## **1. 搭建开发环境**

### **1.1 安装依赖包**

$ cnpm i webpack webpack-cli webpack-dev-server -D

### **1.2 package.json**

+ "scripts": {+ "build": "webpack --mode=development",+ "dev": "webpack-dev-server --mode=development --contentBase=./dist"+ },

## **2. 实现虚拟DOM**



### **2.1 src\index.js**

src\index.js

**import** { h } **from** './vdom';**const** root = document.getElementById('root');**const** oldVnode = h('div', { id: 'container' },

h('span', { style: { color: 'red' } }, 'hello'),

'world'

);console.log(oldVnode);

### **2.2 vdom\index.js**

src\vdom\index.js

**import** h **from** './h';**export** {

h

}

### **2.3 vdom\h.js**

src\vdom\h.js

**import** vnode **from** './vnode';**const** hasOwnProperty = Object.prototype.hasOwnProperty;**const** RESERVED\_PROPS = { key: true };**function** **h**(type, config, ...children) {

**const** props = {};

**let** key = null;

**if** (config) {

**if** (config.key) {

key = config.key;

}

**for** (**let** propName **in** config) {

**if** (hasOwnProperty.call(config, propName) && !RESERVED\_PROPS[propName]) {

props[propName] = config[propName];

}

}

}

**return** vnode(type, key, props, children.map((child, index) => {

**return** **typeof** child == 'number' || **typeof** child == 'string' ? vnode(undefined, undefined, undefined, undefined, child) : child;

}));

}**export** **default** h;

### **2.4 vdom\vnode.js**

src\vdom\vnode.js

**const** VNODE\_TYPE = 'VNODE\_TYPE';**function** **vnode**(type, key, props = {}, children, text, DOMElement) {

**return** {

\_type: VNODE\_TYPE,

type, key, props, children, text, DOMElement

}

}**export** **default** vnode;

## **3.初次渲染**



### **3.1 src\index.js**

src\index.js

**import** { h, mount } **from** './vdom';**const** root = document.getElementById('root');**const** oldVnode = h('div', { id: 'container' },

h('span', { style: { color: 'red' } }, 'hello'),

'world'

);console.log(oldVnode);

mount(oldVnode, root);

### **3.2 vdom\index.js**

src\vdom\index.js

**import** h **from** './h';**import** { mount } **from** './patch';**export** {

h,

mount

}

### **3.3 vdom\vnode.js**

src\vdom\vnode.js

const VNODE\_TYPE = 'VNODE\_TYPE';

//创建虚拟DOM节点

function vnode(type, key, props = {}, children, text, DOMElement) {

return {

\_type: VNODE\_TYPE,

type, key, props, children, text, DOMElement

}

}+//是否是一个虚拟DOM节点+export function isVnode(vnode) {+ return vnode && vnode.\_type === VNODE\_TYPE;+}+//是否是相同的节点 类型相同并且key相同 key可能为null+export function isSameVnode(oldVnode, newVnode) {+ return oldVnode.key === newVnode.key && oldVnode.type === newVnode.type;+}

export default vnode;

### **3.4 vdom\patch.js**

src\vdom\patch.js

**import** vnode, { isVnode, isSameVnode } **from** './vnode';//把虚拟DOM节点封装成一个真实DOM节点**function** **createDOMElementFromVnode**(vnode) {

**let** children = vnode.children;

**let** type = vnode.type;

**let** props = vnode.props;

**if** (type) {

**let** domElement = vnode.domElement = document.createElement(type);

updateProperties(vnode);

**if** (Array.isArray(children)) {

children.forEach(childVnode => domElement.appendChild(createDOMElementFromVnode(childVnode)));

}

} **else** {

vnode.domElement = document.createTextNode(vnode.text);

}

**return** vnode.domElement;

}

**export** **function** **mount**(vnode, root) {

**let** newDOMElement = createDOMElementFromVnode(vnode);

root.appendChild(newDOMElement);

}**function** **updateProperties**(vnode, oldProps = {}) {

**let** domElement = vnode.domElement;

**let** newProps = vnode.props;

**let** oldStyle = oldProps.style || {};

**let** newStyle = newProps.style || {};

**for** (**let** oldAttrName **in** oldStyle)

**if** (!newStyle[oldAttrName])

domElement.style[oldAttrName] = "";

**for** (**let** oldPropName **in** oldProps)

**if** (!newProps[oldPropName])

**delete** domElement[oldPropName];

**for** (**let** propName **in** newProps) {

**if** (propName == 'style') {

**let** styleObject = newProps[propName];

**for** (**let** attr **in** styleObject)

domElement.style[attr] = styleObject[attr];//更新行内样式

} **else** {

domElement[propName] = newProps[propName];

}

}

}

## **4.替换成不同类型**

### **4.1 index.html**

dist\index.html

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>虚拟DOM和DOMDIFF</title>+ <style>+ li {+ width: 100px;+ color: #FFFFFF;+ text-align: center;+ transition: all 1s;+ }

</style>

</head>

### **4.2 src\index.js**

src\index.js

**import** { vnode, h, mount, patch } **from** './vdom';**const** root = document.getElementById('root');**const** oldVnode = h('ul', { id: 'container' },

h('li', { style: { backgroundColor: '#110000' }, key: 'A' }, 'A'),

h('li', { style: { backgroundColor: '#440000' }, key: 'B' }, 'B'),

h('li', { style: { backgroundColor: '#770000' }, key: 'C' }, 'C'),

h('li', { style: { backgroundColor: '#AA0000' }, key: 'D' }, 'D')

);

mount(oldVnode, root);//const newVnode = vnode(undefined, undefined, undefined, undefined, '新的文本');**const** newVnode = h('div', { id: 'container' }, '新的文本');

setTimeout(() => {

patch(oldVnode, newVnode);

}, 1000)

### **4.3 vdom\index.js**

src\vdom\index.js

import h from './h';+import vnode from './vnode';+import { mount, patch } from './patch';

export {

h,+ vnode,

mount,+ patch

}

### **4.4 patch.js**

src\vdom\patch.js

import vnode, { isVnode, isSameVnode } from './vnode';

//把虚拟DOM节点封装成一个真实DOM节点

function createDOMElementFromVnode(vnode) {

let children = vnode.children;

let type = vnode.type;

let props = vnode.props;

if (type) {

let domElement = vnode.domElement = document.createElement(type);

updateProperties(vnode);

if (Array.isArray(children)) {

children.forEach(childVnode => domElement.appendChild(createDOMElementFromVnode(childVnode)));

}

} else {

vnode.domElement = document.createTextNode(vnode.text);

}

return vnode.domElement;

}

export function mount(vnode, root) {

let newDOMElement = createDOMElementFromVnode(vnode);

root.appendChild(newDOMElement);

}

+export function patch(oldVnode, newVnode) {+ if (oldVnode.type != newVnode.type) {+ return oldVnode.domElement.parentNode.replaceChild(createDOMElementFromVnode(newVnode), oldVnode.domElement);+ }+}

function updateProperties(vnode, oldProps = {}) {

let domElement = vnode.domElement;

let newProps = vnode.props;

let oldStyle = oldProps.style || {};

let newStyle = newProps.style || {};

for (let oldAttrName in oldStyle)

if (!newStyle[oldAttrName])

domElement.style[oldAttrName] = "";

for (let oldPropName in oldProps)

if (!newProps[oldPropName])

delete domElement[oldPropName];

for (let propName in newProps) {

if (propName == 'style') {

let styleObject = newProps[propName];

for (let attr in styleObject)

domElement.style[attr] = styleObject[attr];//更新行内样式

} else {

domElement[propName] = newProps[propName];

}

}

}

## **5.一方有儿子**

### **5.1 src\index.js**

src\index.js

**import** { vnode, h, mount, patch } **from** './vdom';**const** root = document.getElementById('root');/\* const oldVnode = h('ul', { id: 'container' },

h('li', { style: { backgroundColor: '#110000' }, key: 'A' }, 'A'),

h('li', { style: { backgroundColor: '#440000' }, key: 'B' }, 'B'),

h('li', { style: { backgroundColor: '#770000' }, key: 'C' }, 'C'),

h('li', { style: { backgroundColor: '#AA0000' }, key: 'D' }, 'D')

);

mount(oldVnode, root);

const newVnode = h('ul', { id: 'newContainer', style: { border: '1px solid red', height: '10px' } }); \*/

**const** oldVnode = h('ul', { id: 'container', style: { border: '1px solid red', height: '10px' } });**const** newVnode = h('ul', { id: 'newContainer' },

h('li', { style: { backgroundColor: '#110000' }, key: 'A' }, 'A'),

h('li', { style: { backgroundColor: '#440000' }, key: 'B' }, 'B'),

h('li', { style: { backgroundColor: '#770000' }, key: 'C' }, 'C'),

h('li', { style: { backgroundColor: '#AA0000' }, key: 'D' }, 'D')

);

mount(oldVnode, root);

setTimeout(() => {

patch(oldVnode, newVnode);

}, 1000)

### **5.2 vdom\patch.js**

src\vdom\patch.js

import vnode, { isVnode, isSameVnode } from './vnode';

//把虚拟DOM节点封装成一个真实DOM节点

function createDOMElementFromVnode(vnode) {

let children = vnode.children;

let type = vnode.type;

let props = vnode.props;

if (type) {

let domElement = vnode.domElement = document.createElement(type);

updateProperties(vnode);

if (Array.isArray(children)) {

children.forEach(childVnode => domElement.appendChild(createDOMElementFromVnode(childVnode)));

}

} else {

vnode.domElement = document.createTextNode(vnode.text);

}

return vnode.domElement;

}

export function mount(vnode, root) {

let newDOMElement = createDOMElementFromVnode(vnode);

root.appendChild(newDOMElement);

}

export function patch(oldVnode, newVnode) {

if (oldVnode.type != newVnode.type) {

return oldVnode.domElement.parentNode.replaceChild(createDOMElementFromVnode(newVnode), oldVnode.domElement);

}+ let domElement = newVnode.domElement = oldVnode.domElement;+ updateProperties(newVnode, oldVnode.props);+ let oldChildren = oldVnode.children;+ let newChildren = newVnode.children;+ if (oldChildren.length > 0 && newChildren.length > 0) {++ } else if (oldChildren.length > 0) {//老的有儿子,新的没儿子+ for (let i = 0; i < oldChildren.length; i++) {+ oldVnode.domElement.innerHTML = '';+ }+ } else if (newChildren.length > 0) {//新的有儿子,老的没儿子+ for (let i = 0; i < newChildren.length; i++) {+ oldVnode.domElement.appendChild(createDOMElementFromVnode(newChildren[i]));+ }+ }

}

function updateProperties(vnode, oldProps = {}) {

let domElement = vnode.domElement;

let newProps = vnode.props;

let oldStyle = oldProps.style || {};

let newStyle = newProps.style || {};

for (let oldAttrName in oldStyle)

if (!newStyle[oldAttrName])

domElement.style[oldAttrName] = "";

for (let oldPropName in oldProps)

if (!newProps[oldPropName])

delete domElement[oldPropName];

for (let propName in newProps) {

if (propName == 'style') {

let styleObject = newProps[propName];

for (let attr in styleObject)

domElement.style[attr] = styleObject[attr];//更新行内样式

} else {

domElement[propName] = newProps[propName];

}

}

}

## **6.前面或后面新增加元素**

* 前面后面添加元素 ABCD=>ABCDEF ABCD=>EFABCD
* 前面后面删除元素 ABCD=>ABC ABCD=>BCD

### **6.1 src\index.js**

src\index.js

**import** { vnode, h, mount, patch } **from** './vdom';**const** root = document.getElementById('root');

**const** oldVnode = h('ul', { id: 'container' },

h('li', { style: { backgroundColor: '#110000' }, key: 'A' }, 'A'),

h('li', { style: { backgroundColor: '#440000' }, key: 'B' }, 'B'),

h('li', { style: { backgroundColor: '#770000' }, key: 'C' }, 'C'),

h('li', { style: { backgroundColor: '#AA0000' }, key: 'D' }, 'D')

);**const** newVnode = h('ul', { id: 'newContainer' },

h('li', { style: { backgroundColor: '#440000' }, key: 'B' }, 'B'),

h('li', { style: { backgroundColor: '#770000' }, key: 'C' }, 'C'),

h('li', { style: { backgroundColor: '#AA0000' }, key: 'D' }, 'D'),

h('li', { style: { backgroundColor: '#110000' }, key: 'A' }, 'A')

);

mount(oldVnode, root);

setTimeout(() => {

patch(oldVnode, newVnode);

}, 1000)

### **6.2 vdom\patch.js**

src\vdom\patch.js

import vnode, { isVnode, isSameVnode } from './vnode';

//把虚拟DOM节点封装成一个真实DOM节点

function createDOMElementFromVnode(vnode) {

let children = vnode.children;

let type = vnode.type;

let props = vnode.props;

if (type) {

let domElement = vnode.domElement = document.createElement(type);

updateProperties(vnode);

if (Array.isArray(children)) {

children.forEach(childVnode => domElement.appendChild(createDOMElementFromVnode(childVnode)));

}

} else {

vnode.domElement = document.createTextNode(vnode.text);

}

return vnode.domElement;

}

export function mount(vnode, root) {

let newDOMElement = createDOMElementFromVnode(vnode);

root.appendChild(newDOMElement);

}

export function patch(oldVnode, newVnode) {

if (oldVnode.type != newVnode.type) {

return oldVnode.domElement.parentNode.replaceChild(createDOMElementFromVnode(newVnode), oldVnode.domElement);

}+ //如果新节点是文本节点，那么直接修改文本内容+ if (typeof newVnode.text !== 'undefined') {+ return oldVnode.domElement.textContent = newVnode.text;+ }

let domElement = newVnode.domElement = oldVnode.domElement;

updateProperties(newVnode, oldVnode.props);

let oldChildren = oldVnode.children;

let newChildren = newVnode.children;

if (oldChildren.length > 0 && newChildren.length > 0) {+ updateChildren(domElement, oldChildren, newChildren);

} else if (oldChildren.length > 0) {//老的有儿子,新的没儿子

for (let i = 0; i < oldChildren.length; i++) {

oldVnode.domElement.innerHTML = '';

}

} else if (newChildren.length > 0) {//新的有儿子,老的没儿子

for (let i = 0; i < newChildren.length; i++) {

oldVnode.domElement.appendChild(createDOMElementFromVnode(newChildren[i]));

}

}

}+function updateChildren(parentDOMElement, oldChildren, newChildren) {+ let oldStartIndex = 0, oldStartVnode = oldChildren[0];+ let oldEndIndex = oldChildren.length - 1, oldEndVnode = oldChildren[oldEndIndex];

+ let newStartIndex = 0, newStartVnode = newChildren[0];+ let newEndIndex = newChildren.length - 1, newEndVnode = newChildren[newEndIndex];+ while (oldStartIndex <= oldEndIndex && newStartIndex <= newEndIndex) {+ if (isSameVnode(oldStartVnode, newStartVnode)) {//顺序不变，顺序遍历完成+ patch(oldStartVnode, newStartVnode);+ oldStartVnode = oldChildren[++oldStartIndex];+ newStartVnode = newChildren[++newStartIndex];+ } else if (isSameVnode(oldEndVnode, newEndVnode)) {+ patch(oldEndVnode, newEndVnode);+ oldEndVnode = oldChildren[--oldEndIndex];+ newEndVnode = newChildren[--newEndIndex];+ }+ }+ if (newStartIndex <= newEndIndex) {//把没处理的新节点处理完 ABCD=>ABCDEF或ABCD=>EFABCD+ let beforeDOMElement = newChildren[newEndIndex + 1] == null ? null : newChildren[newEndIndex + 1].domElement;+ for (let i = newStartIndex; i <= newEndIndex; i++) {+ parentDOMElement.insertBefore(createDOMElementFromVnode(newChildren[i]), beforeDOMElement);+ }+ }+ if (oldStartIndex <= oldEndIndex) {+ for (let i = oldStartIndex; i <= oldEndIndex; i++) {+ if (oldChildren[i])+ parentDOMElement.removeChild(oldChildren[i].domElement);+ }+ }+}

function updateProperties(vnode, oldProps = {}) {

let domElement = vnode.domElement;

let newProps = vnode.props;

let oldStyle = oldProps.style || {};

let newStyle = newProps.style || {};

for (let oldAttrName in oldStyle)

if (!newStyle[oldAttrName])

domElement.style[oldAttrName] = "";

for (let oldPropName in oldProps)

if (!newProps[oldPropName])

delete domElement[oldPropName];

for (let propName in newProps) {

if (propName == 'style') {

let styleObject = newProps[propName];

for (let attr in styleObject)

domElement.style[attr] = styleObject[attr];//更新行内样式

} else {

domElement[propName] = newProps[propName];

}

}

}

## **7.头移尾和尾移头**

* 头部移动到尾部 ABCD=>BCDA
* 尾部移动到头部 ABCD=>DABC

### **7.1 src\index.js**

src\index.js

**import** { vnode, h, mount, patch } **from** './vdom';**const** root = document.getElementById('root');/\*\*

1. 前面后面添加元素 `ABCD=>ABCDEF` `ABCD=>EFABCD`

2. 前面后面删除元素 `ABCD=>ABC` `ABCD=>BCD`

\*/**const** oldVnode = h('ul', { id: 'container' },

h('li', { style: { backgroundColor: '#110000' }, key: 'A' }, 'A'),

h('li', { style: { backgroundColor: '#440000' }, key: 'B' }, 'B'),

h('li', { style: { backgroundColor: '#770000' }, key: 'C' }, 'C'),

h('li', { style: { backgroundColor: '#AA0000' }, key: 'D' }, 'D')

);**const** newVnode = h('ul', { id: 'newContainer' },

h('li', { style: { backgroundColor: '#000044' }, key: 'B' }, 'B1'),

h('li', { style: { backgroundColor: '#000077' }, key: 'C' }, 'C1'),

h('li', { style: { backgroundColor: '#0000AA' }, key: 'D' }, 'D1'),

h('li', { style: { backgroundColor: '#000011' }, key: 'A' }, 'A1')

);

mount(oldVnode, root);

setTimeout(() => {

patch(oldVnode, newVnode);

}, 1000)

### **7.2 vdom\patch.js**

src\vdom\patch.js

import vnode, { isVnode, isSameVnode } from './vnode';

//把虚拟DOM节点封装成一个真实DOM节点

function createDOMElementFromVnode(vnode) {

let children = vnode.children;

let type = vnode.type;

let props = vnode.props;

if (type) {

let domElement = vnode.domElement = document.createElement(type);

updateProperties(vnode);

if (Array.isArray(children)) {

children.forEach(childVnode => domElement.appendChild(createDOMElementFromVnode(childVnode)));

}

} else {

vnode.domElement = document.createTextNode(vnode.text);

}

return vnode.domElement;

}

export function mount(vnode, root) {

let newDOMElement = createDOMElementFromVnode(vnode);

root.appendChild(newDOMElement);

}

export function patch(oldVnode, newVnode) {

if (oldVnode.type != newVnode.type) {

return oldVnode.domElement.parentNode.replaceChild(createDOMElementFromVnode(newVnode), oldVnode.domElement);

}

//如果新节点是文本节点，那么直接修改文本内容

if (typeof newVnode.text !== 'undefined') {

return oldVnode.domElement.textContent = newVnode.text;

}

let domElement = newVnode.domElement = oldVnode.domElement;

updateProperties(newVnode, oldVnode.props);

let oldChildren = oldVnode.children;

let newChildren = newVnode.children;

if (oldChildren.length > 0 && newChildren.length > 0) {

updateChildren(domElement, oldChildren, newChildren);

} else if (oldChildren.length > 0) {//老的有儿子,新的没儿子

for (let i = 0; i < oldChildren.length; i++) {

oldVnode.domElement.innerHTML = '';

}

} else if (newChildren.length > 0) {//新的有儿子,老的没儿子

for (let i = 0; i < newChildren.length; i++) {

oldVnode.domElement.appendChild(createDOMElementFromVnode(newChildren[i]));

}

}

}

function updateChildren(parentDOMElement, oldChildren, newChildren) {

let oldStartIndex = 0, oldStartVnode = oldChildren[0];

let oldEndIndex = oldChildren.length - 1, oldEndVnode = oldChildren[oldEndIndex];

let newStartIndex = 0, newStartVnode = newChildren[0];

let newEndIndex = newChildren.length - 1, newEndVnode = newChildren[newEndIndex];

while (oldStartIndex <= oldEndIndex && newStartIndex <= newEndIndex) {

if (isSameVnode(oldStartVnode, newStartVnode)) {//顺序不变，顺序遍历完成

patch(oldStartVnode, newStartVnode);

oldStartVnode = oldChildren[++oldStartIndex];

newStartVnode = newChildren[++newStartIndex];

} else if (isSameVnode(oldEndVnode, newEndVnode)) {

patch(oldEndVnode, newEndVnode);

oldEndVnode = oldChildren[--oldEndIndex];

newEndVnode = newChildren[--newEndIndex];+ } else if (isSameVnode(oldStartVnode, newEndVnode)) {+ patch(oldStartVnode, newEndVnode);+ parentDOMElement.insertBefore(oldStartVnode.domElement, oldEndVnode.domElement.nextSibling);+ oldStartVnode = oldChildren[++oldStartIndex];+ newEndVnode = newChildren[--newEndIndex];+ } else if (isSameVnode(oldEndVnode, newStartVnode)) {+ patch(oldEndVnode, newStartVnode);+ parentDOMElement.insertBefore(oldEndVnode.domElement, oldStartVnode.domElement);+ oldEndVnode = oldChildren[--oldEndIndex];+ newStartVnode = newChildren[++newStartIndex];+ }

}

if (newStartIndex <= newEndIndex) {//把没处理的新节点处理完

let beforeDOMElement = newChildren[newEndIndex + 1] == null ? null : newChildren[newEndIndex + 1].domElement;

for (let i = newStartIndex; i <= newEndIndex; i++) {

parentDOMElement.insertBefore(createDOMElementFromVnode(newChildren[i]), beforeDOMElement);

}

}

if (oldStartIndex <= oldEndIndex) {

for (let i = oldStartIndex; i <= oldEndIndex; i++) {

if (oldChildren[i])

parentDOMElement.removeChild(oldChildren[i].domElement);

}

}

}

function updateProperties(vnode, oldProps = {}) {

let domElement = vnode.domElement;

let newProps = vnode.props;

let oldStyle = oldProps.style || {};

let newStyle = newProps.style || {};

for (let oldAttrName in oldStyle)

if (!newStyle[oldAttrName])

domElement.style[oldAttrName] = "";

for (let oldPropName in oldProps)

if (!newProps[oldPropName])

delete domElement[oldPropName];

for (let propName in newProps) {

if (propName == 'style') {

let styleObject = newProps[propName];

for (let attr in styleObject)

domElement.style[attr] = styleObject[attr];//更新行内样式

} else {

domElement[propName] = newProps[propName];

}

}

}

## **8.有key的其它情况**

* ABCD=>EBADF
* 插入E=>把B插入到A前面=>直接跳到 C=>插入F=>删除老节点剩下的C

### **8.1 src\index.js**

src\index.js

**import** { vnode, h, mount, patch } **from** './vdom';**const** root = document.getElementById('root');/\*\*

1. 前面后面添加元素 `ABCD=>ABCDEF` `ABCD=>EFABCD`

2. 前面后面删除元素 `ABCD=>ABC` `ABCD=>BCD`

\*/**const** oldVnode = h('ul', { id: 'container' },

h('li', { style: { backgroundColor: '#110000' }, key: 'A' }, 'A'),

h('li', { style: { backgroundColor: '#440000' }, key: 'B' }, 'B'),

h('li', { style: { backgroundColor: '#770000' }, key: 'C' }, 'C'),

h('li', { style: { backgroundColor: '#AA0000' }, key: 'D' }, 'D')

);**const** newVnode = h('ul', { id: 'newContainer' },

h('li', { style: { backgroundColor: '#EE0000' }, key: 'E' }, 'E1'),

h('li', { style: { backgroundColor: '#440000' }, key: 'B' }, 'B1'),

h('li', { style: { backgroundColor: '#110000' }, key: 'A' }, 'A1'),

h('li', { style: { backgroundColor: '#AA0000' }, key: 'D' }, 'D1'),

h('li', { style: { backgroundColor: '#FF0000' }, key: 'F' }, 'F1'),

);

mount(oldVnode, root);

setTimeout(() => {

patch(oldVnode, newVnode);

}, 1000)/\*\*

插入E

把B插入到A前面

直接跳到 C

插入F

删除老节点剩下的C

\*/

### **8.2 vdom\patch.js**

src\vdom\patch.js

import vnode, { isVnode, isSameVnode } from './vnode';

//把虚拟DOM节点封装成一个真实DOM节点

function createDOMElementFromVnode(vnode) {

let children = vnode.children;

let type = vnode.type;

let props = vnode.props;

if (type) {

let domElement = vnode.domElement = document.createElement(type);

updateProperties(vnode);

if (Array.isArray(children)) {

children.forEach(childVnode => domElement.appendChild(createDOMElementFromVnode(childVnode)));

}

} else {

vnode.domElement = document.createTextNode(vnode.text);

}

return vnode.domElement;

}

export function mount(vnode, root) {

let newDOMElement = createDOMElementFromVnode(vnode);

root.appendChild(newDOMElement);

}

export function patch(oldVnode, newVnode) {

if (oldVnode.type != newVnode.type) {

return oldVnode.domElement.parentNode.replaceChild(createDOMElementFromVnode(newVnode), oldVnode.domElement);

}

//如果新节点是文本节点，那么直接修改文本内容

if (typeof newVnode.text !== 'undefined') {

return oldVnode.domElement.textContent = newVnode.text;

}

let domElement = newVnode.domElement = oldVnode.domElement;

updateProperties(newVnode, oldVnode.props);

let oldChildren = oldVnode.children;

let newChildren = newVnode.children;

if (oldChildren.length > 0 && newChildren.length > 0) {

updateChildren(domElement, oldChildren, newChildren);

} else if (oldChildren.length > 0) {//老的有儿子,新的没儿子

for (let i = 0; i < oldChildren.length; i++) {

oldVnode.domElement.innerHTML = '';

}

} else if (newChildren.length > 0) {//新的有儿子,老的没儿子

for (let i = 0; i < newChildren.length; i++) {

oldVnode.domElement.appendChild(createDOMElementFromVnode(newChildren[i]));

}

}

}+function createKeyToIndexMap(children) {+ let map = {};+ for (let i = 0; i < children.length; i++) {+ let key = children[i].key;+ if (key) map[key] = i;+ }+ return map;+}

function updateChildren(parentDOMElement, oldChildren, newChildren) {

let oldStartIndex = 0, oldStartVnode = oldChildren[0];

let oldEndIndex = oldChildren.length - 1, oldEndVnode = oldChildren[oldEndIndex];

let newStartIndex = 0, newStartVnode = newChildren[0];

let newEndIndex = newChildren.length - 1, newEndVnode = newChildren[newEndIndex];+ let oldKeyToIndexMap = createKeyToIndexMap(oldChildren);

while (oldStartIndex <= oldEndIndex && newStartIndex <= newEndIndex) {+ if (!oldStartVnode) {+ oldStartVnode = oldChildren[++oldStartIndex];+ oldStartVnode.key && console.log(`直接跳到`, oldStartVnode.key);+ } else if (!oldEndVnode) {+ oldEndVnode = oldChildren[--oldEndIndex];+ oldEndVnode.key && console.log(`直接跳到`, oldEndVnode.key);

} else if (isSameVnode(oldStartVnode, newStartVnode)) {//顺序不变，顺序遍历完成

patch(oldStartVnode, newStartVnode);

oldStartVnode = oldChildren[++oldStartIndex];

newStartVnode = newChildren[++newStartIndex];

} else if (isSameVnode(oldEndVnode, newEndVnode)) {

patch(oldEndVnode, newEndVnode);

oldEndVnode = oldChildren[--oldEndIndex];

newEndVnode = newChildren[--newEndIndex];

} else if (isSameVnode(oldStartVnode, newEndVnode)) {

patch(oldStartVnode, newEndVnode);

parentDOMElement.insertBefore(oldStartVnode.domElement, oldEndVnode.domElement.nextSibling);

oldStartVnode = oldChildren[++oldStartIndex];

newEndVnode = newChildren[--newEndIndex];

} else if (isSameVnode(oldEndVnode, newStartVnode)) {

patch(oldEndVnode, newStartVnode);

parentDOMElement.insertBefore(oldEndVnode.domElement, oldStartVnode.domElement);

oldEndVnode = oldChildren[--oldEndIndex];

newStartVnode = newChildren[++newStartIndex];+ } else {// ABCD=>EBADF+ let oldIndexByKey = oldKeyToIndexMap[newStartVnode.key];+ if (oldIndexByKey == null) {+ console.log(`插入${newStartVnode.key}`);+ parentDOMElement.insertBefore(createDOMElementFromVnode(newStartVnode), oldStartVnode.domElement);+ newStartVnode = newChildren[++newStartIndex];+ } else {+ let oldVnodeToMove = oldChildren[oldIndexByKey];+ if (oldVnodeToMove.type !== newStartVnode.type) {+ console.log(`key一样,type不一样，删除重建`);+ parentDOMElement.insertBefore(createDOMElementFromVnode(newStartVnode), oldStartVnode.domElement);+ } else {+ patch(oldVnodeToMove, newStartVnode);+ oldChildren[oldIndexByKey] = undefined;//设置为undefined,后面遇到了绕过去+ console.log(`把老${oldVnodeToMove.key}插入到${oldStartVnode.key}前面`, oldStartVnode);+ parentDOMElement.insertBefore(oldVnodeToMove.domElement, oldStartVnode.domElement);+ }+ newStartVnode = newChildren[++newStartIndex];+ }+ }

}

if (newStartIndex <= newEndIndex) {//把没处理的新节点处理完

let beforeDOMElement = newChildren[newEndIndex + 1] == null ? null : newChildren[newEndIndex + 1].domElement;

for (let i = newStartIndex; i <= newEndIndex; i++) {+ console.log(`插入新节点剩下的${newChildren[i].key}`);

parentDOMElement.insertBefore(createDOMElementFromVnode(newChildren[i]), beforeDOMElement);

}

}

if (oldStartIndex <= oldEndIndex) {

for (let i = oldStartIndex; i <= oldEndIndex; i++) {

if (oldChildren[i]) {+ console.log(`删除老节点剩下的${oldChildren[i].key}`);

parentDOMElement.removeChild(oldChildren[i].domElement);

}

}

}

}

function updateProperties(vnode, oldProps = {}) {

let domElement = vnode.domElement;

let newProps = vnode.props;

let oldStyle = oldProps.style || {};

let newStyle = newProps.style || {};

for (let oldAttrName in oldStyle)

if (!newStyle[oldAttrName])

domElement.style[oldAttrName] = "";

for (let oldPropName in oldProps)

if (!newProps[oldPropName])

delete domElement[oldPropName];

for (let propName in newProps) {

if (propName == 'style') {

let styleObject = newProps[propName];

for (let attr in styleObject)

domElement.style[attr] = styleObject[attr];//更新行内样式

} else {

domElement[propName] = newProps[propName];

}

}

}

## **9.key可有可无**

src\index.js

**import** { vnode, h, mount, patch } **from** './vdom';**const** root = document.getElementById('root');**const** oldVnode = h('ul', { id: 'container' },

h('li', { style: { backgroundColor: '#110000' }, id: 'A', key: 'A' }, 'A'),

h('li', { style: { backgroundColor: '#440000' }, id: 'B' }, 'B'),

h('li', { style: { backgroundColor: '#770000' }, id: 'C', key: 'C' }, 'C'),

h('li', { style: { backgroundColor: '#AA0000' }, id: 'D' }, 'D')

);**const** newVnode = h('ul', { id: 'newContainer' },

h('div', { style: { backgroundColor: '#440000' }, id: 'B' }, 'B'),

h('span', { style: { backgroundColor: '#110000' }, id: 'A', key: 'A' }, 'A'),

h('div', { style: { backgroundColor: '#770000' }, id: 'Y', key: 'Y' }, 'Y'),

h('div', { style: { backgroundColor: '#AA0000' }, id: 'X' }, 'X'),

h('div', { style: { backgroundColor: '#CC0000' }, id: 'E' }, 'E'),

h('li', { style: { backgroundColor: '#770000' }, id: 'C', key: 'C' }, 'C'),

);

mount(oldVnode, root);

setTimeout(() => {

patch(oldVnode, newVnode);

}, 1000)

## **10.默认key**

src\index.js

**import** { vnode, h, mount, patch } **from** './vdom';**const** root = document.getElementById('root');**const** oldVnode = h('ul', { id: 'container' },

h('li', { style: { backgroundColor: '#110000' }, id: 'A' }, 'A'),

h('li', { style: { backgroundColor: '#440000' }, id: 'B' }, 'B'),

h('li', { style: { backgroundColor: '#770000' }, id: 'C' }, 'C'),

h('li', { style: { backgroundColor: '#AA0000' }, id: 'D' }, 'D')

);**const** newVnode = h('ul', { id: 'newContainer' },

h('li', { style: { backgroundColor: '#110000' }, id: 'A' }, 'A1'),

h('li', { style: { backgroundColor: '#440000' }, id: 'B' }, 'B1'),

h('li', { style: { backgroundColor: '#770000' }, id: 'C' }, 'C1'),

h('li', { style: { backgroundColor: '#AA0000' }, id: 'D' }, 'D1')

);

mount(oldVnode, root);

setTimeout(() => {

patch(oldVnode, newVnode);

}, 1000)