---

link: null
title: 珠峰架构师成长计划
description: JSX
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=285 sentences=2163, words=14963

---

1. 什么是React

- React 是一个用于构建用户界面的JavaScript库 核心专注于视图,目的实现组件化开发

2.创建项目

```
create-react-app zhufengreact
cd zhufengreact
yarn add cross-env
```

3.JSX渲染

3.1 什么是JSX

- 是一种JS和HTML混合的语法,将组件的结构、数据甚至样式都聚合在一起的写法

3.2 什么是元素

- JSX其实只是一种语法糖,最终会通过babeljs (https://www.babeljs.cn/repl)转译成 `React.createElement`语法
- `React.createElement`会返回一个React元素
- React元素事实上是普通的JS对象，用来描述你在屏幕上看到的内容
- `ReactDOM`来确保浏览器中的真实DOM数据和React元素保持一致

JSX

```
hello
```

转译后的代码

```
React.createElement("h1", {
  className: "title",
  style: {
    color: 'red'
  }
}, "hello");
```

返回的结果

```
{
  type:'h1',
  props:{
    className: "title",
    style: {
      color: 'red'
    }
  },
  children:"hello"
}
```

3.3 JSX实现

3.3.1 package.json

```
{
  "name": "zhufengreact",
  "version": "0.1.0",
  "scripts": {
+   "start": "cross-env DISABLE_NEW_JSX_TRANSFORM=true react-scripts start",
+   "build": "cross-env DISABLE_NEW_JSX_TRANSFORM=true react-scripts build",
+   "test": "cross-env DISABLE_NEW_JSX_TRANSFORM=true react-scripts test",
+   "eject": "cross-env DISABLE_NEW_JSX_TRANSFORM=true react-scripts eject"
  },
}
```

3.2 src\index.js

src\index.js

```
import React from "./react";
import ReactDOM from "./react-dom";
let element1 = (
  <div className="title" style={{ color: "red" }}>
    <span>hellospan>world
  div>
);
console.log(JSON.stringify(element1, null, 2));
ReactDOM.render(element1, document.getElementById("root"));
```

3.3 constants.js

src\constants.js

```
export const REACT_TEXT = Symbol('REACT_TEXT');
export const REACT_ELEMENT = Symbol('react.element');
```

3.4 src\utils.js

src\utils.js

```
import { REACT_TEXT } from "./constants";
export function wrapToVdom(element) {
  return typeof element === "string" || typeof element === "number"
    ? { type: REACT_TEXT, props: element }
    : element;
}
```

3.5 react.js

src\react.js

```
import { wrapToVdom } from "./utils";
import { REACT_ELEMENT } from "./constants";
function createElement(type, config, children) {
  let ref;
  let key;
  if (config) {
    delete config.__source;
    delete config.__self;
    ref = config.ref;
    delete config.ref;
    key = config.key;
    delete config.key;
  }
  let props = { ...config };
  if (arguments.length > 3) {
    props.children = Array.prototype.slice.call(arguments, 2).map(wrapToVdom);
  } else {
    props.children = wrapToVdom(children);
  }
  return {
    $typeof: REACT_ELEMENT,
    type,
    ref,
    key,
    props,
  };
}
const React = {
  createElement,
};
export default React;
```

3.6 react-dom.js

src\react-dom.js

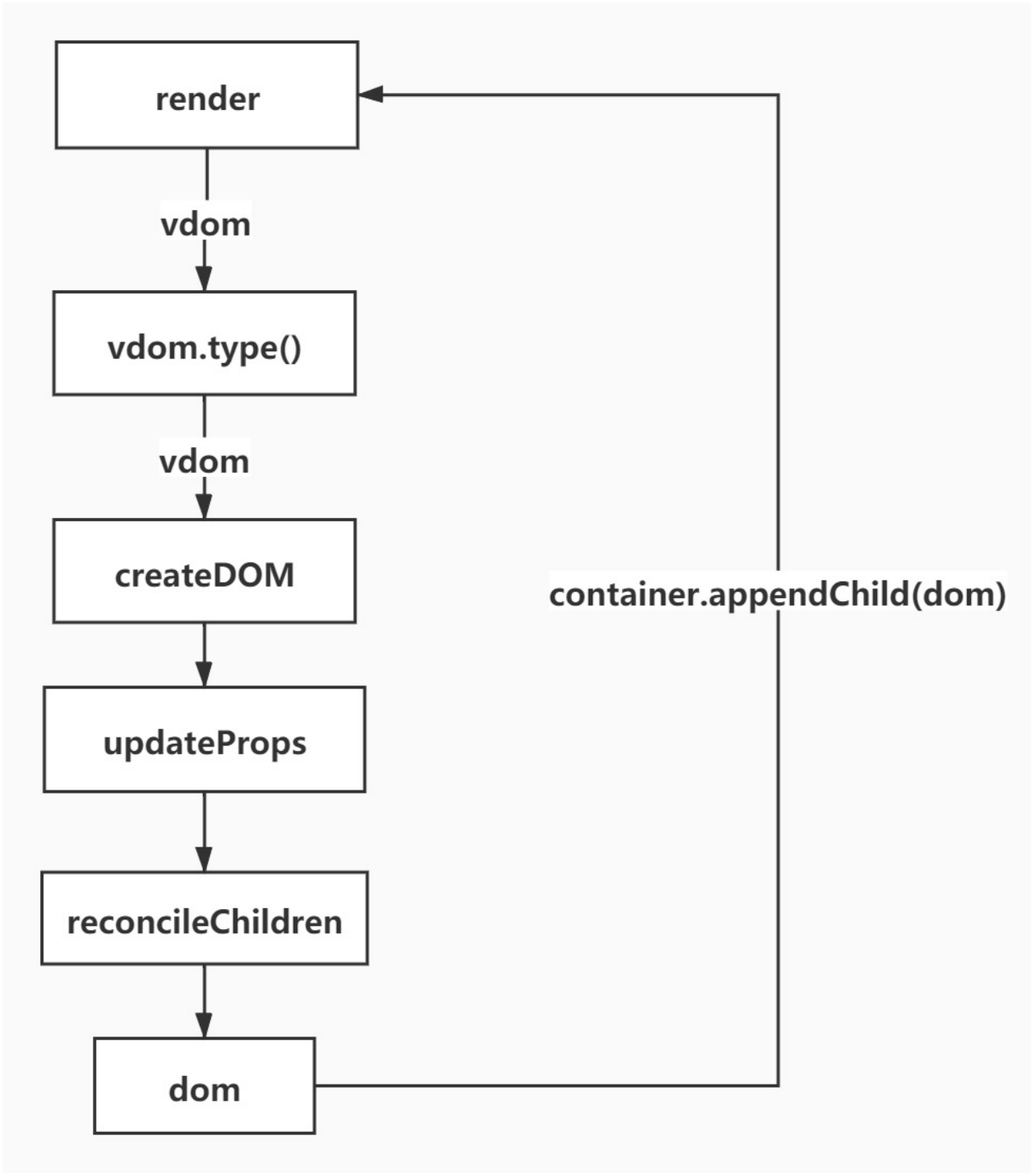
```
import { REACT_TEXT } from "./constants";
function render(vdom, container) {
  mount(vdom, container);
}
export function mount(vdom, container) {
  let newDOM = createDOM(vdom);
  container.appendChild(newDOM);
}
export function createDOM(vdom) {
  let { type, props } = vdom;
  let dom;
  if (type === REACT_TEXT) {
    dom = document.createTextNode(props);
  } else {
    dom = document.createElement(type);
  }
  if (props) {
    updateProps(dom, {}, props);
    if (typeof props.children == "object" && props.children.type) {
      mount(props.children, dom);
    } else if (Array.isArray(props.children)) {
      reconcileChildren(props.children, dom);
    }
  }
  vdom.dom = dom;
  return dom;
}
function updateProps(dom, oldProps={}, newProps={}) {
    for (let key in newProps) {
        if (key === 'children') {
            continue;
        } else if (key === 'style') {
            let styleObj = newProps[key];
            for (let attr in styleObj) {
                dom.style[attr] = styleObj[attr];
            }
        }else {
            dom[key] = newProps[key];
        }
    }
    for(let key in oldProps){
        if(!newProps.hasOwnProperty(key)){
            dom[key] = null;
        }
    }
}
function reconcileChildren(childrenVdom, parentDOM) {
  for (let i = 0; i < childrenVdom.length; i++) {
    mount(childrenVdom[i], parentDOM);
  }
}
const ReactDOM = {
  render,
};
export default ReactDOM;
```

4.组件

- 可以将UI切分成一些独立的、可复用的组件，这样你就只需专注于构建每一个单独的部件
- 组件从概念上类似于 JavaScript 函数。它接受任意的入参(props属性)，并返回用于描述页面展示内容的 React 元素

4.1 函数(定义的)组件

- 函数组件接收一个单一的props对象并返回了一个React元素
- 组件名称必须以大写字母开头
- 组件必须在使用的时候定义或引用它
- 组件的返回值只能有一个根元素
- React元素不但可以是DOM标签，还可以是用户自定义的组件
- 当 React 元素为用户自定义组件时，它会将 JSX 所接收的属性（attributes）转换为单个对象传递给组件，这个对象被称之为 props



4.2 实现

4.2.1 src\index.js

```
import React from "./react";
import ReactDOM from "./react-dom";
+function FunctionComponent(props){
+  return {props.name}{props.children};
+}
+let element = world;
ReactDOM.render(element, document.getElementById("root"));
```

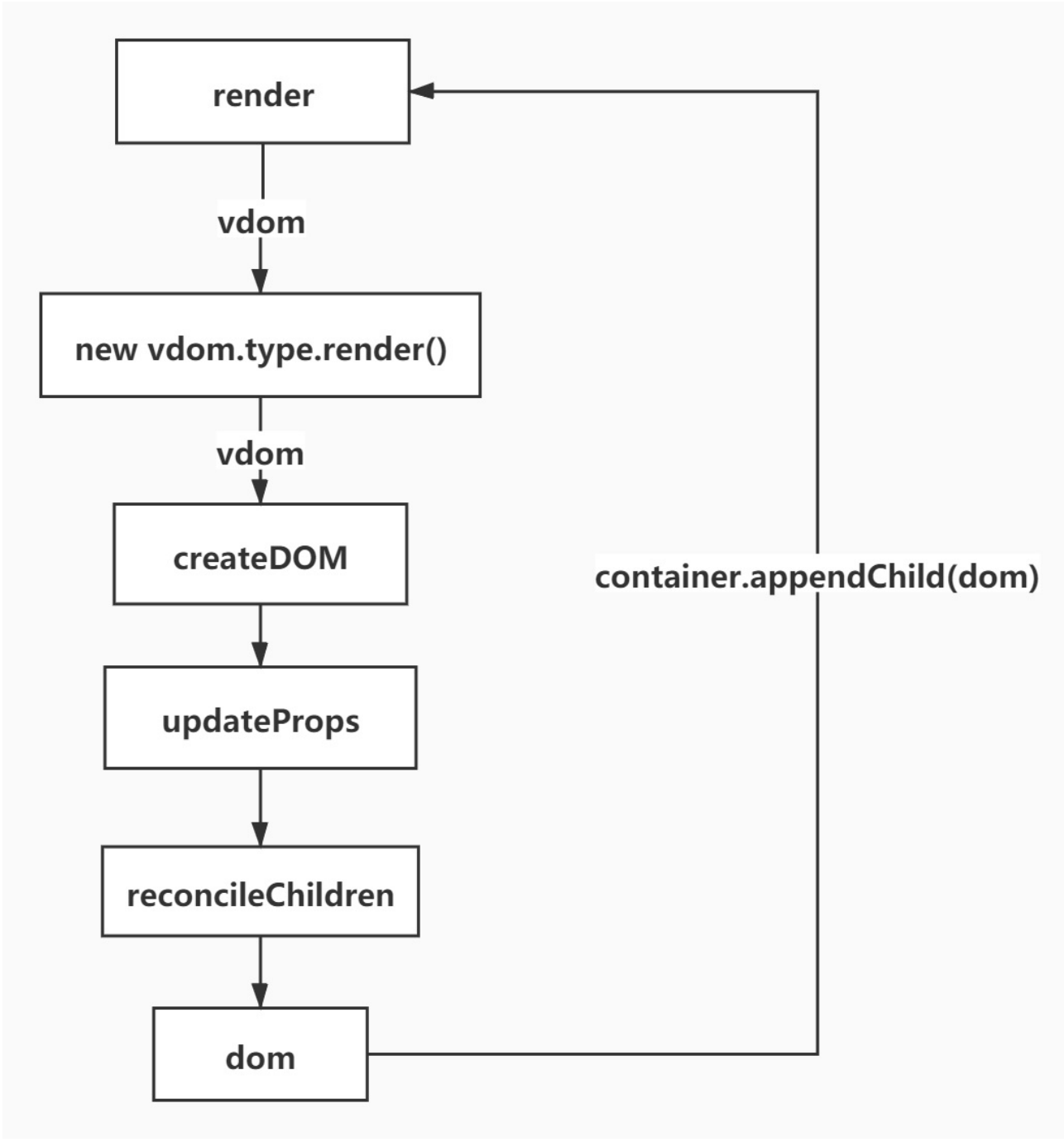
4.2.2 src\react-dom.js

src\react-dom.js

```
import { REACT_TEXT } from "./constants";
function render(vdom, parentDOM) {
    let newDOM = createDOM(vdom)
    if (newDOM) {
        parentDOM.appendChild(newDOM);
    }
}
export function createDOM(vdom) {
  let { type, props } = vdom;
  let dom;
  if (type
    dom = document.createTextNode(props);
+ } else if (typeof type === "function") {
+   return mountFunctionComponent(vdom);
+ } else {
    dom = document.createElement(type);
  }
  if (props) {
    updateProps(dom, {}, props);
    if (typeof props.children == "object" && props.children.type) {
      mount(props.children, dom);
    } else if (Array.isArray(props.children)) {
      reconcileChildren(props.children, dom);
    }
  }
  vdom.dom = dom;
  return dom;
}
+function mountFunctionComponent(vdom){
+    let {type,props}= vdom;
+    let renderVdom = type(props);
+    return createDOM(renderVdom);
+}
function updateProps(dom, oldProps={}, newProps={}) {
    for (let key in newProps) {
        if (key
            continue;
        } else if (key
            let styleObj = newProps[key];
            for (let attr in styleObj) {
                dom.style[attr] = styleObj[attr];
            }
        } else if (key.startsWith('on')) {
            addEvent(dom, key.toLocaleLowerCase(), newProps[key]);
        } else {
            dom[key] = newProps[key];
        }
    }
    for(let key in oldProps){
        if(!newProps.hasOwnProperty(key)){
            dom[key] = null;
        }
    }
}
function reconcileChildren(childrenVdom, parentDOM) {
  for (let i = 0; i < childrenVdom.length; i++) {
    mount(childrenVdom[i], parentDOM);
  }
}
const ReactDOM = {
  render,
};
export default ReactDOM;
```

4.2 类(定义的)组件

- 也可以通过类定义组件
- 类组件的渲染是根据属性创建类的实例，并调用实例的render方法返回一个React元素



4.2.1 src\index.js

src\index.js

```
import React from "./react";
import ReactDOM from "./react-dom";
+class ClassComponent extends React.Component{
+    render(){
+        return {this.props.name}{this.props.children};
+    }
+}
+let element = world;
ReactDOM.render(element, document.getElementById("root"));
```

4.2.2 src\Component.js

src\Component.js

```
export class Component{
    static isReactComponent=true
    constructor(props){
        this.props = props;
    }
}
```

4.2.3 src\react.js

src\react.js

```
import { wrapToVdom } from "./utils";
+import {Component} from './Component';
function createElement(type, config, children) {
  let ref;
  let key;
  if (config) {
    delete config.__source;
    delete config.__self;
    ref = config.ref;
    delete config.ref;
    key = config.key;
    delete config.key;
  }
  let props = { ...config };
  if (arguments.length > 3) {
    props.children = Array.prototype.slice.call(arguments, 2).map(wrapToVdom);
  } else {
    props.children = wrapToVdom(children);
  }
  return {
    type,
    ref,
    key,
    props,
  };
}
const React = {
  createElement,
+  Component
};
export default React;
```

4.2.4 src\react-dom.js

src\react-dom.js

```
import { REACT_TEXT } from "./constants";
function render(vdom, parentDOM) {
    let newDOM = createDOM(vdom)
    if (newDOM) {
        parentDOM.appendChild(newDOM);
    }
}
export function createDOM(vdom) {
  let { type, props } = vdom;
  let dom;
  if (type
    dom = document.createTextNode(props);
  } else if (typeof type
+    if (type.isReactComponent) {
+      return mountClassComponent(vdom);
+    } else {
      return mountFunctionComponent(vdom);
    }
  } else {
    dom = document.createElement(type);
  }
  if (props) {
    updateProps(dom, {}, props);
    if (typeof props.children == "object" && props.children.type) {
      mount(props.children, dom);
    } else if (Array.isArray(props.children)) {
      reconcileChildren(props.children, dom);
    }
  }
  vdom.dom = dom;
  return dom;
}
+function mountClassComponent(vdom){
+    let {type,props}= vdom;
+    let classInstance = new type(props);
+    let renderVdom = classInstance.render();
+    let dom =  createDOM(renderVdom);
+    return dom;
+}
function mountFunctionComponent(vdom) {
  let { type, props } = vdom;
  let renderVdom = type(props);
  return createDOM(renderVdom);
}
function updateProps(dom, oldProps={}, newProps={}) {
    for (let key in newProps) {
        if (key
            continue;
        } else if (key
            let styleObj = newProps[key];
            for (let attr in styleObj) {
                dom.style[attr] = styleObj[attr];
            }
        } else if (key.startsWith('on')) {
            addEvent(dom, key.toLocaleLowerCase(), newProps[key]);
        } else {
            dom[key] = newProps[key];
        }
    }
    for(let key in oldProps){
        if(!newProps.hasOwnProperty(key)){
            dom[key] = null;
        }
    }
}
function reconcileChildren(childrenVdom, parentDOM) {
  for (let i = 0; i < childrenVdom.length; i++) {
    mount(childrenVdom[i], parentDOM);
  }
}
const ReactDOM = {
  render,
};
export default ReactDOM;
```

5.类组件的更新

5.1 组件状态

- 组件的数据来源有两个地方，分别是属性对象和状态对象
- 属性是父组件传递过来的
- 状态是自己内部的,改变状态唯一的方式就是 `setState`
- 属性和状态的变化都会影响视图更新
- 不要直接修改 State，构造函数是唯一可以给 this.state 赋值的地方

5.3 更新组新实现

5.3.1 src\index.js

```
import React from "./react";
import ReactDOM from "./react-dom";
class Counter extends React.Component {
    constructor(props) {
        super(props);
        this.state = { number: 0 };
    }
    handleClick = () => {
        this.setState({ number: this.state.number + 1 });
        console.log(this.state);

    }
    render() {
        return (

                {this.props.title}
                number:{this.state.number}
                +

        )
    }
}
ReactDOM.render(, document.getElementById("root"));
```

5.3.2 src\Component.js

src\Component.js

```
+import { findDOM, compareTwoVdom } from './react-dom';
+class Updater {
+    constructor(classInstance) {
+        this.classInstance = classInstance;
+        this.pendingStates = [];
+        this.callbacks = [];
+    }
+    addState(partialState, callback) {
+        this.pendingStates.push(partialState);///等待更新的或者说等待生效的状态
+        if (typeof callback === 'function')
+            this.callbacks.push(callback);//状态更新后的回调
+        this.emitUpdate();
+    }
+    emitUpdate() {
+        this.updateComponent();
+    }
+    updateComponent() {
+        let { classInstance, pendingStates } = this;
+        if (pendingStates.length > 0) {
+            shouldUpdate(classInstance, this.getState());
+        }
+    }
+    getState() {
+        let { classInstance, pendingStates } = this;
+        let { state } = classInstance;
+        pendingStates.forEach((nextState) => {
+            if (typeof nextState === 'function') {
+                nextState = nextState(state);
+            }
+            state = { ...state, ...nextState };
+        });
+        pendingStates.length = 0;
+        return state;
+    }
+}
+function shouldUpdate(classInstance, nextState) {
+    classInstance.state = nextState;
+    classInstance.forceUpdate();
+}
export class Component {
    static isReactComponent = true;
    constructor(props) {
        this.props = props;
+        this.state = {};
+        this.updater = new Updater(this);
    }
+    setState(partialState, callback) {
+        this.updater.addState(partialState, callback);
+    }
+    forceUpdate() {
+        let oldRenderVdom = this.oldRenderVdom;
+        let oldDOM = findDOM(oldRenderVdom);
+        let newRenderVdom = this.render();
+        compareTwoVdom(oldDOM.parentNode, oldRenderVdom, newRenderVdom);
+        this.oldRenderVdom = newRenderVdom;
+    }
+}
```

5.3.3 react-dom.js

src\react-dom.js

```
import { REACT_TEXT } from "./constants";
function render(vdom, parentDOM) {
    let newDOM = createDOM(vdom)
    if (newDOM) {
        parentDOM.appendChild(newDOM);
    }
}
export function createDOM(vdom) {
  let { type, props } = vdom;
  let dom;
  if (type
    dom = document.createTextNode(props);
  } else if (typeof type
    if (type.isReactComponent) {
      return mountClassComponent(vdom);
    } else {
      return mountFunctionComponent(vdom);
    }
  } else {
    dom = document.createElement(type);
  }
  if (props) {
    updateProps(dom, {}, props);
    if (typeof props.children == "object" && props.children.type) {
      mount(props.children, dom);
    } else if (Array.isArray(props.children)) {
      reconcileChildren(props.children, dom);
    }
  }
  vdom.dom = dom;
  return dom;
}
function mountClassComponent(vdom) {
  let { type, props } = vdom;
  let classInstance = new type(props);
  let renderVdom = classInstance.render();
+ classInstance.oldRenderVdom = renderVdom;
  let dom = createDOM(renderVdom);
  return dom;
}
function mountFunctionComponent(vdom) {
  let { type, props } = vdom;
  let renderVdom = type(props);
+ vdom.oldRenderVdom = renderVdom;
  return createDOM(renderVdom);
}
function updateProps(dom, oldProps={}, newProps={}) {
    for (let key in newProps) {
        if (key
            continue;
        } else if (key
            let styleObj = newProps[key];
            for (let attr in styleObj) {
                dom.style[attr] = styleObj[attr];
            }
+       } else if (/^on[A-Z].*/.test(key)) {
+           dom[key.toLowerCase()]=newProps[key];
+       } else {
            dom[key] = newProps[key];
        }
    }
    for(let key in oldProps){
        if(!newProps.hasOwnProperty(key)){
            dom[key] = null;
        }
    }
}
+export function findDOM(vdom) {
+    if (!vdom) return null;
+    if (vdom.dom) {
+        return vdom.dom;
+    } else {
+        let renderVdom = vdom.oldRenderVdom;
+        return findDOM(renderVdom);
+    }
+}

+export function compareTwoVdom(parentDOM, oldVdom, newVdom) {
+    let oldDOM = findDOM(oldVdom);
+    let newDOM = createDOM(newVdom);
+    parentDOM.replaceChild(newDOM, oldDOM);
+}
function reconcileChildren(childrenVdom, parentDOM) {
  for (let i = 0; i < childrenVdom.length; i++) {
    mount(childrenVdom[i], parentDOM);
  }
}
const ReactDOM = {
  render,
};
export default ReactDOM;
```

6.合成事件和批量更新

- State 的更新会被合并 当你调用 setState() 的时候，React 会把你提供的对象合并到当前的 state
- State 的更新可能是异步的
    - 出于性能考虑，React 可能会把多个 setState() 调用合并成一个调用
    - 因为 this.props 和 this.state 可能会异步更新，所以你不要依赖他们的值来更新下一个状态
    - 可以让 setState() 接收一个函数而不是一个对象。这个函数用上一个 state 作为第一个参数
- 事件处理
    - React 事件的命名采用小驼峰式(camelCase),而不是纯小写

- 使用 JSX 语法时你需要传入一个函数作为事件处理函数，而不是一个字符串
- 你不能通过返回 `false` 的方式阻止默认行为。你必须显式的使用 `preventDefault`

6.1 src\index.js

src\index.js

```
import React from "./react";
import ReactDOM from "./react-dom";
class Counter extends React.Component {
    constructor(props) {
        super(props);
        this.state = { number: 0 };
    }
    handleClick = () => {
+       this.setState({ number: this.state.number + 1 });
+       console.log(this.state);
+       this.setState({ number: this.state.number + 1 });
+       console.log(this.state);
+       setTimeout(()=>{
+           this.setState({ number: this.state.number + 1 });
+           console.log(this.state);
+           this.setState({ number: this.state.number + 1 });
+           console.log(this.state);
+       });
    }
    render() {
        return (

                {this.props.title}
                number:{this.state.number}
                +

        )
    }
}
ReactDOM.render(, document.getElementById("root"));
```

6.2 src\utils.js

src\utils.js

```
import { REACT_TEXT } from "./constants";
export function wrapToVdom(element) {
  return typeof element
    ? { type: REACT_TEXT, props: { content: element } }
    : element;
}

+export function isFunction(obj) {
+    return typeof obj === 'function';
+}
```

6.3 src\Component.js

src\Component.js

```
import { findDOM, compareTwoVdom } from './react-dom';
+export let updateQueue = {
+    isBatchingUpdate:false,
+    updaters:new Set(),
+    batchUpdate(){//批量更新
+      updateQueue.isBatchingUpdate = false;
+      for (var updater of updateQueue.updaters){
+          updater.updateComponent();
+      }
+      updateQueue.updaters.clear();
+    }
+}
class Updater {
    constructor(classInstance) {
        this.classInstance = classInstance;
        this.pendingStates = [];
        this.callbacks = [];
    }
    addState(partialState, callback) {
        this.pendingStates.push(partialState);///等待更新的或者说等待生效的状态
        if (typeof callback
            this.callbacks.push(callback);//状态更新后的回调
        this.emitUpdate();
    }
+   emitUpdate(nextProps) {
+       this.nextProps = nextProps;
+       if(updateQueue.isBatchingUpdate){
+           updateQueue.updaters.add(this);
+       }else{
+           this.updateComponent();
+       }
+   }
    updateComponent() {
        let { classInstance, pendingStates } = this;
+       if (this.nextProps || pendingStates.length > 0) {
+           shouldUpdate(classInstance,this.nextProps, this.getState());
+       }
    }
    getState() {
        let { classInstance, pendingStates } = this;
        let { state } = classInstance;
        pendingStates.forEach((nextState) => {
            if (typeof nextState
                nextState = nextState(state);
            }
            state = { ...state, ...nextState };
        });
        pendingStates.length = 0;
        return state;
    }
}
+function shouldUpdate(classInstance,nextProps, nextState) {
+   if(nextProps) classInstance.props = nextProps;
    classInstance.state = nextState;
    classInstance.forceUpdate();
}
export class Component {
    static isReactComponent = true;
    constructor(props) {
        this.props = props;
        this.state = {};
        this.updater = new Updater(this);
    }
    setState(partialState, callback) {
        this.updater.addState(partialState, callback);
    }
    forceUpdate() {
        let oldRenderVdom = this.oldRenderVdom;
        let oldDOM = findDOM(oldRenderVdom);
        let newRenderVdom = this.render();
        compareTwoVdom(oldDOM.parentNode, oldRenderVdom, newRenderVdom);
        this.oldRenderVdom = newRenderVdom;
    }
}
```

6.4 src\event.js

src\event.js

```
import { updateQueue } from './Component';
export function addEvent(dom, eventType, handler) {
    let store = dom.store||(dom.store ={})
    store[eventType] = handler;
    if (!document[eventType]) {
        document[eventType] = dispatchEvent;
    }
}

function dispatchEvent(event) {
  let { target, type } = event;
  let eventType = `on${type}`;
  let syntheticEvent = createSyntheticEvent(event);
  updateQueue.isBatchingUpdate = true;
  while (target) {
    let { store } = target;
    let handler = store && store[eventType]
    handler && handler(syntheticEvent);

    if (syntheticEvent.isPropagationStopped) {
      break;
    }
    target = target.parentNode;
  }
  updateQueue.batchUpdate();
}
function createSyntheticEvent(nativeEvent) {
  let syntheticEvent = {};
  for (let key in nativeEvent) {
    let value = nativeEvent[key];
    if(typeof value === 'function')value=value.bind(nativeEvent);
    syntheticEvent[key] = nativeEvent[key];
  }
  syntheticEvent.nativeEvent = nativeEvent;
  syntheticEvent.isDefaultPrevented = false;
  syntheticEvent.isPropagationStopped = false;
  syntheticEvent.preventDefault = preventDefault;
  syntheticEvent.stopPropagation = stopPropagation;
  return syntheticEvent;
}

function preventDefault() {
  this.defaultPrevented = true;
  const event = this.nativeEvent;
  if (event.preventDefault) {
    event.preventDefault();
  } else {
    event.returnValue = false;
  }
  this.isDefaultPrevented = true;
}

function stopPropagation() {
  const event = this.nativeEvent;
  if (event.stopPropagation) {
    event.stopPropagation();
  } else {
    event.cancelBubble = true;
  }
  this.isPropagationStopped = true;
}
```

6.5 src\react-dom.js

src\react-dom.js

```
import { REACT_TEXT } from "./constants";
+import { addEvent } from './event';
function render(vdom, parentDOM) {
    let newDOM = createDOM(vdom)
    if (newDOM) {
        parentDOM.appendChild(newDOM);
    }
}
export function createDOM(vdom) {
  let { type, props } = vdom;
  let dom;
  if (type
    dom = document.createTextNode(props);
  } else if (typeof type
    if (type.isReactComponent) {
      return mountClassComponent(vdom);
    } else {
      return mountFunctionComponent(vdom);
    }
  } else {
    dom = document.createElement(type);
  }
  if (props) {
    updateProps(dom, {}, props);
    if (typeof props.children == "object" && props.children.type) {
      mount(props.children, dom);
    } else if (Array.isArray(props.children)) {
      reconcileChildren(props.children, dom);
    }
  }
  vdom.dom = dom;
  return dom;
}
function mountClassComponent(vdom) {
  let { type, props } = vdom;
  let classInstance = new type(props);
  let renderVdom = classInstance.render();
  classInstance.oldRenderVdom = renderVdom;
  let dom = createDOM(renderVdom);
  return dom;
}
function mountFunctionComponent(vdom) {
  let { type, props } = vdom;
  let renderVdom = type(props);
  vdom.oldRenderVdom = renderVdom;
  return createDOM(renderVdom);
}
function updateProps(dom, oldProps={}, newProps={}) {
    for (let key in newProps) {
        if (key
            continue;
        } else if (key
            let styleObj = newProps[key];
            for (let attr in styleObj) {
                dom.style[attr] = styleObj[attr];
            }
+        } else if (key.startsWith('on')) {
+            addEvent(dom, key.toLocaleLowerCase(), newProps[key]);
        } else {
            dom[key] = newProps[key];
        }
    }
    for(let key in oldProps){
        if(!newProps.hasOwnProperty(key)){
            dom[key] = null;
        }
    }
}
export function findDOM(vdom){
    let {type}= vdom;
    let dom;
    if(typeof type
        dom=findDOM(vdom.oldRenderVdom);
    }else{
        dom=vdom.dom;
    }
    return dom;
}
export function compareTwoVdom(parentDOM, oldVdom, newVdom) {
    let oldDOM = findDOM(oldVdom);
    let newDOM = createDOM(newVdom);
    parentDOM.replaceChild(newDOM, oldDOM);
}
function reconcileChildren(childrenVdom, parentDOM) {
  for (let i = 0; i < childrenVdom.length; i++) {
    mount(childrenVdom[i], parentDOM);
  }
}
const ReactDOM = {
  render,
};
export default ReactDOM;
```

7.ref

- Refs 提供了一种方式，允许我们访问 DOM 节点或在 render 方法中创建的 React 元素

7.1 为 DOM 元素添加 ref

- 可以使用 ref 去存储 DOM 节点的引用
- 当 ref 属性用于 HTML 元素时，构造函数中使用 React.createRef() 创建的 ref 接收底层 DOM 元素作为其 current 属性

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
class Sum extends React.Component {
    a
    b
    result
    constructor(props) {
        super(props);
        this.a = React.createRef();
        this.b = React.createRef();
        this.result = React.createRef();
    }
    handleAdd = () => {
        let a = this.a.current.value;
        let b = this.b.current.value;
        this.result.current.value = a + b;
    }
    render() {
        return (
            <>
                <input ref={this.a} />+<input ref={this.b} /><button onClick={this.handleAdd}>=button><input ref={this.result} />
            </>
        );
    }
}
ReactDOM.render(
    <Sum />,
    document.getElementById('root')
);
```

7.2 为 class 组件添加 Ref

- 当 ref 属性用于自定义 class 组件时，ref 对象接收组件的挂载实例作为其 current 属性

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
class Form extends React.Component {
    input
    constructor(props) {
        super(props);
        this.input = React.createRef();
    }
    getFocus = () => {
        this.input.current.getFocus();
    }
    render() {
        return (
            <>
                <TextInput ref={this.input} />
                <button onClick={this.getFocus}>获得焦点button>
            </>
        );
    }
}
class TextInput extends React.Component {
    input
    constructor(props) {
        super(props);
        this.input = React.createRef();
    }
    getFocus = () => {
        this.input.current.focus();
    }
    render() {
        return <input ref={this.input} />
    }
}
ReactDOM.render(
    <Form />,
    document.getElementById('root')
);
```

7.3 Ref转发

- 你不能在函数组件上使用 ref 属性，因为他们没有实例
- Ref 转发是一项将 ref 自动地通过组件传递到其一子组件的技巧
- Ref 转发允许某些组件接收 ref，并将其向下传递给子组件

src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
interface InputProps { }
const TextInput = React.forwardRef((props, ref) => (
    <input ref={ref} />
));
class Form extends React.Component {
    input
    constructor(props) {
        super(props);
        this.input = React.createRef();
    }
    getFocus = () => {
        console.log(this.input.current);

        this.input.current.focus();
    }
    render() {
        return (
            <>
                <TextInput ref={this.input} />
                <button onClick={this.getFocus}>获得焦点button>
            </>
        );
    }
}

ReactDOM.render(
    <Form />,
    document.getElementById('root')
);
```

7.4 ref实现

7.4.1 src\constants.js

```
export const REACT_TEXT = Symbol('REACT_TEXT');
+export const REACT_FORWARD_REF_TYPE = Symbol('react.forward_ref');
```

7.4.2 src\react.js

src\react.js

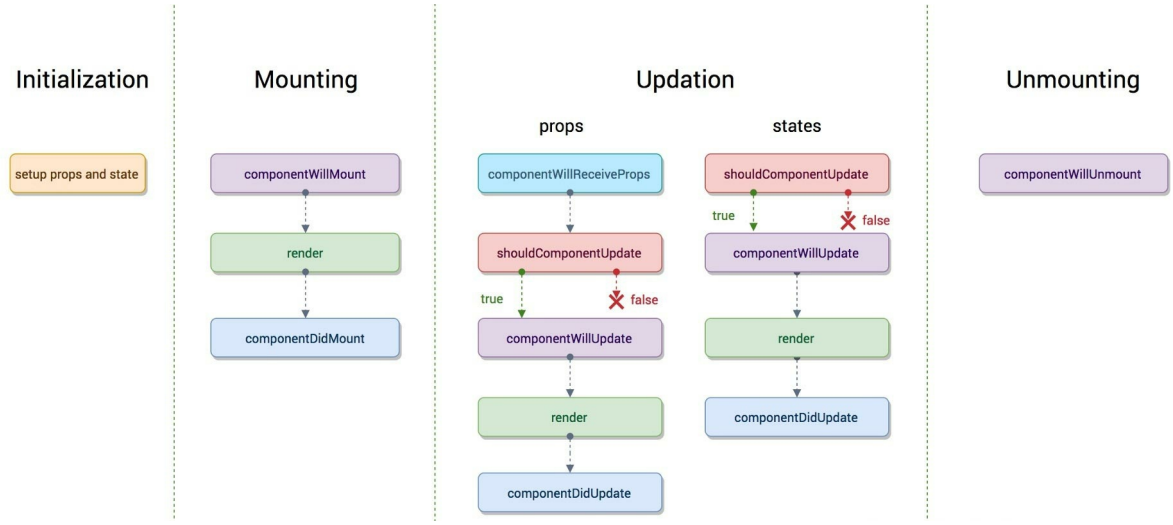
```
import { wrapToVdom } from "./utils";
import {Component} from './Component';
function createElement(type, config, children) {
  let ref;
  let key;
  if (config) {
    delete config.__source;
    delete config.__self;
    ref = config.ref;
    delete config.ref;
    key = config.key;
    delete config.key;
  }
  let props = { ...config };
  if (arguments.length > 3) {
    props.children = Array.prototype.slice.call(arguments, 2).map(wrapToVdom);
  } else {
    props.children = wrapToVdom(children);
  }
  return {
    type,
    ref,
    key,
    props,
  };
}
+function createRef(){
+    return {current:null};
+}
+function forwardRef(render) {
+  var elementType = {
+    $typeof: REACT_FORWARD_REF_TYPE,
+    render: render
+  };
+  return elementType;
+}
const React = {
  createElement,
  Component,
+ createRef
};
export default React;
```

7.4.3 react-dom.js

src\react-dom.js

```
+import {REACT_TEXT,REACT_FORWARD_REF_TYPE} from './constants';
import { addEvent } from './event';
function render(vdom, parentDOM) {
    let newDOM = createDOM(vdom)
    if (newDOM) {
        parentDOM.appendChild(newDOM);
    }
}
export function createDOM(vdom) {
+ let { type, props,ref } = vdom;
  let dom;
+ if(type&&type.$typeof===REACT_FORWARD_REF_TYPE){
+   return mountForwardComponent(vdom);
  }else if (type
    dom = document.createTextNode(props);
  } else if (typeof type
    if (type.isReactComponent) {
      return mountClassComponent(vdom);
    } else {
      return mountFunctionComponent(vdom);
    }
  } else {
    dom = document.createElement(type);
  }
  if (props) {
    updateProps(dom, {}, props);
    if (typeof props.children == "object" && props.children.type) {
      mount(props.children, dom);
    } else if (Array.isArray(props.children)) {
      reconcileChildren(props.children, dom);
    }
  }
  vdom.dom = dom;
+ if(ref) ref.current = dom;
  return dom;
}
+function mountForwardComponent(vdom){
+    let {type,props,ref} = vdom;
+    let renderVdom = type.render(props,ref);
+    vdom.oldRenderVdom = renderVdom;
+    return createDOM(renderVdom);
+}
function mountClassComponent(vdom) {
+ let { type, props,ref } = vdom;
  let classInstance = new type(props);
+ if(ref) ref.current = classInstance;
  let renderVdom = classInstance.render();
  classInstance.oldRenderVdom = renderVdom;
  let dom = createDOM(renderVdom);
  return dom;
}
function mountFunctionComponent(vdom) {
  let { type, props } = vdom;
  let renderVdom = type(props);
  vdom.oldRenderVdom = renderVdom;
  return createDOM(renderVdom);
}
function updateProps(dom, oldProps={}, newProps={}) {
    for (let key in newProps) {
        if (key
            continue;
        } else if (key
            let styleObj = newProps[key];
            for (let attr in styleObj) {
                dom.style[attr] = styleObj[attr];
            }
        } else if (key.startsWith('on')) {
            addEvent(dom, key.toLocaleLowerCase(), newProps[key]);
        } else {
            dom[key] = newProps[key];
        }
    }
    for(let key in oldProps){
        if(!newProps.hasOwnProperty(key)){
            dom[key] = null;
        }
    }
}
export function findDOM(vdom){
    let {type}= vdom;
    let dom;
    if(typeof type
        dom=findDOM(vdom.oldRenderVdom);
    }else{
        dom=vdom.dom;
    }
    return dom;
}
export function compareTwoVdom(parentDOM, oldVdom, newVdom) {
    let oldDOM = findDOM(oldVdom);
    let newDOM = createDOM(newVdom);
    parentDOM.replaceChild(newDOM, oldDOM);
}
function reconcileChildren(childrenVdom, parentDOM) {
  for (let i = 0; i < childrenVdom.length; i++) {
    mount(childrenVdom[i], parentDOM);
  }
}
const ReactDOM = {
  render,
};
export default ReactDOM;
```

8.基本生命周期



8.1 src\index.js

src\index.js

```
import React from './react';
import ReactDOM from './react-dom';
class Counter extends React.Component{
    static defaultProps = {
        name: '珠峰架构'
    };
    constructor(props) {
        super(props);
        this.state = { number: 0 }
        console.log('Counter 1.constructor')
    }
    componentWillMount() {
        console.log('Counter 2.componentWillMount');
    }
    componentDidMount() {
        console.log('Counter 4.componentDidMount');
    }
    handleClick = () => {
        this.setState({ number: this.state.number + 1 });
    };

    shouldComponentUpdate(nextProps, nextState) {
        console.log('Counter 5.shouldComponentUpdate');
        return nextState.number % 2 === 0;

    }
    componentWillUpdate() {
        console.log('Counter 6.componentWillUpdate');
    }
    componentDidUpdate() {
        console.log('Counter 7.componentDidUpdate');
    }
    render() {
        console.log('Counter 3.render');
        return (
            <div>
                <p>{this.state.number}p>
                <button onClick={this.handleClick}>+button>
            div>
        )
    }
}
ReactDOM.render(<Counter />, document.getElementById('root'));
```

8.2 src\Component.js

src\Component.js

```
import { findDOM, compareTwoVdom } from './react-dom';
export let updateQueue = {
    isBatchingUpdate:false,
    updaters:[],
    batchUpdate(){//批量更新
    updateQueue.isBatchingUpdate = false;
      for(let updater of updateQueue.updaters){
        updater.updateComponent();
      }
      updateQueue.updaters.length=0;
    }
}
class Updater {
    constructor(classInstance) {
        this.classInstance = classInstance;
        this.pendingStates = [];
        this.callbacks = [];
    }
    addState(partialState, callback) {
        this.pendingStates.push(partialState);///等待更新的或者说等待生效的状态
        if (typeof callback
            this.callbacks.push(callback);//状态更新后的回调
        this.emitUpdate();
    }
    emitUpdate(nextProps) {
        this.nextProps = nextProps;
        if(updateQueue.isBatchingUpdate){
            updateQueue.updaters.push(this);
        }else{
            this.updateComponent();
        }
    }
    updateComponent() {
        let { classInstance, pendingStates } = this;
        if (this.nextProps || pendingStates.length > 0) {
            shouldUpdate(classInstance,this.nextProps, this.getState());
        }
    }
    getState() {
        let { classInstance, pendingStates } = this;
        let { state } = classInstance;
        pendingStates.forEach((nextState) => {
            if (typeof nextState
                nextState = nextState(state);
            }
            state = { ...state, ...nextState };
        });
        pendingStates.length = 0;
        return state;
    }
}
function shouldUpdate(classInstance,nextProps, nextState) {
+    let willUpdate = true;
+    if(classInstance.shouldComponentUpdate
+       &&!classInstance.shouldComponentUpdate(nextProps,nextState)){
+           willUpdate = false;
+    }
+    if(willUpdate && classInstance.componentWillUpdate){
+        classInstance.componentWillUpdate();
+    }
+    if(nextProps){
+        classInstance.props = nextProps;
+    }
+    classInstance.state = nextState;
+    if(willUpdate) classInstance.forceUpdate();
}
export class Component {
    static isReactComponent = true;
    constructor(props) {
        this.props = props;
        this.state = {};
        this.updater = new Updater(this);
    }
    setState(partialState, callback) {
        this.updater.addState(partialState, callback);
    }
    forceUpdate() {
        let oldRenderVdom = this.oldRenderVdom;
        let oldDOM = findDOM(oldRenderVdom);
        let newRenderVdom = this.render();
        compareTwoVdom(oldDOM.parentNode, oldRenderVdom, newRenderVdom);
        this.oldRenderVdom = newRenderVdom;
+       if(this.componentDidUpdate){
+           this.componentDidUpdate(this.props,this.state);
+       }
    }
}
```

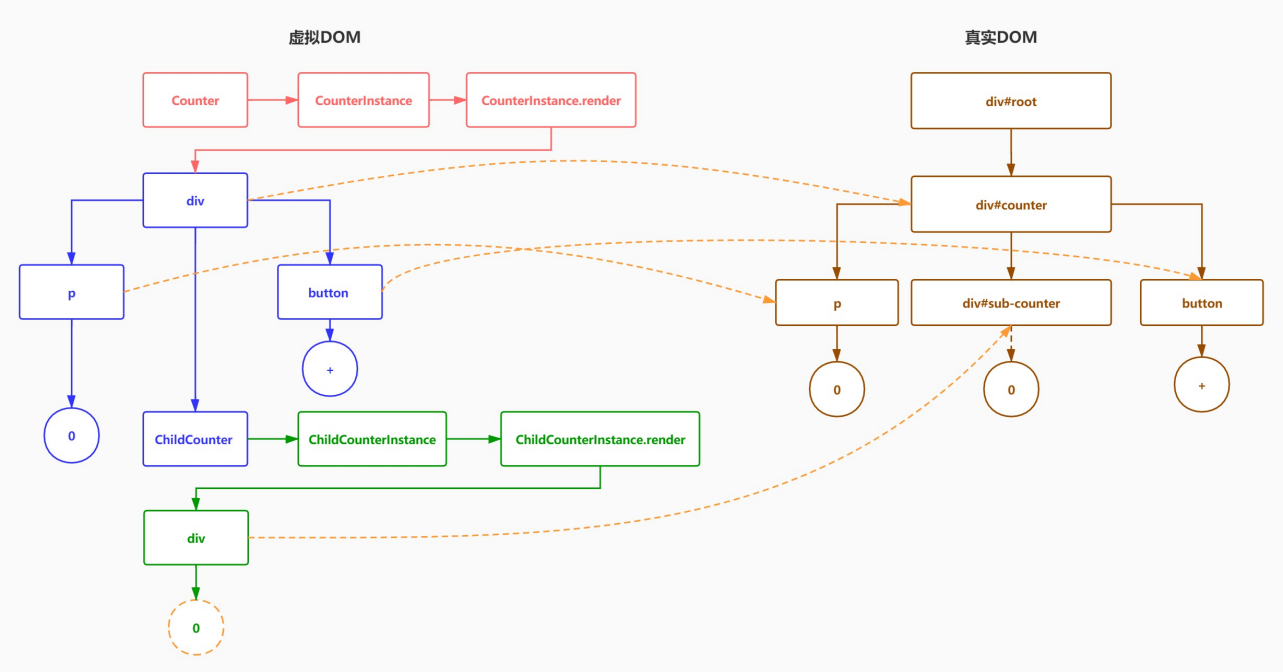
8.3 src\react-dom.js

src\react-dom.js

```
import {REACT_TEXT,REACT_FORWARD_REF_TYPE} from './constants';
import { addEvent } from './event';

function render(vdom, parentDOM) {
    let newDOM = createDOM(vdom)
    if (newDOM) {
        parentDOM.appendChild(newDOM);
+       if (newDOM._componentDidMount) newDOM._componentDidMount();
    }
}
export function createDOM(vdom) {
  let { type, props,ref } = vdom;
```

```
  let dom;
  if(type&&type.$typeof
    return mountForwardComponent(vdom);
  }else if (type
    dom = document.createTextNode(props);
  } else if (typeof type
    if (type.isReactComponent) {
      return mountClassComponent(vdom);
    } else {
      return mountFunctionComponent(vdom);
    }
  } else {
    dom = document.createElement(type);
  }
  if (props) {
    updateProps(dom, {}, props);
    if (typeof props.children == "object" && props.children.type) {
      mount(props.children, dom);
    } else if (Array.isArray(props.children)) {
      reconcileChildren(props.children, dom);
    }
  }
  vdom.dom = dom;
  if(ref) ref.current = dom;
  return dom;
}
function mountForwardComponent(vdom){
    let {type,props,ref} = vdom;
    let renderVdom = type.render(props,ref);
    vdom.oldRenderVdom = renderVdom;
    return createDOM(renderVdom);
}
function mountClassComponent(vdom) {
  let { type, props,ref } = vdom;
  let classInstance = new type(props);
+ if(ref) ref.current = classInstance;
+ if (classInstance.componentWillMount) classInstance.componentWillMount();
  let renderVdom = classInstance.render();
  classInstance.oldRenderVdom = renderVdom;
  let dom = createDOM(renderVdom);
+ if (classInstance.componentDidMount)
+   dom.componentDidMount = classInstance.componentDidMount.bind(classInstance);
  return dom;
}
function mountFunctionComponent(vdom) {
  let { type, props } = vdom;
  let renderVdom = type(props);
  vdom.oldRenderVdom = renderVdom;
  return createDOM(renderVdom);
}
function updateProps(dom, oldProps={}, newProps={}) {
    for (let key in newProps) {
        if (key
            continue;
        } else if (key
            let styleObj = newProps[key];
            for (let attr in styleObj) {
                dom.style[attr] = styleObj[attr];
            }
        } else if (key.startsWith('on')) {
            addEvent(dom, key.toLocaleLowerCase(), newProps[key]);
        } else {
            dom[key] = newProps[key];
        }
    }
    for(let key in oldProps){
        if(!newProps.hasOwnProperty(key)){
            dom[key] = null;
        }
    }
}
export function findDOM(vdom){
    let {type}= vdom;
    let dom;
    if(typeof type
        dom=findDOM(vdom.oldRenderVdom);
    }else{
        dom=vdom.dom;
    }
    return dom;
}
export function compareTwoVdom(parentDOM, oldVdom, newVdom) {
    let oldDOM = findDOM(oldVdom);
    let newDOM = createDOM(newVdom);
    parentDOM.replaceChild(newDOM, oldDOM);
}
function reconcileChildren(childrenVdom, parentDOM) {
  for (let i = 0; i < childrenVdom.length; i++) {
    mount(childrenVdom[i], parentDOM);
  }
}
const ReactDOM = {
  render,
};
export default ReactDOM;
```

9.子组件生命周期



9.1 src\index.js

src\index.js

```
import React from './react';
import ReactDOM from './react-dom';
class Counter extends React.Component{
    static defaultProps = {
        name: '珠峰架构'
    };
    constructor(props) {
        super(props);
        this.state = { number: 0 }
        console.log('Counter 1.constructor')
    }
    componentWillMount() {
        console.log('Counter 2.componentWillMount');
    }
    componentDidMount() {
        console.log('Counter 4.componentDidMount');
    }
    handleClick = () => {
        this.setState({ number: this.state.number + 1 });
    };

    shouldComponentUpdate(nextProps, nextState) {
        console.log('Counter 5.shouldComponentUpdate');
        return nextState.number % 2 === 0;

    }
    componentWillUpdate() {
        console.log('Counter 6.componentWillUpdate');
    }
    componentDidUpdate() {
        console.log('Counter 7.componentDidUpdate');
    }
    render() {
        console.log('Counter 3.render');
        return (
            <div>
                <p>{this.state.number}p>
                {this.state.number === 4 ? null : <ChildCounter count={this.state.number} />}
                <button onClick={this.handleClick}>+button>
            div>
        )
    }
}
class ChildCounter extends React.Component {
    componentWillUnmount() {
        console.log(' ChildCounter 6.componentWillUnmount')
    }
    componentWillMount() {
        console.log('ChildCounter 1.componentWillMount')
    }
    render() {
        console.log('ChildCounter 2.render')
        return (<div>
            {this.props.count}
        div>)
    }
    componentDidMount() {
        console.log('ChildCounter 3.componentDidMount')
    }
    componentWillReceiveProps(newProps) {
        console.log('ChildCounter 4.componentWillReceiveProps')
    }
    shouldComponentUpdate(nextProps, nextState) {
        console.log('ChildCounter 5.shouldComponentUpdate')
        return nextProps.count % 3 === 0;
    }
}
ReactDOM.render(<Counter />, document.getElementById('root'));
```

9.2 src\react-dom.js

src\react-dom.js

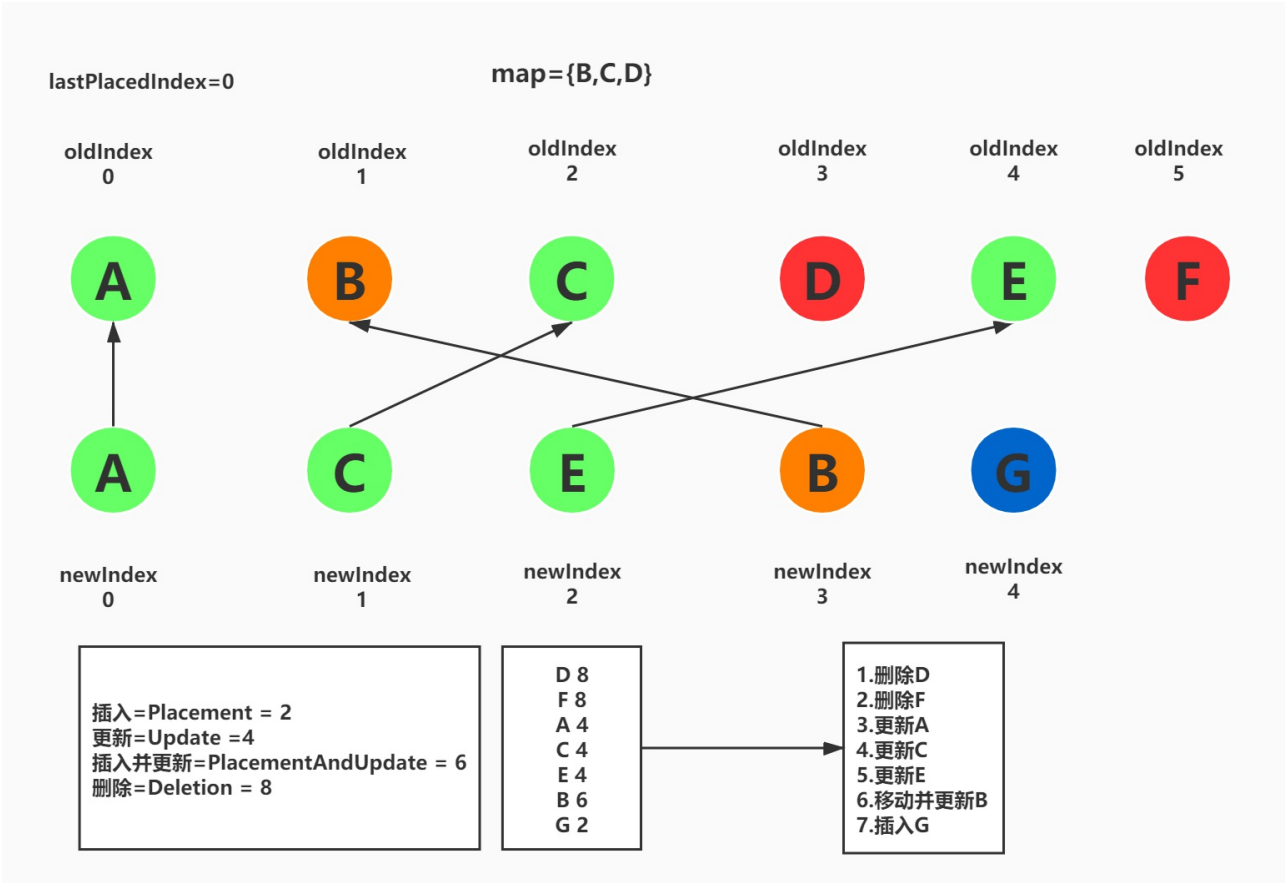
```
+import { REACT_TEXT, REACT_FORWARD_REF_TYPE } from "./constants";
+import { addEvent } from "./event";

function render(vdom, parentDOM) {
    let newDOM = createDOM(vdom)
    if (newDOM) {
        parentDOM.appendChild(newDOM);
        if (newDOM._componentDidMount) newDOM._componentDidMount();
    }
}
export function createDOM(vdom) {
 let { type, props, ref } = vdom;
  let dom;
  if (type && type.$typeof
    return mountForwardComponent(vdom);
  } else if (type
    dom = document.createTextNode(props);
  } else if (typeof type
    if (type.isReactComponent) {
      return mountClassComponent(vdom);
    } else {
      return mountFunctionComponent(vdom);
    }
  } else {
    dom = document.createElement(type);
  }
  if (props) {
    updateProps(dom, {}, props);
    if (typeof props.children == "object" && props.children.type) {
      mount(props.children, dom);
```

```
    } else if (Array.isArray(props.children)) {
      reconcileChildren(props.children, dom);
    }
  }
  vdom.dom = dom;
  if (ref) ref.current = dom;
  return dom;
}
function mountForwardComponent(vdom) {
  let { type, props, ref } = vdom;
  let renderVdom = type.render(props, ref);
  vdom.oldRenderVdom = renderVdom;
  return createDOM(renderVdom);
}
function mountClassComponent(vdom) {
  let { type, props, ref } = vdom;
  let classInstance = new type(props);
+ vdom.classInstance = classInstance;
  if (ref) ref.current = classInstance;
  if (classInstance.componentWillMount) classInstance.componentWillMount();
  let renderVdom = classInstance.render();
  classInstance.oldRenderVdom =  renderVdom;
  let dom = createDOM(renderVdom);
  if (classInstance.componentDidMount)
    dom.componentDidMount = classInstance.componentDidMount.bind(classInstance);
  return dom;
}
function mountFunctionComponent(vdom) {
  let { type, props } = vdom;
  let renderVdom = type(props);
  vdom.oldRenderVdom = renderVdom;
  return createDOM(renderVdom);
}
function updateProps(dom, oldProps={}, newProps={}) {
    for (let key in newProps) {
        if (key
            continue;
        } else if (key
            let styleObj = newProps[key];
            for (let attr in styleObj) {
                dom.style[attr] = styleObj[attr];
            }
        } else if (key.startsWith('on')) {
            addEvent(dom, key.toLocaleLowerCase(), newProps[key]);
        } else {
            dom[key] = newProps[key];
        }
    }
    for(let key in oldProps){
        if(!newProps.hasOwnProperty(key)){
            dom[key] = null;
        }
    }
}
export function findDOM(vdom) {
    if (!vdom) return null;
    if (vdom.dom) {//vdom={type:'h1'}
        return vdom.dom;
    } else {
        let renderVdom = vdom.classInstance ? vdom.classInstance.oldRenderVdom : vdom.oldRenderVdom;
        return findDOM(renderVdom);
    }
}
+function unMountVdom(vdom) {
+    let { type, props, ref } = vdom;
+    let currentDOM = findDOM(vdom);//获取此虚拟DOM对应的真实DOM
+    //vdom可能是原生组件span 类组件 classComponent 也可能是函数组件Function
+    if (vdom.classInstance && vdom.classInstance.componentWillUnmount) {
+        vdom.classInstance.componentWillUnmount();
+    }
+    if (ref) {
+        ref.current = null;
+    }
+    //如果此虚拟DOM有子节点的话，递归全部删除
+    if (props.children) {
+        //得到儿子的数组
+        let children = Array.isArray(props.children) ? props.children : [props.children];
+        children.forEach(unMountVdom);
+    }
+    //把自己这个虚拟DOM对应的真实DOM从界面删除
+    if (currentDOM) currentDOM.remove();
+}
+export function compareTwoVdom(parentDOM, oldVdom, newVdom,nextDOM) {
+  if (!oldVdom && !newVdom) {
+    //老和新都是没有
+    return;
+  } else if (!!oldVdom && !newVdom) {
+    //老有新没有
+    unMountVdom(oldVdom);
+  } else if (!oldVdom && !!newVdom) {
+    //老没有新的有
+    let newDOM = createDOM(newVdom);
+    if (nextDOM) parentDOM.insertBefore(newDOM, nextDOM);
+    else parentDOM.appendChild(newDOM);
+    if (newDOM.componentDidMount) newDOM.componentDidMount();
+    return;
+  } else if (!!oldVdom && !!newVdom && oldVdom.type !== newVdom.type) {
+    //新老都有，但类型不同
+    let newDOM = createDOM(newVdom);
+    unMountVdom(oldVdom);
+    if (newDOM.componentDidMount) newDOM.componentDidMount();
+  } else {
+    updateElement(oldVdom, newVdom);
+  }
```

```
+}
+function updateElement(oldVdom, newVdom) {
+    if (oldVdom.type === REACT_TEXT) {
+       let currentDOM = newVdom.dom = findDOM(oldVdom);
+       if (oldVdom.props !== newVdom.props) {
+           currentDOM.textContent = newVdom.props;
+       }
+       return;
+    }else if (typeof oldVdom.type === 'string') {
+        let currentDOM = newVdom.dom = findDOM(oldVdom);
+        updateProps(currentDOM, oldVdom.props, newVdom.props);
+        updateChildren(currentDOM, oldVdom.props.children, newVdom.props.children);
+    } else if (typeof oldVdom.type === 'function') {
+        if (oldVdom.type.isReactComponent) {
+            updateClassComponent(oldVdom, newVdom);
+        } else {
+            updateFunctionComponent(oldVdom, newVdom);
+        }
+    }
+}
+function updateFunctionComponent(oldVdom, newVdom) {
+    let currentDOM = findDOM(oldVdom);
+    if (!currentDOM) return;
+    let parentDOM = currentDOM.parentNode;
+    let { type, props } = newVdom;
+    let newRenderVdom = type(props);
+    compareTwoVdom(parentDOM, oldVdom.oldRenderVdom, newRenderVdom);
+    newVdom.oldRenderVdom = newRenderVdom;
+}
+function updateClassComponent(oldVdom, newVdom) {
+    let classInstance = newVdom.classInstance = oldVdom.classInstance;
+    if (classInstance.componentWillReceiveProps) {
+        classInstance.componentWillReceiveProps(newVdom.props);
+    }
+    classInstance.updater.emitUpdate(newVdom.props);
+}
+function updateChildren(parentDOM, oldVChildren, newVChildren) {
+    oldVChildren = (Array.isArray(oldVChildren) ? oldVChildren : oldVChildren ? [oldVChildren]).filter(item => item) : [];
+    newVChildren = (Array.isArray(newVChildren) ? newVChildren : newVChildren ? [newVChildren]).filter(item => item) : [];
+    let maxLength = Math.max(oldVChildren.length, newVChildren.length);
+    for (let i = 0; i < maxLength; i++) {
+       let nextVdom = oldVChildren.find((item,index)=>index>i&&item&&findDOM(item));
+       compareTwoVdom(parentDOM, oldVChildren[i], newVChildren[i],nextVdom&&findDOM(nextVdom));
+    }
+}
function reconcileChildren(childrenVdom, parentDOM) {
  for (let i = 0; i < childrenVdom.length; i++) {
    mount(childrenVdom[i], parentDOM);
  }
}
const ReactDOM = {
  render,
};
export default ReactDOM;
```

10.DOM-DIFF算法

- 只对同级节点进行对比，如果DOM节点跨层级移动，则React不会复用
- 不同类型的元素会产出不同的结构 ， 会销毁老结构，创建新结构
- 可以通过 key标识移动的元素



10.1 src\index.js

src\index.js

```
import React from './react';
import ReactDOM from './react-dom';
class Counter extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            list: ['A', 'B', 'C', 'D', 'E', 'F']
        }
    }
    handleClick = () => {
        this.setState({
            list: ['A', 'C', 'E', 'B', 'G']
        });
    };
    render() {
        return (
            <React.Fragment>
                <ul>
                    {
                        this.state.list.map(item => <li key={item}>{item}li>)
                    }

                ul>
                <button onClick={this.handleClick}>+button>
            React.Fragment>
        )
    }
}
ReactDOM.render(<Counter />, document.getElementById('root'));
```

10.2 src\constants.js

src\constants.js

```
export const REACT_TEXT = Symbol('REACT_TEXT');
export const REACT_FORWARD_REF_TYPE = Symbol('react.forward_ref');
+export const REACT_FRAGMENT = Symbol('react.fragment')
+export const PLACEMENT = 'PLACEMENT';
+export const MOVE = 'MOVE';
```

10.3 src\react.js

src\react.js

```
import { wrapToVdom } from "./utils";
import { Component } from './Component';
+import { REACT_FORWARD_REF_TYPE,REACT_FRAGMENT } from './constants';
function createElement(type, config, children) {
  let ref;
  let key;
  if (config) {
    delete config.__source;
    delete config.__self;
    ref = config.ref;
    delete config.ref;
    key = config.key;
    delete config.key;
  }
  let props = { ...config };
  if (arguments.length > 3) {
    props.children = Array.prototype.slice.call(arguments, 2).map(wrapToVdom);
  } else {
    props.children = wrapToVdom(children);
  }
  return {
    type,
    ref,
    key,
    props,
  };
}
function createRef() {
  return { current: null };
}
function forwardRef(render) {
  var elementType = {
    $typeof: REACT_FORWARD_REF_TYPE,
    render: render
  };
  return elementType;
}
const React = {
  createElement,
  Component,
  createRef,
  forwardRef,
  Fragment: REACT_FRAGMENT
};
export default React;
```

10.4 src\react-dom.js

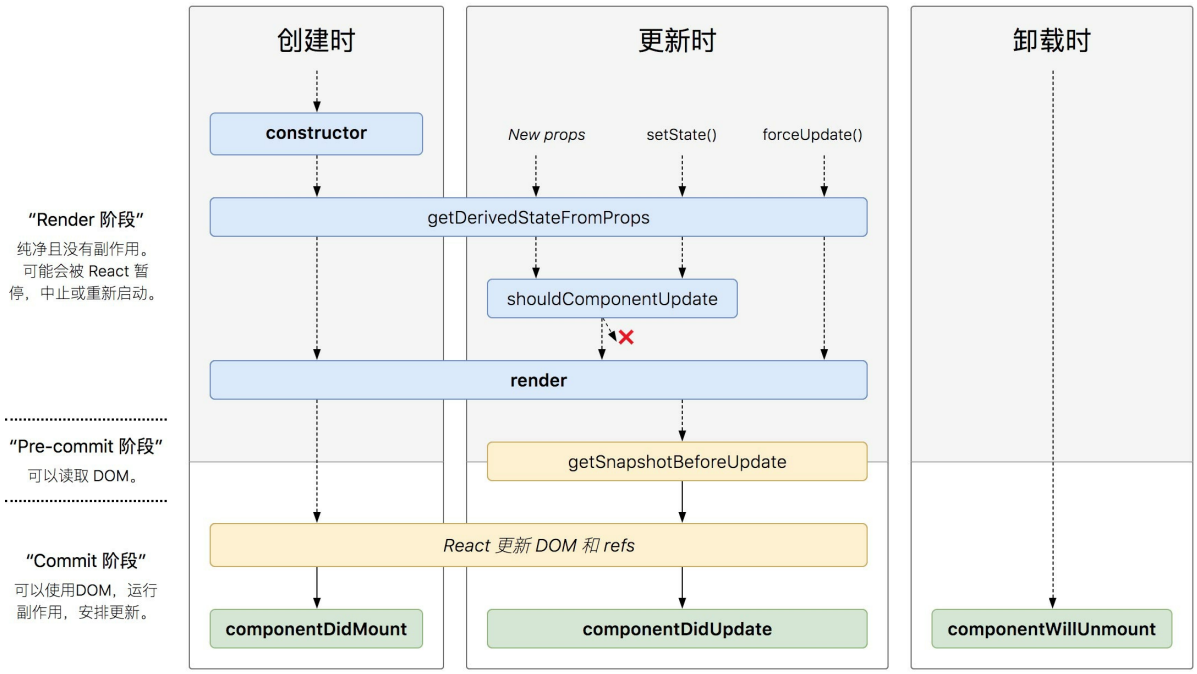
src\react-dom.js

```
+import { REACT_TEXT, REACT_FORWARD_REF_TYPE, PLACEMENT, MOVE ,REACT_FRAGMENT} from "./constants";
import { addEvent } from "./event";
+import React from './react';
function render(vdom, parentDOM) {
    let newDOM = createDOM(vdom)
    if (newDOM) {
        parentDOM.appendChild(newDOM);
        if (newDOM._componentDidMount) newDOM._componentDidMount();
    }
}
export function createDOM(vdom) {
  let { type, props, ref } = vdom;
  let dom;
  if (type && type.$typeof
    return mountForwardComponent(vdom);
  } else if (type
    dom = document.createTextNode(props);
+ } else if (oldVdom.type === REACT_FRAGMENT) {
+  dom = document.createDocumentFragment();
+ }else if (typeof type === "function") {
    if (type.isReactComponent) {
      return mountClassComponent(vdom);
    } else {
     return mountFunctionComponent(vdom);
    }
  } else {
    dom = document.createElement(type);
  }
  if (props) {
    updateProps(dom, {}, props);
    if (typeof props.children == "object" && props.children.type) {
+     props.children.mountIndex = 0;
      mount(props.children, dom);
    } else if (Array.isArray(props.children)) {
      reconcileChildren(props.children, dom);
    }
  }
  vdom.dom = dom;
  if (ref) ref.current = dom;
  return dom;
}

function mountForwardComponent(vdom) {
  let { type, props, ref } = vdom;
  let renderVdom = type.render(props, ref);
  vdom.oldRenderVdom = renderVdom;
  return createDOM(renderVdom);
}
function mountClassComponent(vdom) {
  let { type, props, ref } = vdom;
  let classInstance = new type(props);
  vdom.classInstance = classInstance;
  if (ref) ref.current = classInstance;
  if (classInstance.componentWillMount) classInstance.componentWillMount();
```

```
  let renderVdom = classInstance.render();
  classInstance.oldRenderVdom  = renderVdom;
  let dom = createDOM(renderVdom);
  if (classInstance.componentDidMount)
    dom.componentDidMount = classInstance.componentDidMount.bind(classInstance);
  return dom;
}
function mountFunctionComponent(vdom) {
  let { type, props } = vdom;
  let renderVdom = type(props);
  vdom.oldRenderVdom = renderVdom;
  return createDOM(renderVdom);
}
function updateProps(dom, oldProps={}, newProps={}) {
    for (let key in newProps) {
        if (key
            continue;
        } else if (key
            let styleObj = newProps[key];
            for (let attr in styleObj) {
                dom.style[attr] = styleObj[attr];
            }
        } else if (key.startsWith('on')) {
            addEvent(dom, key.toLocaleLowerCase(), newProps[key]);
        } else {
            dom[key] = newProps[key];
        }
    }
    for(let key in oldProps){
        if(!newProps.hasOwnProperty(key)){
            dom[key] = null;
        }
    }
}
export function findDOM(vdom) {
    if (!vdom) return null;
    if (vdom.dom) {//vdom={type:'h1'}
        return vdom.dom;
    } else {
        //如果不这样修改就需要在更新的时候也同步vdom.oldRenderVdom
        let renderVdom = vdom.classInstance ? vdom.classInstance.oldRenderVdom : vdom.oldRenderVdom;
        return findDOM(renderVdom);
    }
}
function unMountVdom(vdom) {
    let { type, props, ref } = vdom;
    let currentDOM = findDOM(vdom);//获取此虚拟DOM对应的真实DOM
    //vdom可能是原生组件span 类组件 classComponent 也可能是函数组件Function
    if (vdom.classInstance && vdom.classInstance.componentWillUnmount) {
        vdom.classInstance.componentWillUnmount();
    }
    if (ref) {
        ref.current = null;
    }
    //如果此虚拟DOM有子节点的话，递归全部删除
    if (props.children) {
        //得到儿子的数组
        let children = Array.isArray(props.children) ? props.children : [props.children];
        children.forEach(unMountVdom);
    }
    //把自己这个虚拟DOM对应的真实DOM从界面删除
    if (currentDOM) currentDOM.remove();
}
export function compareTwoVdom(parentDOM, oldVdom, newVdom, nextDOM) {
  if (!oldVdom && !newVdom) {
    //老和新都是没有
    return;
  } else if (!!oldVdom && !newVdom) {
    //老有新没有
    unMountVdom(oldVdom);
  } else if (!oldVdom && !!newVdom) {
    //老没有新的有
    let newDOM = createDOM(newVdom);
    if (nextDOM) parentDOM.insertBefore(newDOM, nextDOM);
    else parentDOM.appendChild(newDOM);
    if (newDOM.componentDidMount) newDOM.componentDidMount();
    return;
  } else if (!!oldVdom && !!newVdom && oldVdom.type !== newVdom.type) {
    //新老都有，但类型不同
    let newDOM = createDOM(newVdom);
    unMountVdom(oldVdom);
    if (newDOM.componentDidMount) newDOM.componentDidMount();
  } else {
    updateElement(oldVdom, newVdom);
  }
}
function updateElement(oldVdom, newVdom) {
  if (oldVdom.type
    let currentDOM = newVdom.dom = findDOM(oldVdom);
    if (oldVdom.props !== newVdom.props) {
      currentDOM.textContent = newVdom.props;
    }
    return;
  } else if (typeof oldVdom.type
    let currentDOM = newVdom.dom = findDOM(oldVdom);
    updateProps(currentDOM, oldVdom.props, newVdom.props);
    updateChildren(currentDOM, oldVdom.props.children, newVdom.props.children);
+ } else if (oldVdom.type === REACT_FRAGMENT) {
+   let currentDOM = newVdom.dom = findDOM(oldVdom);
+   updateChildren(currentDOM, oldVdom.props.children, newVdom.props.children);
+ } else if (typeof oldVdom.type === 'function') {
    if (oldVdom.type.isReactComponent) {
      updateClassComponent(oldVdom, newVdom);
    } else {
```

```
    updateFunctionComponent(oldVdom, newVdom);
   }
  }
}
function updateFunctionComponent(oldVdom, newVdom) {
  let currentDOM = findDOM(oldVdom);
  if (!currentDOM) return;
  let parentDOM = currentDOM.parentNode;
  let { type, props } = newVdom;
  let newRenderVdom = type(props);
  compareTwoVdom(parentDOM, oldVdom.oldRenderVdom, newRenderVdom);
  newVdom.oldRenderVdom = newRenderVdom;
}
function updateClassComponent(oldVdom, newVdom) {
  let classInstance = newVdom.classInstance = oldVdom.classInstance;
  //如果findDOM不从classInstance上获取oldRenderVdom就需要在更新的时候也同步
  //newVdom.oldRenderVdom = newVdom.oldRenderVdom
  if (classInstance.componentWillReceiveProps) {
    classInstance.componentWillReceiveProps();
  }
  classInstance.updater.emitUpdate(newVdom.props);
}
function updateChildren(parentDOM, oldVChildren, newVChildren) {
+  oldVChildren = (Array.isArray(oldVChildren) ? oldVChildren : oldVChildren ? [oldVChildren]).filter(item => item) : [];
+  newVChildren = (Array.isArray(newVChildren) ? newVChildren : newVChildren ? [newVChildren]).filter(item => item) : [];
+  let keyedOldMap = {};
+  let lastPlacedIndex = 0;
+  oldVChildren.forEach((oldVChild, index) => {
+    let oldKey = oldVChild.key ? oldVChild.key : index;
+    keyedOldMap[oldKey] = oldVChild;
+  });
+  let patch = [];
+  newVChildren.forEach((newVChild, index) => {
+    newVChild.mountIndex = index;
+    let newKey = newVChild.key ? newVChild.key : index;
+    let oldVChild = keyedOldMap[newKey];
+    if (oldVChild) {
+      updateElement(oldVChild, newVChild);
+      if (oldVChild.mountIndex < lastPlacedIndex) {
+        patch.push({
+          type: MOVE,
+          oldVChild,
+          newVChild,
+          mountIndex: index
+        });
+      }
+      delete keyedOldMap[newKey];
+      lastPlacedIndex = Math.max(lastPlacedIndex, oldVChild.mountIndex);
+    } else {
+      patch.push({
+        type: PLACEMENT,
+        newVChild,
+        mountIndex: index
+      });
+    }
+  });
+  let moveVChild = patch.filter(action => action.type === MOVE).map(action => action.oldVChild);
+  Object.values(keyedOldMap).concat(moveVChild).forEach((oldVChild) => {
+    let currentDOM = findDOM(oldVChild);
+    parentDOM.removeChild(currentDOM);
+  });
+  patch.forEach(action => {
+    let { type, oldVChild, newVChild, mountIndex } = action;
+    let childNodes = parentDOM.childNodes;
+    if (type === PLACEMENT) {
+      let newDOM = createDOM(newVChild);
+      let childNode = childNodes[mountIndex];
+      if (childNode) {
+        parentDOM.insertBefore(newDOM, childNode);
+      } else {
+        parentDOM.appendChild(newDOM);
+      }
+    } else if (type === MOVE) {
+      let oldDOM = findDOM(oldVChild);
+      let childNode = childNodes[mountIndex];
+      if (childNode) {
+        parentDOM.insertBefore(oldDOM, childNode);
+      } else {
+        parentDOM.appendChild(oldDOM);
+      }
+    }
+  });
}
function reconcileChildren(childrenVdom, parentDOM) {
  for (let i = 0; i < childrenVdom.length; i++) {
+   childrenVdom[i].mountIndex = i;
    mount(childrenVdom[i], parentDOM);
  }
}
const ReactDOM = {
  render,
};
export default ReactDOM;
```

11.新的生命周期



11.1 getDerivedStateFromProps

- `static getDerivedStateFromProps(props, state)` 这个生命周期的功能实际上就是将传入的props映射到state上面

```
import React from 'react';
import ReactDOM from 'react-dom';
class Counter extends React.Component{
    static defaultProps = {
        name: '珠峰架构'
    };
    constructor(props) {
        super(props);
        this.state = { number: 0 }
    }

    handleClick = () => {
        this.setState({ number: this.state.number + 1 });
    };

    render() {
        console.log('3.render');
        return (
            <div>
                <p>{this.state.number}p>
                <ChildCounter number={this.state.number} />
                <button onClick={this.handleClick}>+button>
            div>
        )
    }
}
class ChildCounter extends React.Component {
    constructor(props) {
        super(props);
        this.state = { number: 0 };
    }
    static getDerivedStateFromProps(nextProps, prevState) {
        const { count } = nextProps;

        if (count % 2 === 0) {
            return { number: number * 2 };
        } else {
            return { number: number * 3 };
        }
    }
    render() {
        console.log('child-render', this.state)
        return (<div>
            {this.state.number}
        div>)
    }

}

ReactDOM.render(
    <Counter />,
    document.getElementById('root')
);
```

11.2 getSnapshotBeforeUpdate

- getSnapshotBeforeUpdate() 被调用于render之后，可以读取但无法使用DOM的时候。它使您的组件可以在可能更改之前从DOM捕获一些信息（例如滚动位置）。此生命周期返回的任何值都将作为参数传递给componentDidUpdate()

```
import React from './react';
import ReactDOM from './react-dom';
class ScrollingList extends React.Component {
    constructor(props) {
        super(props);
        this.state = { messages: [] }
        this.wrapper = React.createRef();
    }

    addMessage() {
        this.setState(state => ({
            messages: [`${state.messages.length}`, ...state.messages],
        }))
    }
    componentDidMount() {
        this.timeID = window.setInterval(() => {
            this.addMessage();
        }, 1000)
    }
    componentWillUnmount() {
        window.clearInterval(this.timeID);
    }
    getSnapshotBeforeUpdate() {
        return {prevScrollTop:this.wrapper.current.scrollTop,prevScrollHeight:this.wrapper.current.scrollHeight};
    }
    componentDidUpdate(pervProps, pervState, {prevScrollHeight,prevScrollTop}) {

        this.wrapper.current.scrollTop = prevScrollTop + (this.wrapper.current.scrollHeight - prevScrollHeight);
    }
    render() {
        let style = {
            height: '100px',
            width: '200px',
            border: '1px solid red',
            overflow: 'auto'
        }

        return (
            <div style={style} ref={this.wrapper} >
                {this.state.messages.map((message, index) => (
                    <div key={index}>{message}div>
                ))}
            div>
        );
    }
}

ReactDOM.render(
    <ScrollingList />,
    document.getElementById('root')
);
```

11.3 实现

11.3.1 src\Component.js

```
import { findDOM, compareTwoVdom } from './react-dom';
export let updateQueue = {
    isBatchingUpdate: false,
    updaters: [],
    batchUpdate() {//批量更新
        updateQueue.isBatchingUpdate = false;
        for (let updater of updateQueue.updaters) {
            updater.updateComponent();
        }
        updateQueue.updaters.length = 0;
    }
}
class Updater {
    constructor(classInstance) {
        this.classInstance = classInstance;
        this.pendingStates = [];
        this.callbacks = [];
    }
    addState(partialState, callback) {
        this.pendingStates.push(partialState);///等待更新的或者说等待生效的状态
        if (typeof callback
            this.callbacks.push(callback);//状态更新后的回调
        this.emitUpdate();
    }
    emitUpdate(nextProps) {
        this.nextProps = nextProps;
        if (updateQueue.isBatchingUpdate) {
            updateQueue.updaters.push(this);
        } else {
            this.updateComponent();
        }
    }
    updateComponent() {
        let { classInstance, pendingStates } = this;
        if (this.nextProps || pendingStates.length > 0) {
            shouldUpdate(classInstance, this.nextProps, this.getState());
        }
    }
    getState() {
        let { classInstance, pendingStates } = this;
        let { state } = classInstance;
        pendingStates.forEach((nextState) => {
            if (typeof nextState
                nextState = nextState(state);
            }
            state = { ...state, ...nextState };
        });
        pendingStates.length = 0;
        return state;
    }
}
function shouldUpdate(classInstance, nextProps, nextState) {
    let willUpdate = true;
    if (classInstance.shouldComponentUpdate
        && !classInstance.shouldComponentUpdate(nextProps, nextState)) {
        willUpdate = false;
    }
    if (willUpdate && classInstance.componentWillUpdate) {
        classInstance.componentWillUpdate();
    }
    if (nextProps) {
        classInstance.props = nextProps;
    }
    classInstance.state = nextState;
    if (willUpdate) classInstance.forceUpdate();
}
export class Component {
    static isReactComponent = true;
    constructor(props) {
        this.props = props;
        this.state = {};
        this.updater = new Updater(this);
    }
    setState(partialState, callback) {
        this.updater.addState(partialState, callback);
    }
    forceUpdate() {
        let oldRenderVdom = this.oldRenderVdom;
        let oldDOM = findDOM(oldRenderVdom);
+       if (this.constructor.getDerivedStateFromProps) {
+           let newState = this.constructor.getDerivedStateFromProps(this.props, this.state);
+           if (newState)
+               this.state =  { ...this.state, ...newState };
+       }
+       let snapshot = this.getSnapshotBeforeUpdate && this.getSnapshotBeforeUpdate();
        let newRenderVdom = this.render();
        compareTwoVdom(oldDOM.parentNode, oldRenderVdom, newRenderVdom);
        this.oldRenderVdom = newRenderVdom;
        if (this.componentDidUpdate) {
+           this.componentDidUpdate(this.props, this.state, snapshot);
        }
    }
}
```

12. Context(上下文)

- 在某些场景下，你想在整个组件树中传递数据，但却不想手动地在每一层传递属性。你可以直接在 React 中使用强大的contextAPI解决上述问题
- 在一个典型的 React 应用中，数据是通过 props 属性自上而下（由父及子）进行传递的，但这种做法对于某些类型的属性而言是极其繁琐的（例如：地区偏好，UI 主题），这些属性是应用程序中许多组件都需要的。Context 提供了一种在组件之间共享此类值的方式，而不必显式地通过组件树的逐层传递 props

12.1 src\index.js #

```
import React from './react';
import ReactDOM from './react-dom';
let ThemeContext = React.createContext();
console.log(ThemeContext);
const { Provider, Consumer } = ThemeContext;
let style = { margin: '5px', padding: '5px' };
function Title(props) {
  console.log('Title');
  return (
    <Consumer>
      {
        (contextValue) => (
          <div style={{ ...style, border: `5px solid ${contextValue.color}` }}>
            Title
          div>
        )
      }
    Consumer>
  )
}
class Header extends React.Component {
  static contextType = ThemeContext
  render() {
    console.log('Header');
    return (
      <div style={{ ...style, border: `5px solid ${this.context.color}` }}>
        Header
        <Title />
      div>
    )
  }
}
function Content() {
  console.log('Content');
  return (
    <Consumer>
      {
        (contextValue) => (
          <div style={{ ...style, border: `5px solid ${contextValue.color}` }}>
            Content
            <button style={{ color: 'red' }} onClick={() => contextValue.changeColor('red')}>变红button>
            <button style={{ color: 'green' }} onClick={() => contextValue.changeColor('green')}>变绿button>
          div>
        )
      }
    Consumer>
  )
}
class Main extends React.Component {
  static contextType = ThemeContext
  render() {
    console.log('Main');
    return (
      <div style={{ ...style, border: `5px solid ${this.context.color}` }}>
        Main
        <Content />
      div>
    )
  }
}

class Page extends React.Component {
  constructor(props) {
    super(props);
    this.state = { color: 'black' };
  }
  changeColor = (color) => {
    this.setState({ color });
  }
  render() {

    console.log('Page');
    let contextValue = { color: this.state.color, changeColor: this.changeColor };
    return (
      <Provider value={contextValue}>
        <div style={{ ...style, width: '250px', border: `5px solid ${this.state.color}` }}>
          Page
          <Header />
          <Main />
        div>
      Provider >
    )
  }
}
ReactDOM.render(
  <Page />,
  document.getElementById('root')
);
```

12.2 src\constants.js

src\constants.js

```
export const REACT_TEXT = Symbol('REACT_TEXT');
export const REACT_FORWARD_REF_TYPE = Symbol('react.forward_ref');

export const PLACEMENT = 'PLACEMENT';
export const MOVE = 'MOVE';

+export const REACT_PROVIDER = Symbol('react.provider');
+export const REACT_CONTEXT = Symbol('react.context');
```

12.3 src\Component.js

src\Component.js

```
import { findDOM, compareTwoVdom } from './react-dom';
export let updateQueue = {
    isBatchingUpdate: false,
    updaters: [],
    batchUpdate() {//批量更新
        updateQueue.isBatchingUpdate = false;
        for (let updater of updateQueue.updaters) {
            updater.updateComponent();
        }
        updateQueue.updaters.length = 0;
    }
}
class Updater {
    constructor(classInstance) {
        this.classInstance = classInstance;
        this.pendingStates = [];
        this.callbacks = [];
    }
    addState(partialState, callback) {
        this.pendingStates.push(partialState);///等待更新的或者说等待生效的状态
        if (typeof callback
            this.callbacks.push(callback);//状态更新后的回调
        this.emitUpdate();
    }
    emitUpdate(nextProps) {
        this.nextProps = nextProps;
        if (updateQueue.isBatchingUpdate) {
            updateQueue.updaters.push(this);
        } else {
            this.updateComponent();
        }
    }
    updateComponent() {
        let { classInstance, pendingStates } = this;
        if (this.nextProps || pendingStates.length > 0) {
            shouldUpdate(classInstance, this.nextProps, this.getState());
        }
    }
    getState() {
        let { classInstance, pendingStates } = this;
        let { state } = classInstance;
        pendingStates.forEach((nextState) => {
            if (typeof nextState
                nextState = nextState(state);
            }
            state = { ...state, ...nextState };
        });
        pendingStates.length = 0;
        return state;
    }
}
function shouldUpdate(classInstance, nextProps, nextState) {
    let willUpdate = true;
    if (classInstance.shouldComponentUpdate
        && !classInstance.shouldComponentUpdate(nextProps, nextState)) {
        willUpdate = false;
    }
    if (willUpdate && classInstance.componentWillUpdate) {
        classInstance.componentWillUpdate();
    }
    if (nextProps) {
        classInstance.props = nextProps;
    }
    classInstance.state = nextState;
    if (willUpdate) classInstance.forceUpdate();
}
export class Component {
    static isReactComponent = true;
    constructor(props) {
        this.props = props;
        this.state = {};
        this.updater = new Updater(this);
    }
    setState(partialState, callback) {
        this.updater.addState(partialState, callback);
    }
    forceUpdate() {
        let oldRenderVdom = this.oldRenderVdom;
        debugger
        let oldDOM = findDOM(oldRenderVdom);
+       if (this.constructor.contextType) {
+           this.context = this.constructor.contextType._currentValue;
+       }
        if (this.constructor.getDerivedStateFromProps) {
            let newState = this.constructor.getDerivedStateFromProps(this.props, this.state);
            if (newState)
                this.state = newState;
        }
        let extraArgs = this.getSnapshotBeforeUpdate && this.getSnapshotBeforeUpdate();
        let newRenderVdom = this.render();
        compareTwoVdom(oldDOM.parentNode, oldRenderVdom, newRenderVdom);
        this.oldRenderVdom = newRenderVdom;
        if (this.componentDidUpdate) {
            this.componentDidUpdate(this.props, this.state, extraArgs);
        }
    }
}
```

12.4 src\react.js

src\react.js

```
import { wrapToVdom } from "./utils";
import { Component } from './Component';
+import { REACT_FORWARD_REF_TYPE, REACT_FRAGMENT,REACT_CONTEXT, REACT_PROVIDER } from './constants';
function createElement(type, config, children) {
  let ref;
  let key;
  if (config) {
    delete config.__source;
    delete config.__self;
    ref = config.ref;
    delete config.ref;
    key = config.key;
    delete config.key;
  }
  let props = { ...config };
  if (arguments.length > 3) {
    props.children = Array.prototype.slice.call(arguments, 2).map(wrapToVdom);
  } else {
    props.children = wrapToVdom(children);
  }
  return {
    type,
    ref,
    key,
    props,
  };
}
function createRef() {
  return { current: null };
}
function forwardRef(render) {
  var elementType = {
    $typeof: REACT_FORWARD_REF_TYPE,
    render: render
  };
  return elementType;
}
+function createContext() {
+  let context = { _currentValue: undefined };
+  context.Provider = {
+    $typeof: REACT_PROVIDER,
+    _context: context
+  }
+  context.Consumer = {
+    $typeof: REACT_CONTEXT,
+    _context: context
+  }
+  return context;
+}
const React = {
  createElement,
  Component,
  createRef,
  forwardRef,
  Fragment:REACT_FRAGMENT,
+ createContext
};
export default React;
```

12.5 src\react-dom.js

src\react-dom.js

```
+import { REACT_TEXT, REACT_FORWARD_REF_TYPE, PLACEMENT, MOVE, REACT_PROVIDER, REACT_CONTEXT,REACT_FRAGMENT } from "./constants";
import { addEvent } from "./event";
import React from './react';
function render(vdom, parentDOM) {
    let newDOM = createDOM(vdom)
    if (newDOM) {
        parentDOM.appendChild(newDOM);
        if (newDOM._componentDidMount) newDOM._componentDidMount();
    }
}
export function createDOM(vdom) {
  let { type, props, ref } = vdom;
  let dom;
+ if (type && type.$typeof === REACT_PROVIDER) {
+   return mountProviderComponent(vdom)
+ } else if (type && type.$typeof === REACT_CONTEXT) {
+   return mountContextComponent(vdom)
+ } else if (type && type.$typeof === REACT_FORWARD_REF_TYPE) {
    return mountForwardComponent(vdom);
  } else if (type
    dom = document.createTextNode(props);
  } else if (type
    dom = document.createDocumentFragment();
  } else if (typeof type
    if (type.isReactComponent) {
      return mountClassComponent(vdom);
    } else {
       return mountFunctionComponent(vdom);
    }
  } else {
    dom = document.createElement(type);
  }
  if (props) {
    updateProps(dom, {}, props);
    if (typeof props.children == "object" && props.children.type) {
      props.children.mountIndex = 0;
      mount(props.children, dom);
    } else if (Array.isArray(props.children)) {
      reconcileChildren(props.children, dom);
    }
```

```
  }
  vdom.dom = dom;
  if (ref) ref.current = dom;
  return dom;
}
+function mountProviderComponent(vdom) {
+  let { type, props } = vdom;
+  let context = type._context;
+  context._currentValue = props.value;
+  let renderVdom = props.children;
+  vdom.oldRenderVdom = renderVdom;
+  return createDOM(renderVdom);
+}
+function mountContextComponent(vdom) {
+  let { type, props } = vdom;
+  let context = type._context;
+  let renderVdom = props.children(context._currentValue);
+  vdom.oldRenderVdom = renderVdom;
+  return createDOM(renderVdom);
+}
function mountForwardComponent(vdom) {
  let { type, props, ref } = vdom;
  let renderVdom = type.render(props, ref);
  vdom.oldRenderVdom = renderVdom;
  return createDOM(renderVdom);
}
function mountClassComponent(vdom) {
  let { type, props, ref } = vdom;
  let classInstance = new type(props);
+ if (type.contextType) {
+   classInstance.context = type.contextType._currentValue;
+ }
  vdom.classInstance = classInstance;
  if (ref) ref.current = classInstance;
  if (classInstance.componentWillMount) classInstance.componentWillMount();
  let renderVdom = classInstance.render();
  classInstance.oldRenderVdom  = renderVdom;
  let dom = createDOM(renderVdom);
  if (classInstance.componentDidMount)
    dom.componentDidMount = classInstance.componentDidMount.bind(classInstance);
  return dom;
}
function mountFunctionComponent(vdom) {
  let { type, props } = vdom;
  let renderVdom = type(props);
  vdom.oldRenderVdom = renderVdom;
  return createDOM(renderVdom);
}
function updateProps(dom, oldProps={}, newProps={}) {
    for (let key in newProps) {
        if (key
            continue;
        } else if (key
            let styleObj = newProps[key];
            for (let attr in styleObj) {
                dom.style[attr] = styleObj[attr];
            }
        } else if (key.startsWith('on')) {
            addEvent(dom, key.toLocaleLowerCase(), newProps[key]);
        } else {
            dom[key] = newProps[key];
        }
    }
    for(let key in oldProps){
        if(!newProps.hasOwnProperty(key)){
            dom[key] = null;
        }
    }
}
export function findDOM(vdom) {
    if (!vdom) return null;
    if (vdom.dom) {//vdom={type:'h1'}
        return vdom.dom;
    } else {
        let renderVdom = vdom.classInstance ? vdom.classInstance.oldRenderVdom : vdom.oldRenderVdom;
        return findDOM(renderVdom);
    }
}
function unMountVdom(vdom) {
    let { type, props, ref } = vdom;
    let currentDOM = findDOM(vdom);//获取此虚拟DOM对应的真实DOM
    //vdom可能是原生组件span 类组件 classComponent 也可能是函数组件Function
    if (vdom.classInstance && vdom.classInstance.componentWillUnmount) {
        vdom.classInstance.componentWillUnmount();
    }
    if (ref) {
        ref.current = null;
    }
    //如果此虚拟DOM有子节点的话，递归全部删除
    if (props.children) {
        //得到儿子的数组
        let children = Array.isArray(props.children) ? props.children : [props.children];
        children.forEach(unMountVdom);
    }
    //把自己这个虚拟DOM对应的真实DOM从界面删除
    if (currentDOM) currentDOM.remove();
}
export function compareTwoVdom(parentDOM, oldVdom, newVdom, nextDOM) {
  if (!oldVdom && !newVdom) {
    //老和新都是没有
    return;
  } else if (!!oldVdom && !newVdom) {
    //老有新没有
    unMountVdom(oldVdom);
```

```
  } else if (!oldVdom && !!newVdom) {
    //老没有新的有
    let newDOM = createDOM(newVdom);
    if (nextDOM) parentDOM.insertBefore(newDOM, nextDOM);
    else parentDOM.appendChild(newDOM);
    if (newDOM.componentDidMount) newDOM.componentDidMount();
    return;
  } else if (!!oldVdom && !!newVdom && oldVdom.type !== newVdom.type) {
    //新老都有，但类型不同
    let newDOM = createDOM(newVdom);
    unMountVdom(oldVdom);
    if (newDOM.componentDidMount) newDOM.componentDidMount();
  } else {
    updateElement(oldVdom, newVdom);
  }
}
function updateElement(oldVdom, newVdom) {
+ if (oldVdom.type.$typeof === REACT_CONTEXT) {
+   updateContextComponent(oldVdom, newVdom);
+ } else if (oldVdom.type.$typeof === REACT_PROVIDER) {
+   updateProviderComponent(oldVdom, newVdom);
+ } else if (oldVdom.type === REACT_TEXT) {
    let currentDOM = newVdom.dom = findDOM(oldVdom);
    if (oldVdom.props !== newVdom.props) {
      currentDOM.textContent = newVdom.props;
    }
    return;
  } else if (oldVdom.type
     let currentDOM = newVdom.dom = findDOM(oldVdom);
    updateChildren(currentDOM, oldVdom.props.children, newVdom.props.children);
  } else if (typeof oldVdom.type
    let currentDOM = newVdom.dom = findDOM(oldVdom);
    updateProps(currentDOM, oldVdom.props, newVdom.props);
    updateChildren(currentDOM, oldVdom.props.children, newVdom.props.children);
  } else if (typeof oldVdom.type
    if (oldVdom.type.isReactComponent) {
      updateClassComponent(oldVdom, newVdom);
    } else {
       updateFunctionComponent(oldVdom, newVdom);
    }
  }
}
+function updateProviderComponent(oldVdom, newVdom) {
+  let parentDOM = findDOM(oldVdom).parentNode;
+  let { type, props } = newVdom;
+  let context = type._context;
+  context._currentValue = props.value;
+  let renderVdom = props.children;
+  compareTwoVdom(parentDOM, oldVdom.oldRenderVdom, renderVdom);
+  newVdom.oldRenderVdom = renderVdom;
+}
+function updateContextComponent(oldVdom, newVdom) {
+  let parentDOM = findDOM(oldVdom).parentNode;
+  let { type, props } = newVdom;
+  let context = type._context;
+  let renderVdom = props.children(context._currentValue);
+  compareTwoVdom(parentDOM, oldVdom.oldRenderVdom, renderVdom);
+  newVdom.oldRenderVdom = renderVdom;
+}
function updateFunctionComponent(oldVdom, newVdom) {
  let currentDOM = findDOM(oldVdom);
  if (!currentDOM) return;
  let parentDOM = currentDOM.parentNode;
  let { type, props } = newVdom;
  let newRenderVdom = type(props);
  compareTwoVdom(parentDOM, oldVdom.oldRenderVdom, newRenderVdom);
  newVdom.oldRenderVdom = newRenderVdom;
}
function updateClassComponent(oldVdom, newVdom) {
  let classInstance = newVdom.classInstance = oldVdom.classInstance;
  if (classInstance.componentWillReceiveProps) {
    classInstance.componentWillReceiveProps();
  }
  classInstance.updater.emitUpdate(newVdom.props);
}
function updateChildren(parentDOM, oldVChildren, newVChildren) {
  oldVChildren = (Array.isArray(oldVChildren) ? oldVChildren : oldVChildren ? [oldVChildren]).filter(item => item) : [];
  newVChildren = (Array.isArray(newVChildren) ? newVChildren : newVChildren ? [newVChildren]).filter(item => item) : [];
  let keyedOldMap = {};
  let lastPlacedIndex = 0;
  oldVChildren.forEach((oldVChild, index) => {
    let oldKey = oldVChild.key ? oldVChild.key : index;
    keyedOldMap[oldKey] = oldVChild;
  });
  let patch = [];
  newVChildren.forEach((newVChild, index) => {
    newVChild.mountIndex = index;
    let newKey = newVChild.key ? newVChild.key : index;
    let oldVChild = keyedOldMap[newKey];
    if (oldVChild) {
      updateElement(oldVChild, newVChild);
      if (oldVChild.mountIndex < lastPlacedIndex) {
        patch.push({
          type: MOVE,
          oldVChild,
          newVChild,
          mount
        });
      }
      delete keyedOldMap[newKey];
      lastPlacedIndex = Math.max(lastPlacedIndex, oldVChild.mountIndex);
    } else {
      patch.push({
        type: PLACEMENT,
```

```
        newVChild,
        mount
      });
    }
  });
  let moveVChild = patch.filter(action => action.type
  Object.values(keyedOldMap).concat(moveVChild).forEach((oldVChild) => {
    let currentDOM = findDOM(oldVChild);
    parentDOM.removeChild(currentDOM);
  });
  patch.forEach(action => {
    let { type, oldVChild, newVChild, mountIndex } = action;
    let childNodes = parentDOM.childNodes;
    if (type
      let newDOM = createDOM(newVChild);
      let childNode = childNodes[mountIndex];
      if (childNode) {
        parentDOM.insertBefore(newDOM, childNode);
      } else {
        parentDOM.appendChild(newDOM);
      }
    } else if (type
      let oldDOM = findDOM(oldVChild);
      let childNode = childNodes[mountIndex];
      if (childNode) {
        parentDOM.insertBefore(oldDOM, childNode);
      } else {
        parentDOM.appendChild(oldDOM);
      }
    }
  });
}
function reconcileChildren(childrenVdom, parentDOM) {
  for (let i = 0; i < childrenVdom.length; i++) {
    childrenVdom[i].mountIndex = i;
    mount(childrenVdom[i], parentDOM);
  }
}
const ReactDOM = {
  render,
};
export default ReactDOM;
```

13. 高阶组件

- 高阶组件就是一个函数，传给它一个组件，它返回一个新的组件
- 高阶组件的作用其实就是为了组件之间的代码复用

```
const NewComponent = higherOrderComponent(OldComponent)
```

13.1 cra支持装饰器

13.1.1 安装

```
npm i react-app-rewired customize-cra @babel/plugin-proposal-decorators -D
```

13.1.2 修改package.json

```
  "scripts": {
    "start": "react-app-rewired start",
    "build": "react-app-rewired build",
    "test": "react-app-rewired test",
    "eject": "react-app-rewired eject"
  }
```

13.1.3 config-overrides.js

```
const { override, disableEsLint, addDecoratorsLegacy } = require('customize-cra');

module.exports = override(
  disableEsLint(),
  addDecoratorsLegacy()
)
```

13.1.4 jsconfig.json

```
{
  "compilerOptions": {
     "experimentalDecorators": true
  }
}
```

13.2 属性代理

- 基于属性代理：操作组件的props

```
import React from 'react';
import ReactDOM from 'react-dom';
const loading = message =>OldComponent =>{
    return class extends React.Component{
        render(){
            const state = {
                show:()=>{
                   console.log('show', message);
                },
                hide:()=>{
                    console.log('hide', message);
                }
            }
            return  (
                <OldComponent {...this.props} {...state} {...{...this.props,...state}}/>
            )
        }
    }
}
@loading('消息')
class Hello extends React.Component{
  render(){
     return <div>hello<button onClick={this.props.show}>showbutton><button onClick={this.props.hide}>hidebutton>div>;
  }
}
let LoadingHello  = loading('消息')(Hello);

ReactDOM.render(
    <LoadingHello/>, document.getElementById('root'));
```

13.3 反向继承

- 基于反向继承：拦截生命周期、state、渲染过程

```
import React from 'react';
import ReactDOM from 'react-dom';
class Button extends React.Component{
    state = {name:'张三'}
    componentWillMount(){
        console.log('Button componentWillMount');
    }
    componentDidMount(){
        console.log('Button componentDidMount');
    }
    render(){
        console.log('Button render');
        return <button name={this.state.name} title={this.props.title}/>
    }
}
const wrapper = OldComponent =>{
    return class NewComponent extends OldComponent{
        state = {number:0}
        componentWillMount(){
            console.log('WrapperButton componentWillMount');
             super.componentWillMount();
        }
        componentDidMount(){
            console.log('WrapperButton componentDidMount');
             super.componentDidMount();
        }
        handleClick = ()=>{
            this.setState({number:this.state.number+1});
        }
        render(){
            console.log('WrapperButton render');
            let renderElement = super.render();
            let newProps = {
                ...renderElement.props,
                ...this.state,
                onClick:this.handleClick
            }
            return  React.cloneElement(
                renderElement,
                newProps,
                this.state.number
            );
        }
    }
}
let WrappedButton = wrapper(Button);
ReactDOM.render(
    <WrappedButton title="标题"/>, document.getElementById('root'));
```

src\react.js

```
import { wrapToVdom } from "./utils";
import { Component } from './Component';
import { REACT_FORWARD_REF_TYPE, REACT_FRAGMENT,REACT_CONTEXT, REACT_PROVIDER } from './constants';
function createElement(type, config, children) {
  let ref;
  let key;
  if (config) {
    delete config.__source;
    delete config.__self;
    ref = config.ref;
    delete config.ref;
    key = config.key;
    delete config.key;
  }
  let props = { ...config };
  if (arguments.length > 3) {
    props.children = Array.prototype.slice.call(arguments, 2).map(wrapToVdom);
  } else {
    props.children = wrapToVdom(children);
  }
  return {
    type,
    ref,
    key,
    props,
  };
}
function createRef() {
  return { current: null };
}
function forwardRef(render) {
  var elementType = {
    $typeof: REACT_FORWARD_REF_TYPE,
    render: render
  };
  return elementType;
}
function createContext() {
  let context = { $typeof: REACT_CONTEXT };
  context.Provider = {
    $typeof: REACT_PROVIDER,
    _context: context
  }
  context.Consumer = {
    $typeof: REACT_CONTEXT,
    _context: context
  }
  return context;
}
+function cloneElement(element, newProps, ...newChildren) {
+  let oldChildren = element.props && element.props.children;
+  let children = [...(Array.isArray(oldChildren) ? oldChildren : [oldChildren]), ...newChildren]
+    .filter(item => item !== undefined)
+    .map(wrapToVdom);
+  if (children.length === 1) children = children[0];
+  let props = { ...element.props, ...newProps, children };
+  return { ...element, props };
+}
const React = {
  createElement,
  Component,
  createRef,
  forwardRef,
  Fragment:REACT_FRAGMENT,
  createContext,
+ cloneElement
};
export default React;
```

14. render props

- render-props (https://zh-hans.reactjs.org/docs/render-props.html)
- `render prop` 是指一种在 React 组件之间使用一个值为函数的 prop 共享代码的简单技术
- 具有 render prop 的组件接受一个函数，该函数返回一个 React 元素并调用它而不是实现自己的渲染逻辑
- render prop 是一个用于告知组件需要渲染什么内容的函数 prop
- 这也是逻辑复用的一种方式

14.1 原生实现

```
import React from 'react';
import ReactDOM from 'react-dom';
class MouseTracker extends React.Component {
    constructor(props) {
        super(props);
        this.state = { x: 0, y: 0 };
    }

    handleMouseMove = (event) => {
        this.setState({
            x: event.clientX,
            y: event.clientY
        });
    }

    render() {
        return (
            <div onMouseMove={this.handleMouseMove}>
                <h1>移动鼠标!h1>
                <p>当前的鼠标位置是 ({this.state.x}, {this.state.y})p>
            div>
        );
    }
}
ReactDOM.render(<MouseTracker />, document.getElementById('root'));
```

14.2 children

- children是一个渲染的方法

```
import React from './react';
import ReactDOM from './react-dom';

class MouseTracker extends React.Component {
    constructor(props) {
        super(props);
        this.state = { x: 0, y: 0 };
    }

    handleMouseMove = (event) => {
        this.setState({
            x: event.clientX,
            y: event.clientY
        });
    }

    render() {
        return (
            <div onMouseMove={this.handleMouseMove}>
                {this.props.children(this.state)}
            div>
        );
    }
}
ReactDOM.render(<MouseTracker >
    {
        (props) => (
            <div>
                <h1>移动鼠标!h1>
                <p>当前的鼠标位置是 ({props.x}, {props.y})p>
            div>
        )
    }
MouseTracker >, document.getElementById('root'));
```

14.3 render属性

```
import React from 'react';
import ReactDOM from 'react-dom';
class MouseTracker extends React.Component {
    constructor(props) {
        super(props);
        this.state = { x: 0, y: 0 };
    }

    handleMouseMove = (event) => {
        this.setState({
            x: event.clientX,
            y: event.clientY
        });
    }

    render() {
        return (
            <div onMouseMove={this.handleMouseMove}>
                {this.props.render(this.state)}
            div>
        );
    }
}

ReactDOM.render(< MouseTracker render={params => (
    <>
        <h1>移动鼠标!h1>
        <p>当前的鼠标位置是 ({params.x}, {params.y})p>
    </>
)} />, document.getElementById('root'));
```

14.4 HOC

```
import React from 'react';
import ReactDOM from 'react-dom';
function withTracker(OldComponent){
  return class MouseTracker extends React.Component{
    constructor(props){
        super(props);
        this.state = {x:0,y:0};
    }
    handleMouseMove = (event)=>{
        this.setState({
            x:event.clientX,
            y:event.clientY
        });
    }
    render(){
        return (
            <div onMouseMove = {this.handleMouseMove}>
               <OldComponent {...this.state}/>
            div>
        )
    }
 }
}

function Show(props){
    return (
        <React.Fragment>
          <h1>请移动鼠标h1>
          <p>当前鼠标的位置是: x:{props.x} y:{props.y}p>
        React.Fragment>
    )
}
let HighShow = withTracker(Show);
ReactDOM.render(
    <HighShow/>, document.getElementById('root'));
```

15.性能优化

15.1 src\index.js

```
import React from './react';
import ReactDOM from './react-dom';
class ClassCounter extends React.PureComponent {
    render() {
        console.log('ClassCounter render');
        return <div>ClassCounter:{this.props.count}div>
    }
}
function FunctionCounter(props) {
    console.log('FunctionCounter render'); debugger
    return <div>FunctionCounter:{props.count}div>
}
const MemoFunctionCounter = React.memo(FunctionCounter);
class App extends React.Component {
    state = { number: 0 }
    amountRef = React.createRef()
    handleClick = () => {
        let nextNumber = this.state.number + parseInt(this.amountRef.current.value);
        this.setState({ number: nextNumber });
    }
    render() {
        return (
            <div>
                <ClassCounter count={this.state.number} />
                <MemoFunctionCounter count={this.state.number} />
                <input ref={this.amountRef} />
                <button onClick={this.handleClick}>+button>
            div>
        )
    }
}
ReactDOM.render(
    <App />, document.getElementById('root'));
```

15.2 src\constants.js

src\constants.js

```
export const REACT_TEXT = Symbol('REACT_TEXT');
export const REACT_FORWARD_REF_TYPE = Symbol('react.forward_ref');

export const PLACEMENT = 'PLACEMENT';
export const MOVE = 'MOVE';

export const REACT_CONTEXT = Symbol('react.context');
export const REACT_PROVIDER = Symbol('react.provider');
+export const REACT_MEMO = Symbol('react.memo')
```

15.3 src\utils.js

src\utils.js

```
import { REACT_TEXT } from "./constants";
export function wrapToVdom(element) {
  return typeof element
    ? { type: REACT_TEXT, props: element }
    : element;
}

export function isFunction(obj) {
  return typeof obj
}

+export function shallowEqual(obj1, obj2) {
+  if (obj1 === obj2) {
+    return true;
+  }
+  if (typeof obj1 != "object" || obj1 === null || typeof obj2 != "object" || obj2 === null) {
+    return false;
+  }
+  let keys1 = Object.keys(obj1);
+  let keys2 = Object.keys(obj2);
+  if (keys1.length !== keys2.length) {
+    return false;
+  }
+  for (let key of keys1) {
+    if (!obj2.hasOwnProperty(key) || obj1[key] !== obj2[key]) {
+      return false;
+    }
+  }
+  return true;
+}
```

15.4 src\react.js

src\react.js

```
+import { wrapToVdom, shallowEqual } from "./utils";
import { Component } from './Component';
+import { REACT_FORWARD_REF_TYPE,REACT_FRAGMENT, REACT_CONTEXT, REACT_PROVIDER, REACT_MEMO } from './constants';
function createElement(type, config, children) {
  let ref;
  let key;
  if (config) {
    delete config.__source;
    delete config.__self;
    ref = config.ref;
    delete config.ref;
    key = config.key;
    delete config.key;
  }
  let props = { ...config };
  if (arguments.length > 3) {
    props.children = Array.prototype.slice.call(arguments, 2).map(wrapToVdom);
  } else {
    props.children = wrapToVdom(children);
  }
  return {
    type,
    ref,
    key,
    props,
  };
}
function createRef() {
  return { current: null };
}
function forwardRef(render) {
  var elementType = {
    $typeof: REACT_FORWARD_REF_TYPE,
    render: render
  };
  return elementType;
}
function createContext() {
  let context = { $typeof: REACT_CONTEXT };
  context.Provider = {
    $typeof: REACT_PROVIDER,
    _context: context
  }
  context.Consumer = {
    $typeof: REACT_CONTEXT,
    _context: context
  }
  return context;
}
function cloneElement(element, newProps, ...newChildren) {
  let oldChildren = element.props && element.props.children;
  let children = [...(Array.isArray(oldChildren) ? oldChildren : [oldChildren]), ...newChildren]
    .filter(item => item !== undefined)
    .map(wrapToVdom);
  if (children.length
  let props = { ...element.props, ...newProps, children };
  return { ...element, props };
}
+class PureComponent extends Component {
+  shouldComponentUpdate(newProps, nextState) {
+    return !shallowEqual(this.props, newProps) || !shallowEqual(this.state, nextState);
+  }
+}
+function memo(type, compare = shallowEqual) {
+  return {
+    $typeof: REACT_MEMO,
+    type,
+    compare
+  }
+}
const React = {
  createElement,
  Component,
  createRef,
  forwardRef,
  Fragment:REACT_FRAGMENT,
  createContext,
  cloneElement,
  PureComponent,
+ memo
};
export default React;
```

15.5 src\react-dom.js

src\react-dom.js

```
+import { REACT_TEXT, REACT_FORWARD_REF_TYPE, PLACEMENT, MOVE, REACT_FRAGMENT,REACT_PROVIDER, REACT_CONTEXT, REACT_MEMO } from "./constants";
import { addEvent } from "./event";
import React from './react';
function render(vdom, parentDOM) {
    let newDOM = createDOM(vdom)
    if (newDOM) {
        parentDOM.appendChild(newDOM);
        if (newDOM._componentDidMount) newDOM._componentDidMount();
    }
}
export function createDOM(vdom) {
  let { type, props, ref } = vdom;
  let dom;
+  if (type && type.$typeof === REACT_MEMO) {
+    return mountMemoComponent(vdom);
+  } else if (type && type.$typeof === REACT_PROVIDER) {
    return mountProviderComponent(vdom)
```

```
  } else if (type && type.$typeof
    return mountContextComponent(vdom)
  } else if (type && type.$typeof
    return mountForwardComponent(vdom);
  } else if (type
    dom = document.createTextNode(props);
  } else if (type
    dom = document.createDocumentFragment();
  } else if (typeof type
    if (type.isReactComponent) {
      return mountClassComponent(vdom);
    } else {
       return mountFunctionComponent(vdom);
    }
  } else {
    dom = document.createElement(type);
  }
  if (props) {
    updateProps(dom, {}, props);
    if (typeof props.children == "object" && props.children.type) {
      props.children.mountIndex = 0;
      mount(props.children, dom);
    } else if (Array.isArray(props.children)) {
      reconcileChildren(props.children, dom);
    }
  }
  vdom.dom = dom;
  if (ref) ref.current = dom;
  return dom;
}

+function mountMemoComponent(vdom) {
+  let { type, props } = vdom;
+  let renderVdom = type.type(props);
+  vdom.oldRenderVdom = renderVdom;
+  return createDOM(renderVdom);
+}
function mountProviderComponent(vdom) {
  let { type, props } = vdom;
  let context = type._context;
  context._currentValue = props.value;
  let renderVdom = props.children;
  vdom.oldRenderVdom = renderVdom;
  return createDOM(renderVdom);
}
function mountContextComponent(vdom) {
  let { type, props } = vdom;
  let context = type._context;
  let renderVdom = props.children(context._currentValue);
  vdom.oldRenderVdom = renderVdom;
  return createDOM(renderVdom);
}
function mountForwardComponent(vdom) {
  let { type, props, ref } = vdom;
  let renderVdom = type.render(props, ref);
  vdom.oldRenderVdom = renderVdom;
  return createDOM(renderVdom);
}
function mountClassComponent(vdom) {
  let { type, props, ref } = vdom;
  let classInstance = new type(props);
  if (type.contextType) {
    classInstance.context = type.contextType._currentValue;
  }
  vdom.classInstance = classInstance;
  if (ref) ref.current = classInstance;
  if (classInstance.componentWillMount) classInstance.componentWillMount();
  let renderVdom = classInstance.render();
  classInstance.oldRenderVdom  = renderVdom;
  let dom = createDOM(renderVdom);
  if (classInstance.componentDidMount)
    dom.componentDidMount = classInstance.componentDidMount.bind(classInstance);
  return dom;
}
function mountFunctionComponent(vdom) {
  let { type, props } = vdom;
  let renderVdom = type(props);
  vdom.oldRenderVdom = renderVdom;
  return createDOM(renderVdom);
}
function updateProps(dom, oldProps={}, newProps={}) {
    for (let key in newProps) {
        if (key
            continue;
        } else if (key
            let styleObj = newProps[key];
            for (let attr in styleObj) {
                dom.style[attr] = styleObj[attr];
            }
        } else if (key.startsWith('on')) {
            addEvent(dom, key.toLocaleLowerCase(), newProps[key]);
        } else {
            dom[key] = newProps[key];
        }
    }
    for(let key in oldProps){
        if(!newProps.hasOwnProperty(key)){
            dom[key] = null;
        }
    }
}
export function findDOM(vdom) {
    if (!vdom) return null;
    if (vdom.dom) {//vdom={type:'h1'}
```

```
        return vdom.dom;
    } else {
        let renderVdom = vdom.classInstance ? vdom.classInstance.oldRenderVdom : vdom.oldRenderVdom;
        return findDOM(renderVdom);
    }
}
function unMountVdom(vdom) {
    let { type, props, ref } = vdom;
    let currentDOM = findDOM(vdom);//获取此虚拟DOM对应的真实DOM
    //vdom可能是原生组件span 类组件 classComponent 也可能是函数组件Function
    if (vdom.classInstance && vdom.classInstance.componentWillUnmount) {
        vdom.classInstance.componentWillUnmount();
    }
    if (ref) {
        ref.current = null;
    }
    //如果此虚拟DOM有子节点的话，递归全部删除
    if (props.children) {
        //得到儿子的数组
        let children = Array.isArray(props.children) ? props.children : [props.children];
        children.forEach(unMountVdom);
    }
    //把自己这个虚拟DOM对应的真实DOM从界面删除
    if (currentDOM) currentDOM.remove();
}
export function compareTwoVdom(parentDOM, oldVdom, newVdom, nextDOM) {
  if (!oldVdom && !newVdom) {
    //老和新都是没有
    return;
  } else if (!!oldVdom && !newVdom) {
    //老有新没有
    unMountVdom(oldVdom);
  } else if (!oldVdom && !!newVdom) {
    //老没有新的有
    let newDOM = createDOM(newVdom);
    if (nextDOM) parentDOM.insertBefore(newDOM, nextDOM);
    else parentDOM.appendChild(newDOM);
    if (newDOM.componentDidMount) newDOM.componentDidMount();
    return;
  } else if (!!oldVdom && !!newVdom && oldVdom.type !== newVdom.type) {
    //新老都有，但类型不同
    let newDOM = createDOM(newVdom);
    unMountVdom(oldVdom);
    if (newDOM.componentDidMount) newDOM.componentDidMount();
  } else {
    updateElement(oldVdom, newVdom);
  }
}
function updateElement(oldVdom, newVdom) {
+  if (oldVdom.type && oldVdom.type.$typeof === REACT_MEMO) {
+    updateMemoComponent(oldVdom, newVdom);
  } else if (oldVdom.type.$typeof
    updateContextComponent(oldVdom, newVdom);
  } else if (oldVdom.type.$typeof
    updateProviderComponent(oldVdom, newVdom);
  } else if (oldVdom.type
    let currentDOM = newVdom.dom = findDOM(oldVdom);
    if (oldVdom.props !== newVdom.props) {
      currentDOM.textContent = newVdom.props;
    }
    return;
  }  else if (oldVdom.type
    let currentDOM = newVdom.dom = findDOM(oldVdom);
    updateChildren(currentDOM, oldVdom.props.children, newVdom.props.children);
  }else if (typeof oldVdom.type
    let currentDOM = newVdom.dom = findDOM(oldVdom);
    updateProps(currentDOM, oldVdom.props, newVdom.props);
    updateChildren(currentDOM, oldVdom.props.children, newVdom.props.children);
  } else if (typeof oldVdom.type
    if (oldVdom.type.isReactComponent) {
      updateClassComponent(oldVdom, newVdom);
    } else {
      updateFunctionComponent(oldVdom, newVdom);
    }
  }
}
+function updateMemoComponent(oldVdom, newVdom) {
+ let { type } = oldVdom;
+ if (!type.compare(oldVdom.props, newVdom.props)) {
+   const oldDOM = findDOM(oldVdom);
+   const parentDOM = oldDOM.parentNode;
+   const { type } = newVdom;
+   let renderVdom = type.type(newVdom.props);
+   compareTwoVdom(parentDOM, oldVdom.oldRenderVdom, renderVdom);
+   newVdom.oldRenderVdom = renderVdom;
+ } else {
+   newVdom.oldRenderVdom = oldVdom.oldRenderVdom;
+ }
+}
function updateProviderComponent(oldVdom, newVdom) {
  let parentDOM = findDOM(oldVdom).parentNode;
  let { type, props } = newVdom;
  let context = type._context;
  context._currentValue = props.value;
  let renderVdom = props.children;
  compareTwoVdom(parentDOM, oldVdom.oldRenderVdom, renderVdom);
  newVdom.oldRenderVdom = renderVdom;
}
function updateContextComponent(oldVdom, newVdom) {
  let parentDOM = findDOM(oldVdom).parentNode;
  let { type, props } = newVdom;
  let context = type._context;
  let renderVdom = props.children(context._currentValue);
  compareTwoVdom(parentDOM, oldVdom.oldRenderVdom, renderVdom);
```

```
  newVdom.oldRenderVdom = renderVdom;
}
function updateFunctionComponent(oldVdom, newVdom) {
  let currentDOM = findDOM(oldVdom);
  if (!currentDOM) return;
  let parentDOM = currentDOM.parentNode;
  let { type, props } = newVdom;
  let newRenderVdom = type(props);
  compareTwoVdom(parentDOM, oldVdom.oldRenderVdom, newRenderVdom);
  newVdom.oldRenderVdom = newRenderVdom;
}
function updateClassComponent(oldVdom, newVdom) {
  let classInstance = newVdom.classInstance = oldVdom.classInstance;
  if (classInstance.componentWillReceiveProps) {
    classInstance.componentWillReceiveProps();
  }
  classInstance.updater.emitUpdate(newVdom.props);
}
function updateChildren(parentDOM, oldVChildren, newVChildren) {
  oldVChildren = (Array.isArray(oldVChildren) ? oldVChildren : oldVChildren ? [oldVChildren]).filter(item => item) : [];
  newVChildren = (Array.isArray(newVChildren) ? newVChildren : newVChildren ? [newVChildren]).filter(item => item) : [];
  let keyedOldMap = {};
  let lastPlacedIndex = 0;
  oldVChildren.forEach((oldVChild, index) => {
    let oldKey = oldVChild.key ? oldVChild.key : index;
    keyedOldMap[oldKey] = oldVChild;
  });
  let patch = [];
  newVChildren.forEach((newVChild, index) => {
    newVChild.mountIndex = index;
    let newKey = newVChild.key ? newVChild.key : index;
    let oldVChild = keyedOldMap[newKey];
    if (oldVChild) {
      updateElement(oldVChild, newVChild);
      if (oldVChild.mountIndex < lastPlacedIndex) {
        patch.push({
          type: MOVE,
          oldVChild,
          newVChild,
          mount
        });
      }
      delete keyedOldMap[newKey];
      lastPlacedIndex = Math.max(lastPlacedIndex, oldVChild.mountIndex);
    } else {
      patch.push({
        type: PLACEMENT,
        newVChild,
        mount
      });
    }
  });
  let moveVChild = patch.filter(action => action.type
  Object.values(keyedOldMap).concat(moveVChild).forEach((oldVChild) => {
    let currentDOM = findDOM(oldVChild);
    parentDOM.removeChild(currentDOM);
  });
  patch.forEach(action => {
    let { type, oldVChild, newVChild, mountIndex } = action;
    let childNodes = parentDOM.childNodes;
    if (type
      let newDOM = createDOM(newVChild);
      let childNode = childNodes[mountIndex];
      if (childNode) {
        parentDOM.insertBefore(newDOM, childNode);
      } else {
        parentDOM.appendChild(newDOM);
      }
    } else if (type
      let oldDOM = findDOM(oldVChild);
      let childNode = childNodes[mountIndex];
      if (childNode) {
        parentDOM.insertBefore(oldDOM, childNode);
      } else {
        parentDOM.appendChild(oldDOM);
      }
    }
  });
}
function reconcileChildren(childrenVdom, parentDOM) {
  for (let i = 0; i < childrenVdom.length; i++) {
    childrenVdom[i].mountIndex = i;
    mount(childrenVdom[i], parentDOM);
  }
}
const ReactDOM = {
  render,
};
export default ReactDOM;
```

16.Portal

- React v16增加了对Portal的直接支持
- 它可以把JSX渲染到一个单独的DOM节点中

16.1 src\index.js

src\index.js

```
import React from './react';
import ReactDOM from './react-dom';
class Dialog extends React.Component {
    constructor(props) {
        super(props);
        this.node = document.createElement('div');
        document.body.appendChild(this.node);
    }
    render() {
        return ReactDOM.createPortal(
            <div className="dialog">
                {this.props.children}
            div>,
            this.node
        );
    }
    componentWillUnmount() {
        window.document.body.removeChild(this.node);
    }
}
class App extends React.Component {
    render() {
        return (
            <div>
                <Dialog>模态窗Dialog>
            div>
        )
    }
}
ReactDOM.render(
    <App />, document.getElementById('root'));
```

16.2 src\react-dom.js

src\react-dom.js

```
import { REACT_TEXT, REACT_FORWARD_REF_TYPE, PLACEMENT, MOVE, REACT_FRAGMENT,REACT_PROVIDER, REACT_CONTEXT, REACT_MEMO } from "./constants";
import { addEvent } from "./event";
import React from './react';
function render(vdom, parentDOM) {
    let newDOM = createDOM(vdom)
    if (newDOM) {
        parentDOM.appendChild(newDOM);
        if (newDOM._componentDidMount) newDOM._componentDidMount();
    }
}
export function createDOM(vdom) {
  let { type, props, ref } = vdom;
  let dom;
  if (type && type.$typeof
    return mountMemoComponent(vdom);
  } else if (type && type.$typeof
    return mountProviderComponent(vdom)
  } else if (type && type.$typeof
    return mountContextComponent(vdom)
  } else if (type && type.$typeof
    return mountForwardComponent(vdom);
  } else if (type
    dom = document.createTextNode(props);
  }else if (type
    dom = document.createDocumentFragment();
  } else if (typeof type
    if (type.isReactComponent) {
      return mountClassComponent(vdom);
    } else {
      return mountFunctionComponent(vdom);
    }
  } else {
    dom = document.createElement(type);
  }
  if (props) {
    updateProps(dom, {}, props);
    if (typeof props.children == "object" && props.children.type) {
      props.children.mountIndex = 0;
      mount(props.children, dom);
    } else if (Array.isArray(props.children)) {
      reconcileChildren(props.children, dom);
    }
  }
  vdom.dom = dom;
  if (ref) ref.current = dom;
  return dom;
}

function mountMemoComponent(vdom) {
  let { type, props } = vdom;
  let renderVdom = type.type(props);
  vdom.oldRenderVdom = renderVdom;
+ if (!renderVdom) return null;
  return createDOM(renderVdom);
}
function mountProviderComponent(vdom) {
  let { type, props } = vdom;
  let context = type._context;
  context._currentValue = props.value;
  let renderVdom = props.children;
  vdom.oldRenderVdom = renderVdom;
+ if (!renderVdom) return null;
  return createDOM(renderVdom);
}
function mountContextComponent(vdom) {
  let { type, props } = vdom;
  let context = type._context;
  let renderVdom = props.children(context._currentValue);
```

```
  vdom.oldRenderVdom = renderVdom;
+ if (!renderVdom) return null;
  return createDOM(renderVdom);
}
function mountForwardComponent(vdom) {
  let { type, props, ref } = vdom;
  let renderVdom = type.render(props, ref);
  vdom.oldRenderVdom = renderVdom;
+ if (!renderVdom) return null;
  return createDOM(renderVdom);
}
function mountClassComponent(vdom) {
  let { type, props, ref } = vdom;
  let classInstance = new type(props);
  if (type.contextType) {
    classInstance.context = type.contextType._currentValue;
  }
  vdom.classInstance = classInstance;
  if (ref) ref.current = classInstance;
  if (classInstance.componentWillMount) classInstance.componentWillMount();
  let renderVdom = classInstance.render();
  classInstance.oldRenderVdom  = renderVdom;
+ if (!renderVdom) return null;
  let dom = createDOM(renderVdom);
  if (classInstance.componentDidMount)
    dom.componentDidMount = classInstance.componentDidMount.bind(classInstance);
  return dom;
}
function mountFunctionComponent(vdom) {
  let { type, props } = vdom;
  let renderVdom = type(props);
  vdom.oldRenderVdom = renderVdom;
+ if (!renderVdom) return null;
  return createDOM(renderVdom);
}
function updateProps(dom, oldProps={}, newProps={}) {
    for (let key in newProps) {
        if (key
            continue;
        } else if (key
            let styleObj = newProps[key];
            for (let attr in styleObj) {
                dom.style[attr] = styleObj[attr];
            }
        } else if (key.startsWith('on')) {
            addEvent(dom, key.toLocaleLowerCase(), newProps[key]);
        } else {
            dom[key] = newProps[key];
        }
    }
    for(let key in oldProps){
        if(!newProps.hasOwnProperty(key)){
            dom[key] = null;
        }
    }
}
export function findDOM(vdom) {
    if (!vdom) return null;
    if (vdom.dom) {//vdom={type:'h1'}
        return vdom.dom;
    } else {
        let renderVdom = vdom.classInstance ? vdom.classInstance.oldRenderVdom : vdom.oldRenderVdom;
        return findDOM(renderVdom);
    }
}
function unMountVdom(vdom) {
    let { type, props, ref } = vdom;
    let currentDOM = findDOM(vdom);//获取此虚拟DOM对应的真实DOM
    //vdom可能是原生组件span 类组件 classComponent 也可能是函数组件Function
    if (vdom.classInstance && vdom.classInstance.componentWillUnmount) {
        vdom.classInstance.componentWillUnmount();
    }
    if (ref) {
        ref.current = null;
    }
    //如果此虚拟DOM有子节点的话，递归全部删除
    if (props.children) {
        //得到儿子的数组
        let children = Array.isArray(props.children) ? props.children : [props.children];
        children.forEach(unMountVdom);
    }
    //把自己这个虚拟DOM对应的真实DOM从界面删除
    if (currentDOM) currentDOM.remove();
}
export function compareTwoVdom(parentDOM, oldVdom, newVdom, nextDOM) {
  if (!oldVdom && !newVdom) {
    //老和新都是没有
    return;
  } else if (!!oldVdom && !newVdom) {
    //老有新没有
    unMountVdom(oldVdom);
  } else if (!oldVdom && !!newVdom) {
    //老没有新的有
    let newDOM = createDOM(newVdom);
    if (nextDOM) parentDOM.insertBefore(newDOM, nextDOM);
    else parentDOM.appendChild(newDOM);
    if (newDOM.componentDidMount) newDOM.componentDidMount();
    return;
  } else if (!!oldVdom && !!newVdom && oldVdom.type !== newVdom.type) {
    //新老都有，但类型不同
    let newDOM = createDOM(newVdom);
    unMountVdom(oldVdom);
    if (newDOM.componentDidMount) newDOM.componentDidMount();
  } else {
```

```
    updateElement(oldVdom, newVdom);
  }
}
function updateElement(oldVdom, newVdom) {
  if (oldVdom.type && oldVdom.type.$typeof
    updateMemoComponent(oldVdom, newVdom);
  } else if (oldVdom.type.$typeof
    updateContextComponent(oldVdom, newVdom);
  } else if (oldVdom.type.$typeof
    updateProviderComponent(oldVdom, newVdom);
  } else if (oldVdom.type
    let currentDOM = newVdom.dom = findDOM(oldVdom);
    if (oldVdom.props !== newVdom.props) {
      currentDOM.textContent = newVdom.props;
    }
    return;
  } else if (typeof oldVdom.type
    let currentDOM = newVdom.dom = findDOM(oldVdom);
    updateProps(currentDOM, oldVdom.props, newVdom.props);
    updateChildren(currentDOM, oldVdom.props.children, newVdom.props.children);
  } else if (oldVdom.type
    let currentDOM = newVdom.dom = findDOM(oldVdom);
    updateChildren(currentDOM, oldVdom.props.children, newVdom.props.children);
  } else if (typeof oldVdom.type
    if (oldVdom.type.isReactComponent) {
      updateClassComponent(oldVdom, newVdom);
    } else {
       updateFunctionComponent(oldVdom, newVdom);
    }
  }
}
function updateMemoComponent(oldVdom, newVdom) {
  let { type } = oldVdom;
  if (!type.compare(oldVdom.props, newVdom.props)) {
    const oldDOM = findDOM(oldVdom);
    const parentDOM = oldDOM.parentNode;
    const { type } = newVdom;
    let renderVdom = type.type(newVdom.props);
    compareTwoVdom(parentDOM, oldVdom.oldRenderVdom, renderVdom);
    newVdom.oldRenderVdom = renderVdom;
  } else {
    newVdom.oldRenderVdom = oldVdom.oldRenderVdom;
  }
}
function updateProviderComponent(oldVdom, newVdom) {
  let parentDOM = findDOM(oldVdom).parentNode;
  let { type, props } = newVdom;
  let context = type._context;
  context._currentValue = props.value;
  let renderVdom = props.children;
  compareTwoVdom(parentDOM, oldVdom.oldRenderVdom, renderVdom);
  newVdom.oldRenderVdom = renderVdom;
}
function updateContextComponent(oldVdom, newVdom) {
  let parentDOM = findDOM(oldVdom).parentNode;
  let { type, props } = newVdom;
  let context = type._context;
  let renderVdom = props.children(context._currentValue);
  compareTwoVdom(parentDOM, oldVdom.oldRenderVdom, renderVdom);
  newVdom.oldRenderVdom = renderVdom;
}
function updateFunctionComponent(oldVdom, newVdom) {
  let currentDOM = findDOM(oldVdom);
  if (!currentDOM) return;
  let parentDOM = currentDOM.parentNode;
  let { type, props } = newVdom;
  let newRenderVdom = type(props);
  compareTwoVdom(parentDOM, oldVdom.oldRenderVdom, newRenderVdom);
  newVdom.oldRenderVdom = newRenderVdom;
}
function updateClassComponent(oldVdom, newVdom) {
  let classInstance = newVdom.classInstance = oldVdom.classInstance;
  if (classInstance.componentWillReceiveProps) {
    classInstance.componentWillReceiveProps();
  }
  classInstance.updater.emitUpdate(newVdom.props);
}
function updateChildren(parentDOM, oldVChildren, newVChildren) {
  oldVChildren = (Array.isArray(oldVChildren) ? oldVChildren : oldVChildren ? [oldVChildren]).filter(item => item) : [];
  newVChildren = (Array.isArray(newVChildren) ? newVChildren : newVChildren ? [newVChildren]).filter(item => item) : [];
  let keyedOldMap = {};
  let lastPlacedIndex = 0;
  oldVChildren.forEach((oldVChild, index) => {
    let oldKey = oldVChild.key ? oldVChild.key : index;
    keyedOldMap[oldKey] = oldVChild;
  });
  let patch = [];
  newVChildren.forEach((newVChild, index) => {
    newVChild.mountIndex = index;
    let newKey = newVChild.key ? newVChild.key : index;
    let oldVChild = keyedOldMap[newKey];
    if (oldVChild) {
      updateElement(oldVChild, newVChild);
      if (oldVChild.mountIndex < lastPlacedIndex) {
        patch.push({
          type: MOVE,
          oldVChild,
          newVChild,
          mount
        });
      }
      delete keyedOldMap[newKey];
      lastPlacedIndex = Math.max(lastPlacedIndex, oldVChild.mountIndex);
    } else {
```

```
      patch.push({
        type: PLACEMENT,
        newVChild,
        mount
      });
    }
  });
  let moveVChild = patch.filter(action => action.type
  Object.values(keyedOldMap).concat(moveVChild).forEach((oldVChild) => {
    let currentDOM = findDOM(oldVChild);
    parentDOM.removeChild(currentDOM);
  });
  patch.forEach(action => {
    let { type, oldVChild, newVChild, mountIndex } = action;
    let childNodes = parentDOM.childNodes;
    if (type
      let newDOM = createDOM(newVChild);
      let childNode = childNodes[mountIndex];
      if (childNode) {
        parentDOM.insertBefore(newDOM, childNode);
      } else {
        parentDOM.appendChild(newDOM);
      }
    } else if (type
      let oldDOM = findDOM(oldVChild);
      let childNode = childNodes[mountIndex];
      if (childNode) {
        parentDOM.insertBefore(oldDOM, childNode);
      } else {
        parentDOM.appendChild(oldDOM);
      }
    }
  });
}
function reconcileChildren(childrenVdom, parentDOM) {
  for (let i = 0; i < childrenVdom.length; i++) {
    childrenVdom[i].mountIndex = i;
    mount(childrenVdom[i], parentDOM);
  }
}
const ReactDOM = {
  render,
  createPortal:render
};
export default ReactDOM;
```

zhufengreact (git@gitee.com:zhufengpeixun/zhufengreact.git)