---

link: null
title: 珠峰架构师成长计划
description: createElement的结果
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=203 sentences=572, words=4083

---

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. 组件化的概念

- 我们可以很直观的将一个复杂的页面分割成若干个独立组件,每个组件包含自己的逻辑和样式 再将这些独立组件组合完成一个复杂的页面。 这样既减少了逻辑复杂度，又实现了代码的重用
    - 可组合：一个组件可以和其他的组件一起使用或者可以直接嵌套在另一个组件内部
    - 可重用：每个组件都是具有独立功能的，它可以被使用在多个场景中
    - 可维护：每个小的组件仅仅包含自身的逻辑，更容易被理解和维护

3.搭建React开发环境

```
cnpm i create-react-app -g
create-react-app zhufeng2019react --typescript
cd zhufeng2019react
npm start
```

包名 用途
react (https://www.npmjs.com/package/react)

React is a JavaScript library for creating user interfaces
@types/react (https://www.npmjs.com/package/@types/react)

This package contains type definitions for React (
http://facebook.github.io/react/ (http://facebook.github.io/react/)) react-dom (https://www.npmjs.com/package/react-dom)

This package serves as the entry point to the DOM and server renderers for React. It is intended to be paired with the generic React package, which is shipped as react to npm
@types/react-dom (https://www.npmjs.com/package/@types/react-dom)

This package contains type definitions for React (react-dom) (
http://facebook.github.io/react/ (http://facebook.github.io/react/)) react-scripts (https://www.npmjs.com/package/react-scripts)

This package includes scripts and configuration used by Create React App typescript TypeScript is a language for application-scale JavaScript
@types/jest (https://www.npmjs.com/package/@types/jest)

This package contains type definitions for Jest (
https://jestjs.io/ (https://jestjs.io/)) @types/node (https://www.npmjs.com/package/@types/node)

This package contains type definitions for Node.js (
http://nodejs.org/ (http://nodejs.org/))

4.JSX

4.1 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定义组件

```
ReactDOM.render(
  <h1>Helloh1>,
  document.getElementById('root')
);
```

4.2 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 `createElement`语法
- React元素是构成 `React`应用的最小单位
- React元素用来描述你在屏幕上看到的内容
- React元素事实上是普通的JS对象,ReactDOM来确保浏览器中的DOM数据和React元素保持一致

```
hello
```

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

createElement的结果

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

4.3 JSX表达式

- 可以任意地在 JSX 当中使用 JavaScript 表达式，在 JSX 当中的表达式要包含在大括号里

```
import React from 'react';
import ReactDOM from 'react-dom';
let title: string = 'hello';
let root: HTMLElement | null = document.getElementById('root');
ReactDOM.render(
    <h1>{title}h1>,
    root
);
```

4.4 JSX属性

- 需要注意的是JSX并不是 HTML,更像 JavaScript
- 在JSX中属性不能包含关键字，像 class需要写成 className, for需要写成 htmlFor,并且属性名需要采用驼峰命名法

```
import React from 'react';
import ReactDOM from 'react-dom';
let title: string = 'hello';
let root: HTMLElement | null = document.getElementById('root');

ReactDOM.render(
    Hello,
    document.getElementById('root')
);
 */
```

4.5 JSX也是对象

- 可以在 if或者 for语句里使用 JSX
- 将它赋值给变量，当作参数传入，作为返回值都可以

if中使用

```
import React from 'react';
import ReactDOM from 'react-dom';
let root: HTMLElement | null = document.getElementById('root');
function greeting(name: string): React.ReactElement {
    if (name) {
        return <h1>Hello, {name}!h1>;
    }
    return <h1>Hello, Stranger.h1>;
}

const element: React.ReactElement = greeting('zhufeng');

ReactDOM.render(
    element,
    root
);
```

for中使用

```
import React from 'react';
import ReactDOM from 'react-dom';
let root: HTMLElement | null = document.getElementById('root');
let names: Array = ['张三', '李四', '王五'];
let elements: Array = [];
for (let i = 0; i < names.length; i++) {
    elements.push(<li>{names[i]}li>);
}
ReactDOM.render(
    <ul>
        {elements}
    ul>,
    root
);
```

4.6 更新元素渲染

- React 元素都是 immutable不可变的。当元素被创建之后，你是无法改变其内容或属性的。一个元素就好像是动画里的一帧，它代表应用界面在某一时间点的样子
- 更新界面的唯一办法是创建一个新的元素，然后将它传入 ReactDOM.render()方法

```
import React from 'react';
import ReactDOM from 'react-dom';
let root: HTMLElement | null = document.getElementById('root');
function tick(): void {
    const element: React.ReactElement = (
        <div>
            {new Date().toLocaleTimeString()}
        div>
    );
    ReactDOM.render(element, root);
}
setInterval(tick, 1000);
```

4.7 React只会更新必要的部分

- React DOM 首先会比较元素内容先后的不同，而在渲染过程中只会更新改变了的部分。
- 即便我们每秒都创建了一个描述整个UI树的新元素，React DOM 也只会更新渲染文本节点中发生变化的内容

5. 组件 & Props

- 可以将UI切分成一些独立的、可复用的部件，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（即 "props"），并返回用于描述页面展示内容的 React 元素

5.1 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

```
function Welcome(props: Props): React.ReactElement {
    return <h1>Hello, {props.name}h1>;
}
```

5.2 类(定义的)组件

```
class Welcome extends React.Component<Props> {
    render(): React.ReactElement {
        return <h1>Hello, {this.props.name}h1>;
    }
}
```

5.3 组件渲染

- 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
- 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

```
import React from 'react';
import ReactDOM from 'react-dom';
let root: HTMLElement | null = document.getElementById('root');
interface Props {
    name: string;
}
function Welcome(props: Props): React.ReactElement {
    return <h1>Hello, {props.name}h1>;
}
class Welcome2 extends React.Component<Props> {
    render(): React.ReactElement {
        return <h1>Hello, {this.props.name}h1>;
    }
}

const element1: React.ReactElement> = <Welcome name="zhufeng" />;
console.log(element1.props.name);
const element2: React.ReactElement> = <Welcome2 name="zhufeng" />;
console.log(element1.props.name);

ReactDOM.render(
    <div>{element1}{element2}div>,
    root
);
```

5.4 复合组件 & 提取组件

- 组件由于嵌套变得难以被修改，可复用的部分也难以被复用，所以可以把大组件切分为更小的组件
- 当你的UI中有一部分重复使用了好几次(比如 Button、 Panel、 Avatar)，或者其自身就足够复杂(比如，App)，类似这些都是抽象成可复用组件的绝佳选择

```
import React from 'react';
import ReactDOM from 'react-dom';
let root: HTMLElement | null = document.getElementById('root');

interface PanelProps {
    header: string;
    body: string;
}
interface HeaderProps {
    header: string;
}
interface BodyProps {
    body: string;
}
class Panel extends React.Component<PanelProps> {
    render(): React.ReactElement {
        let { header, body } = this.props;
        return (
            <div style={{ border: '1px solid red', padding: 5 }}>
                <div className="panel-default panel">
                    <Header header={header}>Header>
                    <Body body={body} />
                div>
            div>
        )
    }
}
class Header extends React.Component<HeaderProps> {
    render(): React.ReactElement {
        return (
            <div style={{ border: '1px solid green' }}>
                {this.props.header}
            div>
        )
    }
}
class Body extends React.Component<BodyProps> {
    render(): React.ReactElement {
        return (
            <div style={{ border: '1px solid blue' }}>
                {this.props.body}
            div>
        )
    }
}

let data: PanelProps = { header: '头部', body: '身体' };
ReactDOM.render(<Panel {...data} />, root);
```

5.5 Props的只读性

- 无论是使用函数或是类来声明一个组件，它决不能修改它自己的 props
- 纯函数没有改变它自己的输入值，当传入的值相同时，总是会返回相同的结果
- 所有的React组件必须像纯函数那样使用它们的props

```
function sum(a, b) {
  return a + b;
}

function withdraw(account, amount) {
  account.total -= amount;
}
```

5.6 类型检查

- 要在组件的 props 上进行类型检查，你只需配置特定的 propTypes 属性
- 您可以通过配置特定的 defaultProps 属性来定义 props 的默认值：

用法 含义 PropTypes.array 数组 PropTypes.bool 布尔类型 PropTypes.func 函数 PropTypes.number 数字 PropTypes.object 对象 PropTypes.string 字符串 PropTypes.symbol Symbol PropTypes.node 任何可被渲染的元素(包括数字、字符串、元素或数组) PropTypes.element 一个 React 元素 PropTypes.instanceOf(Message) Message类的实例 PropTypes.oneOf(['News', 'Photos']) 枚举类型 PropTypes.oneOfType([PropTypes.string,PropTypes.number,PropTypes.instanceOf(Message)]) 几种类型中的任意一个类型 PropTypes.arrayOf(PropTypes.number) 一个数组由某一类型的元素组成

PropTypes.objectOf(PropTypes.number) 可以指定一个对象由某一类型的值组成 PropTypes.shape({color: PropTypes.string,fontSize: PropTypes.number}) 可以指定一个对象由特定的类型值组成 PropTypes.func.isRequired 你可以在任何 PropTypes 属性后面加上 `isRequired`

，确保这个 prop 没有被提供时，会打印警告信息 PropTypes.any.isRequired 任意类型的数据 customProp: function(props, propName, componentName){} 你可以指定一个自定义验证器。它在验证失败时应返回一个 Error 对象

```
import React from 'react';
import ReactDOM from 'react-dom';
import PropTypes from 'prop-types';
interface PersonProps {
    name?: string;
    age?: number,
    gender?: 'male' | 'female',
    hobby?: Array,
    position?: { x: number, y: number },
    friends?: Arrayname: string, age: number }>,
    [prop: string]: any
}
let root: HTMLElement | null = document.getElementById('root');
class Person extends React.Component<PersonProps> {
    static defaultProps: PersonProps = {
        name: 'Stranger'
    }
    static propTypes = {
        name: PropTypes.string.isRequired,
        gender: PropTypes.oneOf(['male', 'female']),
        hobby: PropTypes.arrayOf(PropTypes.string),
        position: PropTypes.shape({
            x: PropTypes.number,
            y: PropTypes.number
        }),
        age(props: PersonProps, propName: string, componentName: string) {
            let age = props[propName];
            if (age < 0 || age > 120) {
                return new Error(`Invalid Prop ${propName} supplied to ${componentName}`)
            }
        },
        friends: PropTypes.arrayOf((propValue: any, key: string, componentName: string, location: string, propFullName: string): Error | null => {
            console.log('propValue=' + JSON.stringify(propValue, null, 2), "key=" + key, "componentName=" + componentName, "location=" + location,
"propFullName=" + propFullName);

            let age = propValue[key].age;
            if (age < 0 || age > 120) {
                return new Error(
                    'Invalid prop `' + propFullName + '.age` supplied to' +
                    ' `' + componentName + '`. Validation failed.'
                );
            }
            return null;
        })
    }
    render() {
        let { name, age, gender, hobby, position } = this.props;
        return (
            <table>
                <thead>
                    <tr>
                        <td>姓名td>
                        <td>年龄td>
                        <td>性别td>
                        <td>爱好td>
                        <td>位置td>
                    tr>
                thead>
                <tbody>
                    <tr>
                        <td>{name}td>
                        <td>{age}td>
                        <td>{gender}td>
                        <td>{hobby!.join(',')}td>
                        <td>{position!.x + ' ' + position!.y}td>
                    tr>
                tbody>
            table>
        )
    }
}
let personProps: PersonProps = {
    age: 80,
    gender: 'male',
    hobby: ['basketball', 'football'],
    position: { x: 10, y: 10 },
    friends: [{ name: 'zhangsan', age: 10 }, { name: 'lisi', age: -20 }]
}
ReactDOM.render(<Person {...personProps} />, root);
```

6. 虚拟DOM

6.1 src\index.tsx

src\index.tsx

```
import React from './react';
import ReactDOM from './react-dom';
import { FunctionComponentElement, ComponentElement } from './typings';

interface Props {
    title: string
}

class Welcome extends React.Component {
    render() {
        return React.createElement('h1', { className: 'title' }, this.props.title);
    }
}
let element: ComponentElement = React.createElement(Welcome, { title: '标题' }) as ComponentElement;
ReactDOM.render(element, document.getElementById('root') as HTMLElement);
```

6.2 src\react.tsx

src\react.tsx

```
import { FunctionComponent, ComponentClass, ReactElement, PropsWithChildren, Component } from './typings';

export function createElement<P>(type: string | FunctionComponent | ComponentClass, config: P, ...children: Array): ReactElement<P> {
    let props = { ...config, children };
    const element: ReactElement, FunctionComponent> | ComponentClass> | string> = {
        type: type, props,
    };
    return element;
}

export default {
    createElement, Component
}
```

6.3 src\typings.tsx

src\typings.tsx

```
export interface Component {
    render(): ReactElement
}
export class Component<P = any> {
    static isReactComponent: boolean = true;
    constructor(public props: P) {
        this.props = props;
    }
}

export type JSXElementConstructor = ((props: P) => ReactElement) | (new (props: P) => Component);

export interface ReactElement = string | JSXElementConstructor> {
    type: T;
    props: P;
}
export type PropsWithChildren = P & { children?: Array };
export interface FunctionComponent {
    (props: PropsWithChildren): ReactElement;
}

export interface ComponentClass {
    new(props: P, context?: any): Component;
}
export interface FunctionComponentElement extends ReactElement> {

}
export interface ComponentElement extends ReactElement> {

}
export interface CSSProperties extends Record {
    [key: string]: any
}
export interface HTMLAttributes {
    className?: string;
    style?: CSSProperties;
}
```

6.4 src\react-dom.tsx

src\react-dom.tsx

```
import { FunctionComponent, ComponentClass, ReactElement, PropsWithChildren, HTMLAttributes } from './typings';
function render<P>(element: ReactElement, FunctionComponent> | ComponentClass> | string> | string, container: HTMLElement) {
    if (typeof element == 'string') {
        return container!.appendChild(document.createTextNode(element))
    }
    let type, props: HTMLAttributes & P & Record;
    type = element.type;
    props = element.props;
    if ((type as any).isReactComponent) {
        element = new (type as ComponentClass>)(props as PropsWithChildren).render();
        type = element.type;
        props = element.props;
    } else if (typeof type == 'function') {
        element = (type as FunctionComponent)(props);
        type = element.type;
        props = element.props;
    }
    let domElement = document.createElement(type as string);
    for (let propName in props) {
        if (propName === 'children') {
            let children: Array | undefined = props.children;
            if (children) {
                children.forEach((child: ReactElement | string) => render(child, domElement));
            }
        } else if (propName === 'className') {
            if (props.className)
                domElement.className = props.className;
        } else if (propName === 'style') {
            let styleObj = props.style;

            if (styleObj) {
                let cssText = Object.keys(styleObj).map(attr => {
                    return `${attr.replace(/([A-Z])/g, function () { return "-" + arguments[1] })}:${styleObj![attr]}`;
                }).join(';');
                domElement.style.cssText = cssText;
            }

        } else {
            domElement.setAttribute(propName, props[propName]);
        }
    }
    container!.appendChild(domElement);
}
export default { render };
```

6. 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(默认属性，属性校验)
- 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

```
import React from 'react';
import ReactDOM from 'react-dom';
interface Props {

}
interface State {
    date: any
}
class Clock extends React.Component<Props, State>{
    timerID
    constructor(props) {
        super(props);
        this.state = { date: new Date() };
    }

    componentDidMount() {
        this.timerID = setInterval(
            () => this.tick(),
            1000
        );
    }

    componentWillUnmount() {
        clearInterval(this.timerID);
    }

    tick() {
        this.setState({
            date: new Date()
        });
    }

    render() {
        return (
            <div>
                <h1>Hello, world!h1>
                <h2>It is {this.state.date.toLocaleTimeString()}h2>
            div>
        );
    }
}

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

6.1 不要直接修改 State

- 构造函数是唯一可以给 this.state 赋值的地方

```
import React from 'react';
import ReactDOM from 'react-dom';
interface Props {
}
interface State {
    number: number
}
class Counter extends React.Component<Props, State> {
    timerID
    constructor(props) {
        super(props);
        this.state = {
            number: 0
        };
    }

    componentDidMount() {
        this.timerID = setInterval(
            () => {
                this.setState({ number: this.state.number + 1 });

            },
            1000
        );
    }

    componentWillUnmount() {
        clearInterval(this.timerID);
    }

    render() {
        return (
            <div >
                <p> {this.state.number} p>
            div>
        );
    }
}

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

6.2 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

```
import React from 'react';
import ReactDOM from 'react-dom';
interface Props {
}
interface State {
    number: number
}
class Counter extends React.Component<Props, State> {
    constructor(props) {
        super(props);
        this.state = {
            number: 0
        };
    }
    handleClick = () => {

        this.setState((state) => (
            { number: state.number + 1 }
        ));
        this.setState((state) => (
            { number: state.number + 1 }
        ));
    }
    render() {
        return (
            <div >
                <p> {this.state.number} p>
                <button onClick={this.handleClick}>+button>
            div>
        );
    }
}

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

6.3 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

```
import React from 'react';
import ReactDOM from 'react-dom';
interface Props {

}
interface State {
    name: string;
    number: number
}
class Counter extends React.Component<Props, State> {
    constructor(props) {
        super(props);
        this.state = {
            name: 'zhufeng',
            number: 0
        };
    }
    handleClick = () => {

        this.setState((state) => (
            { number: state.number + 1 }
        ));
        this.setState((state) => (
            { number: state.number + 1 }
        ));
    }
    render() {
        return (
            <div >
                <p>{this.state.name}: {this.state.number} p>
                <button onClick={this.handleClick}>+button>
            div>
        );
    }
}

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

6.4 数据是向下流动的

- 不管是父组件或是子组件都无法知道某个组件是有状态的还是无状态的，并且它们也并不关心它是函数组件还是 class 组件
- 这就是为什么称 state 为局部的或是封装的的原因,除了拥有并设置了它的组件，其他组件都无法访问
- 任何的 state 总是所属于特定的组件，而且从该 state 派生的任何数据或 UI 只能影响树中"低于"它们的组件
- 如果你把一个以组件构成的树想象成一个 props 的数据瀑布的话，那么每一个组件的 state 就像是在任意一点上给瀑布增加额外的水源，但是它只能向下流动

```
import React from 'react';
import ReactDOM from 'react-dom';
interface Props {

}
interface State {
    name: string;
    number: number
}
class Counter extends React.Component<Props, State> {
    constructor(props) {
        super(props);
        this.state = {
            name: 'zhufeng',
            number: 0
        };
    }
    handleClick = () => {
        this.setState((state) => (
            { number: state.number + 1 }
        ));
    }
    render() {
        return (
            <div style={{ border: '1px solid red' }}>
                <p>{this.state.name}: {this.state.number} p>
                <button onClick={this.handleClick}>+button>
                <SubCounter number={this.state.number} />
            div>
        );
    }
}
class SubCounter extends React.Component<{ number: number }> {
    render() {
        return <div style={{ border: '1px solid blue' }}>子计数器:{this.props.number}div>;
    }
}
ReactDOM.render(
    <Counter />,
    document.getElementById('root')
);
```

7. 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`

```
import React from 'react';
import ReactDOM from 'react-dom';
class Link extends React.Component {
    handleClick(e: React.MouseEvent) {
        e.preventDefault();
        alert('The link was clicked.');
    }

    render() {
        return (
            <a href="http://www.baidu.com" onClick={this.handleClick}>
                Click me
          a>
        );
    }
}

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

7.2 this

- 你必须谨慎对待 JSX 回调函数中的 this,可以使用:
    - 公共属性(箭头函数)
    - 匿名函数
    - bind进行绑定

```
class LoggingButton extends React.Component {
    handleClick() {
        console.log('this is:', this);
    }
    handleClick1 = () => {
        console.log('this is:', this);
    }
    render() {

        return (
            <button onClick={(event) => this.handleClick(event)}>
                Click me
        button>
        );
    }
}
```

7.3 向事件处理程序传递参数

- 匿名函数
- bind

```
import React from 'react';
import ReactDOM from 'react-dom';
class LoggingButton extends React.Component {
    handleClick = (id, event: React.MouseEvent) => {
        console.log('id:', id);
    }
    render() {
        return (
            <>
                <button onClick={(event) => this.handleClick('1', event)}>
                    Click me
                button>
                <button onClick={this.handleClick.bind(this, '1')}>
                    Click me
                button>
                </>
        );
    }
}
ReactDOM.render(
    <LoggingButton />,
    document.getElementById('root')
);
```

7.4 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
- 在 React 渲染生命周期时，表单元素上的 value 将会覆盖 DOM 节点中的值，在非受控组件中，你经常希望 React 能赋予组件一个初始值，但是不去控制后续的更新。 在这种情况下, 你可以指定一个 defaultValue 属性，而不是 `value`

7.4.1 ref的值是一个字符串

```
import React from 'react';
import ReactDOM from 'react-dom';
class Sum extends React.Component {
    handleAdd = (event: React.MouseEvent) => {
        let a = (this.refs.a as HTMLInputElement).value;
        let b = (this.refs.b as HTMLInputElement).value;
        (this.refs.c as HTMLInputElement).value = a + b;
    }

    render() {
        return (
            <>
                <input ref="a" />+<input ref="b" /><button onClick={this.handleAdd}>=button><input ref="c" />
            </>
        );
    }
}
ReactDOM.render(
    <Sum />,
    document.getElementById('root')
);
```

7.4.2 ref的值是一个函数

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

7.4.3 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

```
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')
);
```

7.4.4 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

```
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')
);
```

7.4.5 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（换句话说，"转发"它）给子组件

```
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>
          </>
      );
  }
}

function TextInput() {
  return <input />
}
ReactDOM.render(
  <Form />,
  document.getElementById('root')
);
```

使用forwardRef

```
import React from 'react';
import ReactDOM from 'react-dom';
interface InputProps { }
const TextInput = React.forwardRef((props: InputProps, ref: React.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')
);
```

8.实现

8.1 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);

    }
    render() {
        return (
            <div>
                <p>number:{this.state.number}p>
                <button onClick={this.handleClick}>+button>
            div>
        )
    }
}
ReactDOM.render(<Counter title="计数器" />, document.getElementById('root'));
```

8.2 react\index.js

src\react\index.js

```
import { _render } from '../react-dom';
import { useDebugValue } from 'react';
function createElement(type, config, ...children) {
    let props = { ...config, children };
    let element = { type, props };
    return element;
}
function renderComponent(componentInstance) {
    const renderElement = componentInstance.render();
    const newDOM = _render(renderElement.type, renderElement.props, componentInstance);
    componentInstance.dom.parentNode.replaceChild(newDOM, componentInstance.dom);
    componentInstance.dom = newDOM;
}
class Component {
    static isReactComponent = true
    constructor(props) {
        this.props = props;
        this.updateQueue = [];
        this.isBatchUpdate = false;
    }
    setState = (partialState) => {
        this.updateQueue.push(partialState);
        if (!this.isBatchUpdate)
            this.flushState();
    }
    flushState = () => {
        this.state = this.updateQueue.reduce((acc, cur) => {
            acc = { ...acc, ...cur };
            return acc;
        }, this.state);
        renderComponent(this);
    }
}

export default {
    createElement,
    Component
}
```

8.3 react-dom\index.js

src\react-dom\index.js

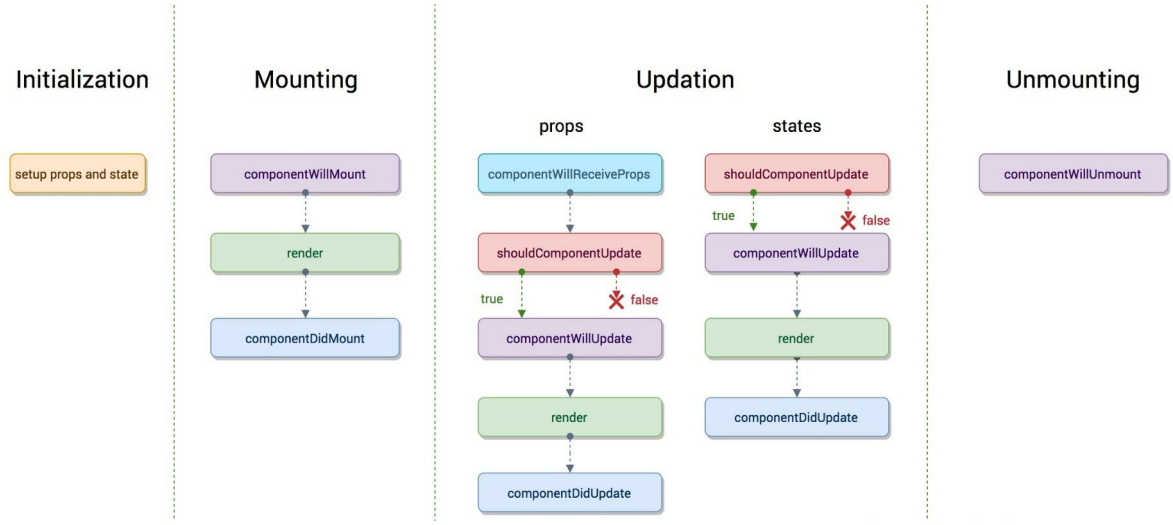
```
function render(element, parent, componentInstance) {
    if (typeof element === 'string' || typeof element === 'number') {
        return parent.appendChild(document.createTextNode(element));
    }
    let type = element.type, props = element.props;
    let isReactComponent = type.isReactComponent;
    if (isReactComponent) {
        componentInstance = new type(props);
        element = componentInstance.render();
        type = element.type;
        props = element.props;
    } else if (typeof type === 'function') {
        element = type(props);
        type = element.type;
        props = element.props;
    }
    let dom = _render(type, props, componentInstance);
    if (componentInstance && isReactComponent) {
        componentInstance.dom = dom;
    }
    parent.appendChild(dom);
}
class SyntheticEvent {
    constructor() {
        this.events = {};
    }
    on = (type, dom, handler, componentInstance) => {
        if (this.events[type]) {
            this.events[type].push({ dom, handler, componentInstance });
        } else {
            this.events[type] = [{ dom, handler, componentInstance }];
        }
    }
    emit = (type, event) => {
        let listeners = this.events[type];
        listeners && listeners.forEach(item => {
            if (event.target === item.dom) {
                if (item.componentInstance) {
                    item.componentInstance.isBatchUpdate = true;
                }
                item.handler(event);
                if (item.componentInstance) {
                    item.componentInstance.isBatchUpdate = false;
                    item.componentInstance.flushState();
                }
            }
        });

    }
}
let syntheticEvent = new SyntheticEvent();
document.onclick = syntheticEvent.emit.bind(null, 'onclick');
export function _render(type, props, componentInstance) {
    let dom = document.createElement(type);
    for (let propName in props) {
        if (propName === 'children') {
            let children = props.children;
            children.forEach(child => render(child, dom, componentInstance));
        } else if (propName === 'style') {
            let styleObj = props.style;
            for (let attr in styleObj)
                dom.style[attr] = styleObj[attr];
        } else if (propName === 'className') {
            dom.className = props.className
        } else if (propName.startsWith('on')) {

            syntheticEvent.on(propName.toLowerCase(), dom, props[propName], componentInstance);
        } else {
            dom.setAttribute(propName, props[propName]);
        }
    }
    return dom;
}
export default {
    render
}
```

8.生命周期

8.1 旧版生命周期

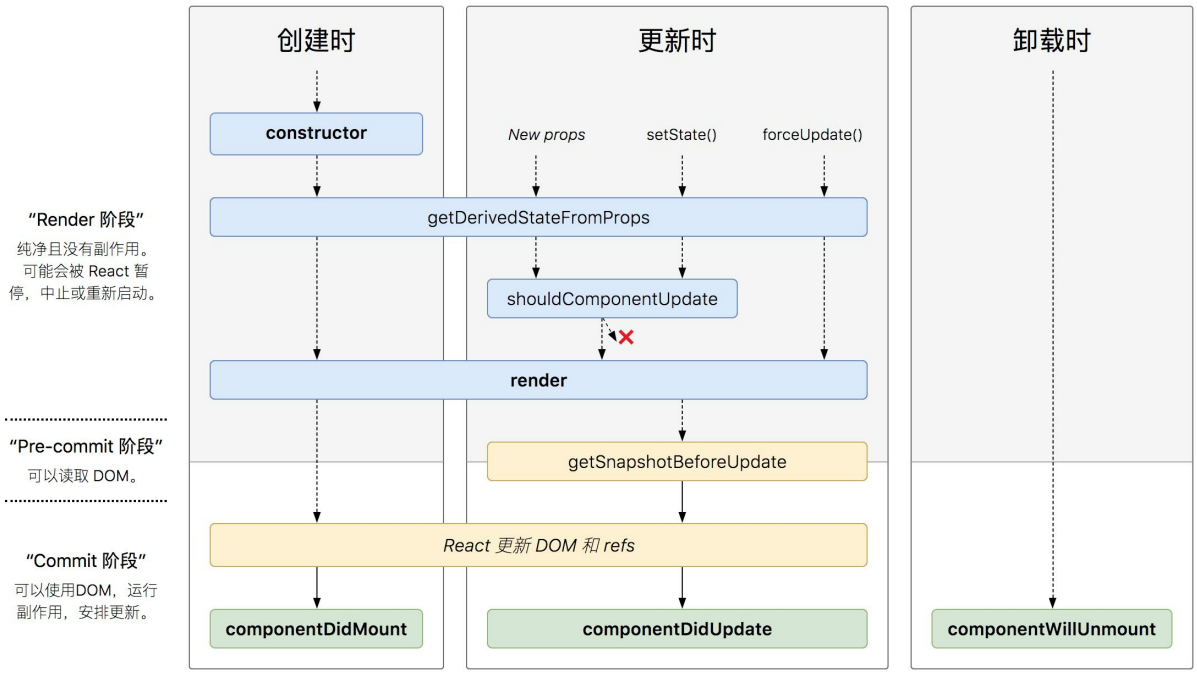



```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
interface Props { }
interface State { number: number }
class Counter extends React.Component<Props, State> {
    static defaultProps = {
        name: '珠峰架构'
    };
    constructor(props) {
        super(props);
        this.state = { number: 0 }
        console.log('1.constructor构造函数')
    }
    componentWillMount() {
        console.log('2.组件将要加载 componentWillMount');
    }
    componentDidMount() {
        console.log('4.组件挂载完成 componentDidMount');
    }
    handleClick = () => {
        this.setState({ number: this.state.number + 1 });
    };

    shouldComponentUpdate(nextProps, nextState): boolean {
        console.log('5.组件是否更新 shouldComponentUpdate');
        return nextState.number % 2 == 0;

    }
    componentWillUpdate() {
        console.log('6.组件将要更新 componentWillUpdate');
    }
    componentDidUpdate() {
        console.log('7.组件完成更新 componentDidUpdate');
    }
    render() {
        console.log('3.render');
        return (
            <div>
                <p>{this.state.number}p>
                {this.state.number > 3 ? null : <ChildCounter n={this.state.number} />}
                <button onClick={this.handleClick}>+button>
            div>
        )
    }
}
class ChildCounter extends Component<{ n: number }> {
    componentWillUnmount() {
        console.log('组件将要卸载componentWillUnmount')
    }
    componentWillMount() {
        console.log('child componentWillMount')
    }
    render() {
        console.log('child-render')
        return (<div>
            {this.props.n}
        div>)
    }
    componentDidMount() {
        console.log('child componentDidMount')
    }
    componentWillReceiveProps(newProps) {
        console.log('child componentWillReceiveProps')
    }
    shouldComponentUpdate(nextProps, nextState) {
        return nextProps.n % 3 == 0;
    }
}
ReactDOM.render(<Counter />, document.getElementById('root'));
```

8.2 新版生命周期 #



8.2.1 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';
interface Props { }
interface State { number: number }
class Counter extends React.Component<Props, State> {
    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<{ number: number }, { number: number }> {
    constructor(props) {
        super(props);
        this.state = { number: 0 };
    }
    static getDerivedStateFromProps(nextProps, prevState) {
        const { number } = nextProps;

        if (number % 2 == 0) {
            return { number: number * 2 };
        } else {
            return { number: number * 3 };
        }

        return null;
    }
    render() {
        console.log('child-render', this.state)
        return (<div>
            {this.state.number}
        div>)
    }

}

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

8.2.2 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';
interface Props { }
interface State { messages: Array }
class ScrollingList extends React.Component<Props, State> {
    wrapper
    timeID
    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 this.wrapper.current.scrollHeight;
    }
    componentDidUpdate(pervProps, pervState, prevScrollHeight) {
        const curScrollTop = this.wrapper.current.scrollTop;

        this.wrapper.current.scrollTop = curScrollTop + (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')
);
```

参考

- react文档 (https://zh-hans.reactjs.org)
- react源码 (https://github.com/zhufengnodejs/react)