比较同级元素。
- 给列表结构的每个单元添加唯一的 key 属性，方便比较。
- React 只会匹配相同 组件（class）的 component（这里面的 class 指的是组件的名字）
- 合并操作，调用 component 的 setState 方法的时候, React 将其标记为 dirty.到每一个事件循环结束, React 检查所有标记 dirty 的 component 重新绘制.
- 选择性子树渲染。开发人员可以重写 shouldComponentUpdate 提高 diff 的性能。

## 4、react性能优化

(1)、尽量减少对dom的操作，尤其是移动操作

## 5、react 生命周期函数

- 初始化阶段：
  - componentWillMount: 组件即将被装载、渲染到页面上
  - render:组件在这里生成虚拟的 DOM 节点
  - componentDidMount:组件真正在被装载之后

- 运行中状态：
  - `componentWillReceiveProps`:组件将要接收到属性的时候调用
  - `shouldComponentUpdate`:组件接收到新属性或者新状态的时候（可以返回 `false`，接收数据后不更新，阻止 `render` 调用，后面的函数不会被继续执行了）
  - `componentWillUpdate`:组件即将更新不能修改属性和状态
  - `render`:组件重新描绘
  - `componentDidUpdate`:组件已经更新
- 销毁阶段：
  - `componentWillUnmount`:组件即将销毁

## **shouldComponentUpdate 是做什么的，（react 性能优化是哪个周期函数？）**

`shouldComponentUpdate` 这个方法用来判断是否需要调用 `render` 方法重新描绘 `dom`。因为 `dom` 的描绘非常消耗性能，如果我们能在 `shouldComponentUpdate` 方法中能够写出更优化的 `dom diff` 算法，可以极大的提高性能。

## **另一个版本**

- **`componentWillMount`** 在渲染前调用,在客户端也在服务端。
- **`componentDidMount`** : 在第一次渲染后调用，只在客户端。之后组件已经生成了对应的DOM结构，可以通过`this.getDOMNode()`来进行访问。如果你想和其他JavaScript框架一起使用，可以在这个方法中调用`setTimeout`, `setInterval`或者发送AJAX请求等操作(防止异步操作阻塞UI)。
- **`componentWillReceiveProps`** 在组件接收到一个新的 `prop` (更新后)时被调用。这个方法在初始化`render`时不会被调用。
- **`shouldComponentUpdate`** 返回一个布尔值。在组件接收到新的`props`或者`state`时被调用。在初始化时或者使用`forceUpdate`时不被调用。

可以在你确认不需要更新组件时使用。

- **`componentWillUpdate`**在组件接收到新的`props`或者`state`但还没有`render`时被调用。在初始化时不会被调用。
- **`componentDidUpdate`** 在组件完成更新后立即调用。在初始化时不会被调用。

- **componentWillUnmount**在组件从 DOM 中移除之前立刻被调用。

### (1)、渲染组件的生命周期执行顺序：

- 1、**constructor**
- 2、componentWillMount
- 3、render
- 4、componentDidMount
- 5、componentWillUpdate
- 6、render
- 7、componentDidUpdate

### (2)、调用this.setState()时生命周期执行的顺序

- 1、componentWillUpdate
- 2、render
- 3、componentDidUpdate

### (3)、调用接口时，生命周期触发顺序

- 1、**componentWillUpdate**
- 2、**render**
- 3、**componentDidUpdate**
- 4、**componentWillUpdate**
- 5、**render**
- 6、**componentDidUpdate**

## 6、为什么虚拟 dom 会提高性能？

虚拟 dom 相当于在 js 和真实 dom 中间加了一个缓存，利用 dom diff 算法避免了没有必要的 dom 操作，从而提高性能。

用 JavaScript 对象结构表示 DOM 树的结构；然后用这个树构建一个真正的 DOM 树，插到文档当中，当状态变更的时候，重新构造一棵新的对象树。然后用新的树和旧的树进行比较，计算两棵树之间的差异，然后根据差异对界面进行最小化重渲染。

## 8、React 中 refs 的作用是什么？

Refs 是 React 提供给我们安全访问 DOM 元素或者某个组件实例的句柄。我们可以为元素添加 ref 属性然后在回调函数中接受该元素在 DOM 树中的句柄，该值会作为回调函数的第一个参数返回：

## 9、何为高阶组件(higher order component)

是一个以组件为参数并返回一个新组件的函数。

## 10、为什么建议传递给 setState 的参数是一个 callback 而不是一个对象

因为 this.props 和 this.state 的更新可能是异步的，不能依赖它们的值去计算下一个 state。

## 11、了解 redux 么，说一下 redux 把

- redux 是一个应用数据流框架，主要是解决了组件间状态共享的问题，原理是集中式管理，主要有三个核心方法，action，store，reducer，工作流程是 view 调用 store 的 dispatch 接收 action 传入 store，reducer 进行 state 操作，view 通过 store 提供的 getState 获取最新的数据。
- Redux 和 Flux 很像。主要区别在于 Flux 有多个可以改变应用状态的 store，在 Flux 中 dispatcher 被用来传递数据到注册的回调事件，但是在 redux 中只能定义一个可更新状态的 store，redux 把 store 和 Dispatcher 合并,结构更加简单清晰
- 新增 state,对状态的管理更加明确，通过 redux，流程更加规范了，减少手动编码量，提高了编码效率，同时缺点时当数据更新时有时候组件不需要，但是也要重新绘制，有些影响效率。一般情况下，我们在构建多交互，多数据流的复杂项目应用时才会使用它们

## 12、redux 有什么缺点

- 一个组件所需要的数据，必须由父组件传过来。
- 当一个组件相关数据更新时，即使父组件不需要用到这个组件，父组件还是会重新 render，可能会有效率影响，或者需要写复杂的 shouldComponentUpdate 进行判断。

## 13、reactJS的props.children.map函数来遍历会收到异常提示，为什么？应该如何遍历？

this.props.children的值有三种可能：

如果当前组件没有子节点，它就是undefined;  
如果有一个子节点，数据类型是object;  
如果有多个子节点，数据类型就是array。

系统提供React.Children.map()方法安全的遍历子节点对象

## 14、计算1-10000中出现的0 的次数

```
let s = new Array(10000).fill('').  
    .map((_, i) => i+1)  
    .filter(item => /0/.test(item))  
    .reduce((sum, item) => {return sum+String(item).match(/0/g).length}, 0);
```

这里要注意的是reduce函数必需设置初始值0，而不能忽略。若忽略该参数，则默认sum的初始值为首次循环的那个item，即数组的第一个元素，导致结果出错。

## 15、React有什么特点？

它使用虚拟DOM 而不是真正的DOM。  
它遵循单向数据流或数据绑定。

## 16、列出React的一些主要优点。

- 1) 它提高了应用的性能
- 2) 可以方便地在客户端和服务端使用
- 3) 由于 JSX，代码的可读性很好
- 4) 使用React，编写UI测试用例变得非常容易

## 17、redux的使用

- 1、分模块创建不同的reducer，不同的reducer执行不同的功能，最后暴露出来

```
1 let cartPage = {
2   goods: {
3     proId: 1,
4     proName: 'iphoneXs',
5     slogan: "一个肾都买不起的肾X",
6     proPrice: 12998,
7     proImg: 'https://img10.jiuxian.com/2018/0830/c72b3ee699e14de49510913440b01e014.jpg',
8     qty: 1
9   }
10 }
11
12 // 指定state修改逻辑
13 // 根据不同的action操作旧的state
14 const reducer = function(state=cartPage, action){
15   switch(action.type){
16     // 添加
17     case 'GOODS_CHANGE':
18       return {
19         ...state,
20         data: [...state.data, action.payload]
21       }
22
23     default:
24       return state;
25   }
26 }
27
28
29 export default reducer;
```

2、创建一个引入全部reducer文件的文件，并暴露出该文件

```
1 import { combineReducers } from 'redux';
2
3 import cartReducer from './cartReducer';
4 import goodsReducer from './goodsReducer';
5
6 // 合并成一个Reducer
7
8 const rootReducer = combineReducers({
9   cartReducer,
10  goodsReducer
11 });
12
13 export default rootReducer;
```

3、创建store



```

1 import {createStore} from 'redux';
2 // 引入Reducer
3 import reducer from '../reducers';
4
5 const store = createStore(reducer);
6
7 export default store;

```

#### 4、页面组件上的使用

```

}
let mapStateToProps = function(state){
  // state为保存在store中的数据
  return {
    carlist:state.cartReducer.data
  }
}

let mapDispatchToProps = function(dispatch){
  return {
    addToCar:(goods)=>{
      dispatch({
        type:'CART_ADD',
        payload:goods
      })
    },
    changeQty:(proId,qty)=>{
      dispatch({
        type:'CART_CHANGE_QTY',
        payload:{proId,qty}
      })
    }
  }
}

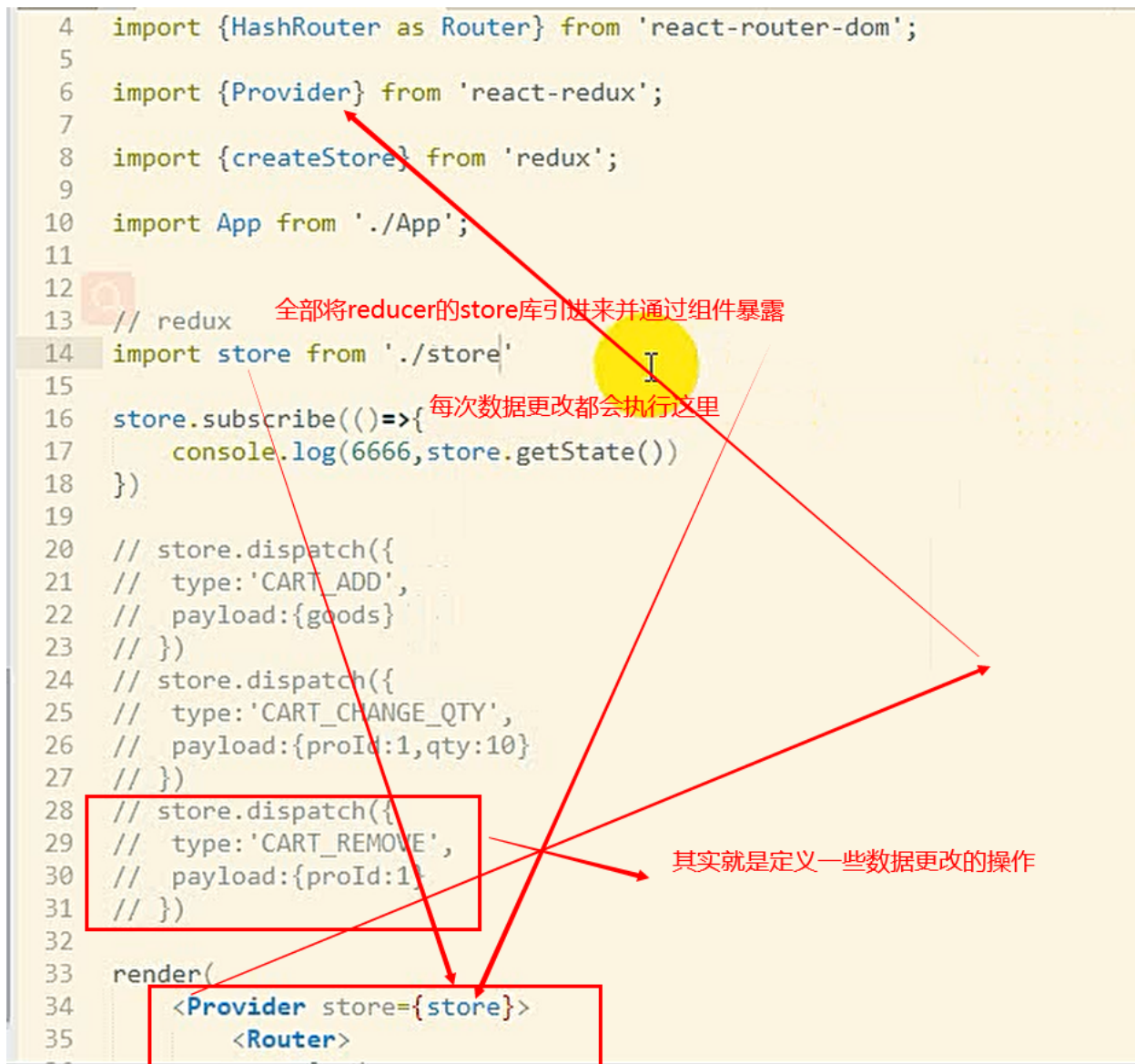
// 连接组件并指定暴露的数据
Home = connect(mapStateToProps,mapDispatchToProps)(Home);

```

```
34
35 let mapStateToProps = state=>{
36   return {
37     carlist:state.data
38   }
39 }
40
41 let mapDispatchToProps = dispatch=>{
42   return {
43     remove(proId){
44       dispatch(removeGoods(proId))
45     },
46     changeGoodsQty(proId,qty){
47       dispatch(changeQty(proId,qty))
48     }
49   }
50 }
51
52 Cart = connect(mapStateToProps,mapDispatchToProps)(Cart);
53
54 export default Cart;
```

## 5、主文件操作





## 19、js继承方式有几种

<https://www.cnblogs.com/ranyonsue/p/11201730.html>

## 20、为什么浏览器无法读取JSX?

浏览器只能处理 JavaScript 对象，而不能读取常规 JavaScript 对象中的 JSX。所以为了使浏览器能够读取 JSX，首先，需要用像 Babel 这样的 JSX 转换器将 JSX 文件转换为 JavaScript 对象，然后再将其传给浏览器。

## 21、与 ES5 相比，React 的 ES6 语法有何不同

## 1) require 与 import

```
1 // ES5
2 var React = require('react');
3
4 // ES6
5 import React from 'react';
```

## 2) export 与 exports

```
1 // ES5
2 module.exports = Component;
3
4 // ES6
5 export default Component;
```


## 22、JS字符串反转

**思想：** 最开始的思路是，先把字符串分割，然后倒序拼接成一个新的字符串


## 方法二: reverse(), join()

```
var name = "My city is WH";  
var resultStr = name.split('').reverse().join('');  
console.log(resultStr); // HW si ytic yM
```

## 方法三: charAt()



```
var name = "My city is WuHan";  
var nameArr = name.split('');  
var resultStr = '';  
for (var i = nameArr.length-1; i >= 0; i--) {  
    resultStr += name.charAt(i);  
}  
console.log(resultStr); // naHuW si ytic yM
```



## 23、节流和防抖:

### 节流:

```

<body>
  <button>点我试试</button>
  <script>
    var btn = document.querySelector('button')
    var flag = true
    btn.onclick = function () {
      if(flag) {
        flag = false
        console.log('发送请求...')

        setTimeout(()=>{
          flag = true
        }, 1000)
      }
    }
  </script>

```

// 快速点击按钮，你会发现，无论你点的有多快，只要点了一次，日志就打印一次  
 // 需求：在快速点击的过程中，如何降低一下日志打印的频率，比如1s内最多执行一回

防抖：

```

<body>
  <button>点我试试</button>
  <script>
    var btn = document.querySelector('button')
    var timer = null
    btn.onclick = function () {
      clearTimeout(timer)

      timer = setTimeout(()=>{
        console.log('发送请求了...')
      }, 1000)
    }
  </script>
</body>

```

// 快速点击按钮，你会发现，无论你点的有多快，只要点了一次，日志就打印一次  
 // 需求：在快速点击的过程中，如何只让最后一次能产生效果

## 24、this.setState()是否异步

```
export default class SetStateDemo extends React.Component{

  constructor(){
    super();
    this.state = {
      count:0
    }
  }

  async increment(){
    // this.setState({
    //   count:this.state.count+1
    // },() => {
    //   console.log(this.state.count);
    // })
    await this.setStateAsync({count:this.state.count+1});
    console.log(this.state.count);
  }

  setStateAsync(state){
    return new Promise((resolve) =>{
      this.setState(state, resolve);
    })
  }

  render(){
    return(
```