link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=54 sentences=276, words=1360

1. 原生JS实现计数器

```
<body>
<button id="counter-btn">
button>

<script>
let counterBtn = document.getElementById('counter-btn');
let number = 0;
counterBtn.addEventListener('click',function(){
    counterBtn.innerHTML = ++number;
});
script>
body>
```

2. HTML结构的复用

2.1 index.html

```
<body>
<div id="counter-app">div>
<script src="index.js">script>
<script>
[redacted]
```

2.2 index.js

```
class Counter{
  render(){
      return (
          `

            0

          `
      )
  }
}
```

3.生成DOM元素并添加事件

3.1 index.html

```

let counterApp = document.getElementById('counter-app');
counterApp.appendChild(new Counter().render());

```

3.2 index.js

```
class Counter{
  constructor(){
      this.state = {number:0};
  }
  createDOMFromString(domString){
    const div = document.createElement('div');
    div.innerHTML = domString;
    return div.children[0];
  }
  increment (){
    this.state = {number:this.state.number+1};
    let oldElement = this.domElement;
    let newElement = this.render();
    oldElement.parentElement.replaceChild(newElement,oldElement);
  }
  render(){
     this.domElement = this.createDOMFromString(`

        ${this.state.number}

     `);
     this.domElement.addEventListener('click',this.increment.bind(this));
     return this.domElement;
  }
}
```

4.抽象Component

4.1 index.html

```
<body>
<div id="counter-app">div>
<script src="index.js">script>
<script>
[redacted]
```

4.2 index.js

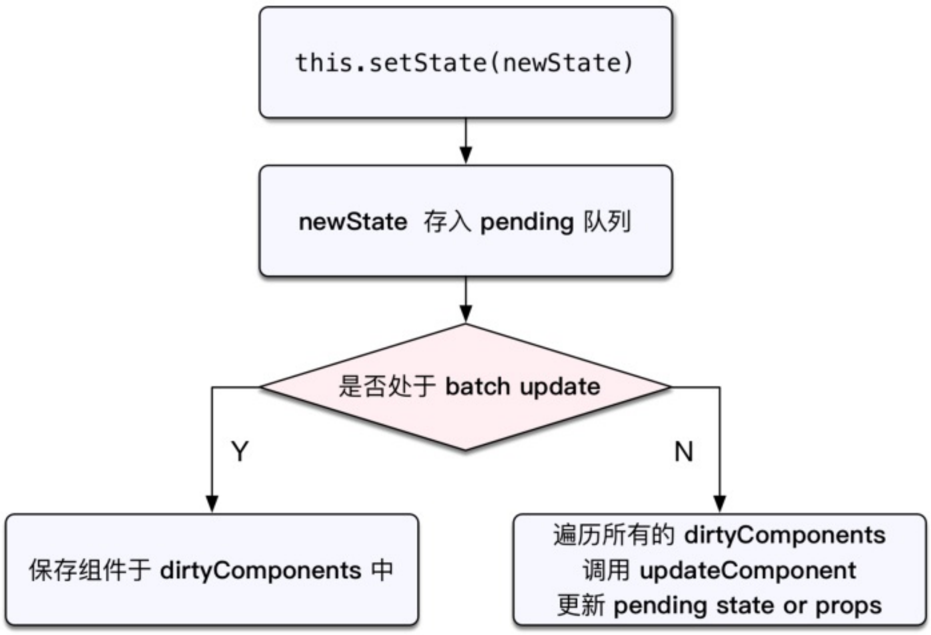
```
window.trigger = function(event,name){
    let component = event.target.component;
    component[name].call(component,event);
}
class Component{
    constructor(props){
        this.props = props;
    }
    createDOMFromString(domString){
        const div = document.createElement('div');
        div.innerHTML = domString;
        return div.children[0];
    }
    setState(partialState){
        this.state = Object.assign(this.state,partialState);
        let oldElement = this.domElement;
        let newElement = this.renderElement();
        oldElement.parentElement.replaceChild(newElement,oldElement);
    }
    renderElement(){
        let renderString = this.render();
        this.domElement = this.createDOMFromString(renderString);
        this.domElement.component = this;
        return this.domElement;
    }
    mount(container){
        container.appendChild(this.renderElement());
    }
}
class Counter extends Component{
  constructor(props){
      super(props);
      this.state = {number:0};
  }
  increment(){
   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);
   },1000);
  }
  render(){
      return (
        `

         ${this.props.name}:${this.state.number}

        `
      )
  }
}
```

5.setState可能是异步的

- 源码 (https://github.com/facebook/react/blob/35962a00084382b49d1f9e3bd36612925f360e5b/src/renderers/shared/reconciler/ReactUpdates.js#L199)


this.setState(newState)
newState 存入 pending 队列
是否处于 batch update
Y
N
保存组件于 dirtyComponents 中
遍历所有的 dirtyComponents
调用 updateComponent
更新 pending state or props

5.1 index.html

```
let counterApp = document.getElementById('counter-app');
new Counter({name:'珠峰架构'}).mount(counterApp);
```

5.2 index.js [#]()

```
let batchingStrategy = {
    isBatchingUpdates:false,
    updaters:[],
    batchedUpdates(){
        this.updaters.forEach(updater => {
            updater.component.updateComponent();
        });
    }
}
class Updater{
    constructor(component){
        this.component = component;
        this.pendingStates = [];
    }
    addState(particalState){
        this.pendingStates.push(particalState);
        batchingStrategy.isBatchingUpdates?batchingStrategy.updaters.push(this):this.component.updateComponent();
    }
}
let transaction = new Transaction({
    initialize() {
        batchingStrategy.isBatchingUpdates = true;
    },
    close() {
        batchingStrategy.isBatchingUpdates = false;
        batchingStrategy.batchedUpdates();
    }
});
window.trigger = function(event,name){
    batchingStrategy.isBatchingUpdates = true;
    let component = event.target.component;
    component[name].bind(component,event);
    batchingStrategy.isBatchingUpdates = false;
    batchingStrategy.batchedUpdates();
}
class Component{
    constructor(props){
        this.props = props;
        this.$updater = new Updater(this);
    }
    createDOMFromString(domString){
        const div = document.createElement('div');
        div.innerHTML = domString;
        return div.children[0];
    }
    setState(particalState){
        this.$updater.addState(particalState);
    }
    updateComponent(){
        let pendingStates = this.$updater.pendingStates;
        pendingStates.forEach(particalState=>Object.assign(this.state,particalState));
        this.$updater.pendingStates.length = 0;
        let oldElement = this.domElement;
        let newElement = this.renderElement();
        oldElement.parentElement.replaceChild(newElement,oldElement);
    }
    renderElement(){
        let renderString = this.render();
        this.domElement = this.createDOMFromString(renderString);
        this.domElement.component = this;
        return this.domElement;
    }
    mount(container){
        container.appendChild(this.renderElement());
    }
}
class Counter extends Component{
  constructor(props){
      super(props);
      this.state = {number:0};
  }
  increment(){
   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);
   },1000);
  }
  render(){
      return (
        `

         ${this.props.name}:${this.state.number}

        `
      )
  }
}
```

6.事务 [#]()

- [源码 (https://github.com/facebook/react/blob/6d5fe44c8602f666a043a4117ccc3bdb29b86e78/src/shared/utils/Transaction.js)](https://github.com/facebook/react/blob/6d5fe44c8602f666a043a4117ccc3bdb29b86e78/src/shared/utils/Transaction.js)
- 一个所谓的 Transaction 就是将需要执行的 method 使用 wrapper 封装起来，再通过 Transaction 提供的 perform 方法执行
- 而在 perform 之前，先执行所有 wrapper 中的 initialize 方法；perform 完成之后（即 method 执行后）再执行所有的 close 方法
- 一组 initialize 及 close 方法称为一个 wrapper

```
*                       wrappers (injected at creation time)
 *                                      +        +
 *                                      |        |
 *                    +-----------------|--------|--------------+
 *                    |                 v        |              |
 *                    |      +---------------+   |              |
 *                    |   +--|    wrapper1   |---|----+         |
 *                    |   |  +---------------+   v    |         |
 *                    |   |          +-------------+  |         |
 *                    |   |     +----|   wrapper2  |--------+   |
 *                    |   |     |    +-------------+  |     |   |
 *                    |   |     |                     |     |   |
 *                    |   v     v                     v     v   | wrapper
 *                    | +---+ +---+   +---------+   +---+ +---+ | invariants
 * perform(anyMethod) | |   | |   |   |         |   |   | |   | | maintained
 * +----------------->|-|---|-|---|-->|anyMethod|---|---|-|---|-|-------->
 *                    | |   | |   |   |         |   |   | |   | |
 *                    | |   | |   |   |         |   |   | |   | |
 *                    | |   | |   |   |         |   |   | |   | |
 *                    | +---+ +---+   +---------+   +---+ +---+ |
 *                    |  initialize                    close    |
 *                    +-----------------------------------------+
 *
```

6.1 transaction [#]

```
function setState() {
    console.log('setState')
}
class Transaction {
    constructor(wrappers) {
        this.wrappers = wrappers;
    }
    perform(func) {
        this.wrappers.forEach(wrapper=>wrapper.initialize())
        func.call();
        this.wrappers.forEach(wrapper=>wrapper.close())
    }

}
let transaction = new Transaction([
    {
        initialize() {
            console.log('before1');
        },
        close() {
            console.log('after1');
        }
    },
    {
        initialize() {
            console.log('before2');
        },
        close() {
            console.log('after2');
        }
    }
]);
transaction.perform(setState);
```

6.2 index.js [#]

```
class Transaction {
    constructor(wrapper){
        this.wrapper = wrapper;
    }
    perform(func){
        this.wrapper.initialize();
        func.call();
        this.wrapper.close();
    }

}
let batchingStrategy = {
    isBatchingUpdates:false,
    updaters:[],
    batchedUpdates(){
        this.updaters.forEach(updater => {
            updater.component.updateComponent();
        });
    }
}
class Updater{
    constructor(component){
        this.component = component;
        this.pendingStates = [];
    }
    addState(particalState){
        this.pendingStates.push(particalState);
        batchingStrategy.isBatchingUpdates?batchingStrategy.updaters.push(this):this.component.updateComponent();
    }
}
let transaction = new Transaction({
    initialize() {
        batchingStrategy.isBatchingUpdates = true;
    },
    close() {
        batchingStrategy.isBatchingUpdates = false;
        batchingStrategy.batchedUpdates();
    }
});
window.trigger = function(event,name){
    let component = event.target.component;
    transaction.perform(component[name].bind(component,event));
}
class Component{
    constructor(props){
        this.props = props;
        this.$updater = new Updater(this);
    }
    createDOMFromString(domString){
        const div = document.createElement('div');
        div.innerHTML = domString;
        return div.children[0];
    }
    setState(particalState){
        this.$updater.addState(particalState);
    }
    updateComponent(){
        let pendingStates = this.$updater.pendingStates;
        pendingStates.forEach(particalState=>Object.assign(this.state,particalState));
        this.$updater.pendingStates.length = 0;
        let oldElement = this.domElement;
        let newElement = this.renderElement();
        oldElement.parentElement.replaceChild(newElement,oldElement);
    }
    renderElement(){
        let renderString = this.render();
        this.domElement = this.createDOMFromString(renderString);
        this.domElement.component = this;
        return this.domElement;
    }
    mount(container){
        container.appendChild(this.renderElement());
    }
}
class Counter extends Component{
  constructor(props){
      super(props);
      this.state = {number:0};
  }
  increment(){
   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);
   },1000);
  }
  render(){
      return (
        `

         ${this.props.name}:${this.state.number}

        `
      )
  }
}
```